The LaTeX Graphics Companion

Illustrating documents with TeX and PostScript

Michel Goossens
CERN, Geneva, Switzerland

Sebastian Rahtz
Elsevier Science Ltd, Oxford, United Kingdom

Frank Mittelbach
LaTeX3 Project, Mainz, Germany

3rd printing April 1999

Addison-Wesley

An imprint of Addison Wesley Longman, Inc.
Reading, Massachusetts • Harlow, England • Menlo Park, California
Berkeley, California • Don Mills, Ontario
Sydney • Bonn • Amsterdam • Tokyo • Mexico City

Библиотека издательских технологий

М. Гуссенс
С. Ратц
Ф. Миттельбах

Путеводитель по пакету LaTeX и его графическим расширениям

Иллюстрирование документов при помощи TeX'а и PostScript'а

Перевод с английского под редакцией
А. И. Лебедева

«Мир»

«Бином ЛЗ»
Москва 2002

УДК 681.322
ББК 32.97
Г96

Переводчики: Лебедев А. И.— гл. 6, 10, 11, приложения A, B; Мусатов К. В.— предисловие, гл. 1–5, 7–9.

Гуссенс М., Ратц С., Миттельбах Ф.

Г96 Путеводитель по пакету LaTeX и его графическим расширениям. Иллюстрирование документов при помощи TeX'а и PostScript'а: Пер. с англ. — М.: Мир: Бином ЛЗ, 2002. — 621 с., ил.

ISBN 5-03-003388-2 («Мир»)
ISBN 5-94774-027-3 («Бином ЛЗ»)

Исчерпывающий справочник по стандартным графическим расширениям LaTeX'а, позволяющим сопроводить текст черно-белыми и цветными иллюстрациями высокого качества. Приводятся подробные описания пакетов XY-pic, PSTricks, METAPOST и PostScript-шрифтов, описываются средства для представления фейнмановских диаграмм, структурных химических формул, электрических схем, музыкальных партитур и настольных игр. Объясняется, как при помощи программы Ghostscript распечатывать PS-файлы на принтерах, не поддерживающих PostScript. Для упомянутых в книге пакетов приводятся адреса в Интернете.

Представляет интерес для всех TeX-пользователей: научных работников, самостоятельно готовящих свои работы к изданию, профессиональных наборщиков и дизайнеров, специалистов по издательским системам и студентов соответствующих специальностей.

УДК 681.322
ББК 32.97

Редакция автоматизации издательских процессов

ISBN 5-03-003388-2 («Мир»)
ISBN 5-94774-027-3 («Бином ЛЗ»)
ISBN 0-201-85469-4 (англ.)

Предисловие редактора перевода

Вы держите в руках пятую книгу серии «Библиотека издательских технологий», представляющую собой перевод одной из книг широко известной трилогии The (LaTeX, Web, Graphics) Companion. Ее авторы — Мишель Гуссенс, Себастиан Ратц и Франк Миттельбах — ведущие специалисты в этой области; они создали ряд полезных пакетов для системы TeX и немало сделали для ее популяризации.

Разработанная выдающимся математиком нашего времени Дональдом Кнутом система TeX располагает исключительными возможностями для форматирования документов, содержащих большое число математических формул. Д. Кнут, создавая пакет для издания своих работ, предоставил TeX во всеобщее пользование при условии сохранения неизменным тщательно отлаженного им кода ядра. Поэтому, в отличие от многих других издательских систем, для которых несовместимость различных версий — обыденное явление, всем построенным на основе TeX'а диалектам присуща полная совместимость.

Среди этих диалектов особое место занимает надстройка Лесли Лэмпорта, носящая имя LaTeX. Благодаря бóльшему удобству в использовании, эта система получила в настоящее время наиболее широкое распространение в научной среде. Практически во всех издательствах научно-технической литературы при приеме работ для публикации предпочтение отдается статьям, подготовленным в LaTeX'е.

Настоящая книга рассматривает задачу иллюстрирования документов, создаваемых с помощью LaTeX'а. Действительно, при работе с различными документами часто возникает необходимость не просто набрать текст с математическими формулами, но также включить в него различные иллюстрации, графики, диаграммы, порой цветные. Причем это касается не только печатных изданий, но и страниц для представления в World Wide Web и слайдов для презентаций на научных конференциях. Подготовка этих иллюстративных материалов может быть осуществлена как при помощи внешних графических редакторов, так и средствами самого TeX'а. Рассмотрению различных подходов к задаче создания иллюстраций средствами TeX'а и взаимной интеграции

их с текстом и математическими формулами и посвящена первая половина книги. Надо отметить, что наряду с «серьезными» приложениями для подготовки научных изданий в книге рассмотрены и довольно интересные узкоспециализированные подходы, используемые, например, для набора музыкальных партитур или шахматных партий.

Превосходное качество форматирования документов с помощью TEX'а часто ограничивается одним семейством шрифтов, поставляемым вместе с системой. Поэтому весьма важной практической задачей является оснащение TEX'а разнообразными шрифтами, среди которых особое значение придается профессионально разработанным PostScript-шрифтам в формате Type 1. Решение этой задачи, от простого подключения уже подготовленных шрифтовых пакетов до создания метрических файлов и виртуальных шрифтов, подробно разобрано во второй половине настоящей книги.

Среди рассматриваемых проблем большое место занимает вопрос о взаимодействии TEX'а с языком PostScript. Дело в том, что язык описания страниц PostScript обладает колоссальными возможностями и в настоящее время является основным языком, используемым в профессиональной издательской деятельности. Интерпретаторы этого языка встроены в большинство высокопроизводительных лазерных принтеров и фотонаборных аппаратов, что позволяет средствами PostScript'а создавать иллюстрации высочайшего качества.

Настоящая книга ориентирована на широкий круг читателей, как на тех, кто использует LATEX в своей повседневной работе, так и на тех, кто создает новые пакеты и программы для этой системы. Сотни пакетов, написанных для LATEX'а программистами-энтузиастами, создавались несогласованным образом, и поэтому объективно довольно трудно получить реальное представление о сравнительных возможностях различных пакетов. Настоящая книга как раз и является своеобразным путеводителем, авторы которой пытаются восстановить пробел, существующий в этой области. По построению книга напоминает справочник, но отличается от него тем, что его авторы, стоявшие у истоков развития LATEX'а и хорошо понимающие направления современного развития этой системы, осознанно отобрали наилучшие решения, использование которых наиболее перспективно. Стиль изложения традиционен для изданий этого рода: все основные идеи обсуждаются на огромном количестве примеров и иллюстраций.

В процессе работы над русским переводом в книгу был внесен ряд дополнений. Была добавлена информация о кодировках, используемых при наборе русских текстов (см. с. 440), внесены исправления, связанные с изменением положения различных файлов в архивах CTAN и в Интернете. В виде подстрочных замечаний отмечены некоторые важные достижения, такие, как появившаяся в последнее время возможность подключения к TEX'у шрифтов в формате TrueType и новые программы просмотра PDF-файлов. Многие из пакетов, описываемых в этом и других путеводителях по пакету LATEX, имеются на CD-ROM'е «TEX Live 6». Его модифицированная под нужды российского пользователя версия распространяется вместе в книгой М. Гуссенс, С. Ратц. «Путеводитель по пакету LATEX и его Web-приложениям». — М.: Мир, 2001.

С любезного согласия фирмы Pearson Computer Publishing для подготовки русского перевода авторы предоставили TEX'овский исходный файл по состоянию на апрель 1999 г. со всеми необходимыми стилями и примерами, в который были внесены авторские исправления по состоянию на октябрь 2000 г. Для того чтобы проиллюстрировать возможность использования описываемых в книге примеров с текстовыми вставками на русском языке, часть примеров была переведена. При этом следует иметь ввиду, что для работы с русифицированной версией и кириллическими шрифтами должны быть подгружены соответствующие пакеты:

inputenc пакет, входящий в базовую поставку LaTeX 2_ε и позволяющий поддерживать разные входные кодировки документа, в том числе и кириллические; задается в преамбуле, например так:
`\usepackage[cp866]{inputenc}`;

fontenc пакет, входящий в базовую поставку LaTeX 2_ε и подключающий шрифты в необходимой кодировке; задается в преамбуле, например так:
`\usepackage[LCY]{inputenc}`;

pscyr пакет кириллических PostScript-шрифтов; при необходимости задается в преамбуле:
`\usepackage{pscyr}`;

babel пакет, поддерживающий работу с многоязычными документами; задается в преамбуле:
`\usepackage[russian]{babel}`.

К сожалению, обнаружилось несколько программ, которые не смогли работать с русским языком, а относительно еще нескольких у нас так и не получилось полной ясности. В связи с этим читателю предоставляется уникальная возможность продолжить наши изыскания и выявить поддающиеся русификации пакеты самостоятельно. Еще одним моментом, о котором приходится сожалеть, стала борьба за экономичность русского издания, вследствие чего оригинал-макет перевода был подготовлен не с разрешением 2400 точек на дюйм, как английский оригинал, а 600 точек на дюйм. Поэтому оригинал-макет и шрифты русского издания существенно отличаются от английского оригинала.

Тем не менее коллектив редакторов и переводчиков выражает надежду, что им удалось адекватно передать дух и содержание английского оригинала и что этот путеводитель займет достойное место среди других справочных изданий на русском языке по LaTeX'у и позволит отечественному читателю в полной мере овладеть еще одной стороной этой блестящей системы, дарованной нам Дональдом Кнутом и Лесли Лэмпортом.

А. Лебедев

Предисловие

Сотням разработчиков LaTeX'а
за их энтузиазм и тяжелый труд

Особая благодарность нашим семьям
без чьего бесконечного терпения
эта книга никогда бы не вышла в свет

В данной книге описываются методы и приемы набора с помощью LaTeX'а, относящиеся к области графики и шрифтов. Показано, как рисовать средствами LaTeX'а и как включать графические файлы в LaTeX'овские документы. Разбирается, как можно программировать изображения, используя METAFONT и METAPOST, а также то, как добиваться специальных эффектов путем включения небольших фрагментов на языке PostScript. Демонстрируется, как манипулировать целыми TeX'овскими страницами в PostScript'е и как воспользоваться всеми возможностями PostScript'овских шрифтов Type 1. И, конечно, подробно рассматривается весь набор инструментов для создания графики внутри самого TeX'а.

TeX является мировым лидером среди разметочных систем типографского набора, а PostScript — лучшим языком описания печатной страницы. В книге описывается, как с помощью этих средств можно добиться еще лучших результатов, если использовать их совместно. Способность TeX'а выполнять математические вычисления, конструировать абзацы, находить точки переноса и его программируемость в сочетании с графической гибкостью и возможностями манипулирования шрифтами PostScript'а значительно расширяют возможности как автора, так и наборщика. В книге описываются драйверы из-dvi-в-PostScript, рассматриваются инструменты для манипулирования PostScript'овскими файлами и подробно обсуждаются характеристики свободно распространяемой программы Ghostscript, позволяющей просматривать и распечатывать PostScript'овские файлы даже в том случае, если у вас нет PostScript'овского принтера.

Эту книгу не следует считать руководством пользователя по применению программных пакетов. Стараясь учить на примерах, мы привели сотни наиболее полезных решений, опирающихся на хорошо известные и проверенные реализации. Но в рамках объема книги невозможно представить руководства каждого из пакетов. Главной целью было продемонстрировать то, насколько легко пользоваться данным пакетом и можно ли с его помощью добиться требуемых результатов, не вникая в детали синтаксиса и настройки при запуске. Тем не менее несколько пакетов, признанных особенно важными, описаны более подробно.

Предполагается, что читатель немного знаком с LaTeX'ом. Вы не сможете читать эту книгу, если никогда раньше не пользовались TeX'ом. Чтобы познакомиться с некоторыми доступными (неграфическими) пакетами, можно начать чтение с книги *LaTeX: A Document Preparation System, Second Edition* (Lamport, 1994) и продолжить книгой *The LaTeX Companion* (Goossens et al., 1994).[1]

Почему LaTeX и почему PostScript?

Эта книга посвящена LaTeX'у, графике и PostScript'у. Мы уверены, что структурированный подход системы LaTeX позволяет наилучшим способом реализовать возможности TeX'а, а LaTeX стал наиболее широко используемым TeX'овским форматом. Поскольку LaTeX привлекает разработчиков новых пакетов (особенно после его кардинального пересмотра в 1993 г.), кое-что из описанного будет работать только в LaTeX'е (особенно это относится к материалу, связанному со схемой выбора шрифтов). Тем, кто предпочитает элегантность оригинального формата plain TeX и его производных, мы заранее приносим свои извинения за смещение в сторону LaTeX'а, но обещаем, что большинство пакетов хорошо работают с любым диалектом TeX'а: достоинства таких систем, как METAPOST, PSTricks, XY-pic и MusiXTeX, доступны всем.

Нам хотелось бы также объяснить, почему мы будем так много говорить о PostScript'е. Этот язык, возникший более десяти лет назад как чрезвычайно гибкий язык описания страниц, продолжает оставаться наиболее популярным среди профессиональных наборщиков. Столь привлекательным его делают следующие особенности:

- большое число шрифтов Type 1, их высокое качество и гибкость;
- аппаратная независимость и мобильность файлов;
- качество графики и число графических пакетов для ее создания;
- средства манипулирования текстом;
- отлаженная технология цветной печати;
- соглашения об инкапсуляции, позволяющие легко вставлять PostScript'овскую графику;
- доступность приложений для просмотра (Display PostScript и Ghostscript/GhostView).

Несколько лет назад у PostScript'а появился потомок — язык PDF (Portable Document Format), используемый программой Adobe Acrobat. Разработанный

[1] Отечественному читателю можно также порекомендовать книги (Грэтцер, 1999), (Котельников, Чеботаев, 1998), и (Львовский, 1995). — *Прим. ред.*

для отображения на экране гипертекстовых документов, формат PDF поднимает стандарты мобильности и эффективности на новую высоту. Хотя в данной книге вопросы, связанные с PDF, не затрагиваются, LaTeX хорошо работает при создании «богатых» PDF-документов, а вскоре после выхода этой книги в свет, возможно, появятся специальные версии TeX'а, создающие документы непосредственно в формате PDF[1]

Мы еще раз приносим извинения тем читателям, кто испытает разочарование, не найдя ничего о связи LaTeX'а с QuickDraw, шрифтами TrueType, GDI для Windows, HPGL, PCL, и т. д., но при таком разнообразии доступных пакетов приходится от чего-то отказываться.

Пожалуйста, учтите, что отсутствие в данной книге конкретного пакета или приложения никоим образом не подразумевает, что мы считаем его менее полезным или менее качественным. Однако мы полагаем, что представленный набор пакетов и инструментов репрезентативен, и выражаем надежду, что каждый найдет для себя в этой книге одну-две интересные темы.

Как организована эта книга

Эту книгу можно разделить на части двумя способами: в соответствии с областью применения и с используемой техникой. Предполагается, что все читатели ознакомятся с гл. 1 перед тем, как переходить куда-либо дальше, поскольку в ней излагаются общие соображения, касающиеся графики, и обобщаются некоторые приемы, более подробно излагаемые в последующих главах. Мы также рекомендуем прочесть гл. 2, в которой описывается LaTeX'овский стандартный графический пакет, так как часто будет возникать потребность в инструментах для включения графических файлов.

Мы приложили все усилия для того, чтобы остальные главы можно было читать независимо. Вы можете перейти непосредственно к тем главам, которые относятся к интересующей вас теме или к тем, где описывается определенный пакет. Следующие три главы посвящены конкретным программам и пакетам:

3 В главе **Работа с METAFONT'ом и METAPOST'ом** показано, как использовать мощь разработанного Дональдом Кнутом языка METAFONT. Он хорошо взаимодействует с LaTeX'ом и предоставляет неповторимые выразительные средства для описания многих типов диаграмм.

4 В главе **Встраивание PostScript'а в LaTeX: пакет** PSTricks описано то, что, по нашему мнению, является наиболее мощной системой интеграции LaTeX'а с языком PostScript.

[1] Авторы оказались прозорливыми, и в настоящее время большинство реализации TeX'а включают пакеты pdftex и pdflatex, основным автором которых является Хан Те Тан. — *Прим. перев.*

5 В главе **Пакет** XY-pic описан пакет, который позволяет в большой степени определить обозначения для многих видов математических диаграмм и реализует их достаточно общим и мобильным способом.

Далее следуют три главы, в которых обсуждаются специализированные области применения и в каждой из которых приводится обзор существующих пакетов:

6 В главе **Применения в химии, физике и технике** рассматриваются вопросы построения структурных формул химических соединений, фейнмановских диаграмм и электрических схем.

7 В главе **Подготовка партитур** описывается мощный пакет MusiXTEX и ряд вспомогательных программ, которыми вы можете воспользоваться для печати прекрасных партитур.

8 Глава **Играя в игры** предназначена для тех, кто использует LaTeX не только для работы, но и для развлечения. В ней показано, как оформлять шахматные партии, изображать расклады в бридже и других карточных играх, и даже справиться с нардами и го.

В последних трех главах рассматриваются проблемы общего характера, в основном не связанные с конкретными LaTeX'овскими пакетами или предметными областями:

9 В главе **Мир цвета** излагаются детали, касающиеся стандартного LaTeX'овского цветового пакета, а также описываются другие пакеты, основанные на нем. Кроме того, затрагиваются вопросы цветового дизайна и проблемы, связанные с цветной печатью.

10 В главе **Использование шрифтов в формате PostScript** описываются все тонкости использования PostScript'овских шрифтов совместно с LaTeX'ом, включая AFM-файлы, виртуальные шрифты, наименования шрифтов и шрифты Multiple Master.

11 В главе **Драйверы и утилиты PostScript'а** рассмотрены некоторые из наиболее широко используемых свободно распространяемых программных пакетов — dvips, Ghostscript, набор утилит psutils и программы, преобразующие PostScript'овские шрифты в формат PK.

Наконец, приложения содержат техническую информацию, дополняющую предыдущие главы. В них также указано, где можно найти все описанное в книге программное обеспечение в Интернете, и приведены примеры, призванные помочь вам получить его разными способами.

Использование и поиск всех этих пакетов и программ

В книге описаны многие пакеты и опции, дополняющие или модифицирующие исходные возможности LaTeX'а. Для того чтобы показать их в действии, в

принципе, надо запустить их все одновременно. Однако по разным причинам это не практично, если не невозможно. Действительно, многие пакеты используют много внутренних переменных, число которых в большинстве реализаций TEX'а ограниченно. При подготовке этой книги применялась другая стратегия (опробованная при подготовке *The LATEX Companion*): почти все примеры запускались в виде отдельных заданий и включались в виде `eps`-файлов. Почти во всех случаях процесс был полностью автоматизирован — приводимый в книге код является именно тем, который использовался для подготовки вывода. Чтобы облегчить вам обработку примеров, каждый фрагмент LATEX'овского кода также содержит команды `\usepackage`, необходимые для его запуска.

Исходные тексты примеров можно найти на CTAN (Comprehensive TEX Archive Network (Полная TEX'овская архивная сеть) — см. Приложение B) в каталоге `info/lgc`. Большинство примеров имеет собственный номер, указанный в книге в рамке рядом с примером. Таким образом, оказываются доступными все исходные пакеты примеров, использованных для создания картинок, причем номер примера служит именем файла. Большинство этих файлов имеют формат текстов LATEX(с расширением `ltx`), но есть также файлы в формате plain TEX (расширение `ptx`), исходные файлы для METAPOST (расширение `mp`), исходные файлы препроцессора MusiXTEX (расширения `abc` и `mpp`), pic-файлы (расширение `pic`) и исходные файлы интерпретатора `m4` (расширение `m4`).

Все пакеты и программы, описанные в этой книге, свободно распространяются и могут быть получены из открытых архивов программного обеспечения. Некоторые относятся к категории public domain, другие защищены знаком авторского права. Некоторые программы доступны лишь в виде исходных текстов или же работают лишь на некоторых компьютерных платформах, поэтому вы должны быть готовы в отдельных случаях «испачкать руки». Мы не можем гарантировать, что следующие версии пакетов или программ будут приводить к результатам, идентичным приведенным в этой книге. Многие из них находятся в состоянии активной разработки и новые или измененные версии появляются несколько раз в году. Книга была завершена в конце 1996 года и примеры обрабатывались версиями, существовавшими на тот момент.

В приложении B помещено детальное описание того, как соединиться с сайтами CTAN и как получить файлы через Интернет. Кроме того, вы можете приобрести у TEX Users Group (группы пользователей TEX'а) CD-ROM, содержащий реализации TEX'а для различных компьютерных платформ, а также множество пакетов и шрифтов. Этот CD-ROM содержит все пакеты, описанные в данной книге и в *The LATEX Companion*.[1]

[1] Модифицированная под нужды российского пользователя версия CD-ROM «TEX Live 6» распространяется в качестве приложения к книге Гуссенс М., Ратц С. Путеводитель по пакету LATEX и его Web-приложениям. — М.: Мир, 2001. — *Прим. ред.*

Благодарности

У этой книги два источника. Первым послужили составленный Себастианом Ратцем обзор TEX'а и графики, первоначально подготовленный для конференции TEXeter в 1988 г., много раз пересматривавшийся и распространенный в 1989 г. в виде технического доклада факультетом информатики Саутхемптонского университета[1], а также техническая поддержка и документы по использованию PostScript'овских шрифтов совместно с LATEX'ом, которую Себастиан Ратц продолжает осуществлять. Вторым источником послужил материал, который Мишель Гуссенс, Франк Миттельбах и Александр Самарин не смогли включить в *The LATEX Companion*. Задолго до окончания работы над книгой стало ясно, что она не сможет в полной мере охватить графику и PostScript и что потребуется (по крайней мере) еще одна книга. Данная книга была задумана, создана и доработана в CERN'е на базе технической поддержки по вопросам обработки документов этой знаменитой лаборатории, осуществлявшейся Мишелем Гуссенсом, которому в 1993–1994 гг. помогал Себастиан Ратц.

Есть много людей, которых мы должны поблагодарить. Конечно, наибольшая благодарность выражается авторам описываемых программ и пакетов. Каждый автор, с которым мы связывались для обсуждения каких-либо проблем, оказывал ощутимую помощь в духе TEX'овского сообщества и часто разрешал использовать примеры из собственной документации.

Христель Миттельбах удалось существенно улучшить музыкальную главу, а Леонор Баррока пришлось потратить много времени на главу о METAFONT'е и METAPOST'е — без их помощи работа над книгой никогда не была бы закончена.

Мы выражаем признательность всем нашим многочисленным коллегам в TEX'овском мире — разработчикам программных пакетов, не только тех, что описаны здесь, но также и сотням других, кто помогает пользователям набирать свои документы быстрее и качественнее. Без постоянных усилий этих энтузиастов TEX не стал бы таким замечательным и гибким инструментом, каким он является сегодня. Мы постарались отдать им должное, указывая имена самого автора и/или других людей, внесших значительный вклад в разработку, при первом описании пакета.

Мы в большом долгу перед Двайтом Аплевичем, Барбарой Битон, Карлом Берри, Дэвидом Карлайлом, Малколмом Кларком, Петером Дойчем, Дени Жиру, Джоном Хобби, Аланом Хёнигом, Россом Муром, Торстеном Охлем, Крисом Роузом, Томом Рокицки, Крисом Роули, Веджилом Стоуксом, Джеффри Тобином, Ульриком Витом и Тимоти ван Зандтом за их терпение при чтении разных глав рукописи. Их многочисленные комментарии, предложения, исправления и подсказки существенно улучшили качество текста.

[1] Себастиан Ратц хотел бы поблагодарить профессора Дэвида Бэррона и ле Карра из Саутхемптонского университета за то, что они делились своим опытом, навыками и оказывали поддержку, когда мы все начали учить TEX десять лет назад.

Со стороны издательства мы хотим поблагодарить Питера Гордона, нашего редактора в Addison Wesley Longman, Inc., кто оказывал нам столь необходимую поддержку и воодушевлял на протяжении тех трех лет, что длился этот проект. Когда наступил производственный этап, Аванда Петерс и Роза Гонзалес с неизменным терпением относились к своеобразию наших стилей и довели работу до благополучного завершения. Катрина Авери, отредактировав, перевела наши туманные тексты на настоящий английский язык, а Арт Огава взял на себя труд повторного просмотра набора и цветных страниц. Мы очень благодарны им за их работу.

Обратная связь

Дорогие читатели, мы хотели бы попросить вас о помощи. Мы убедительно просим присылать любые комментарии, предложения или замечания любому из авторов, которые с удовольствием исправят все замеченные ошибки или недосмотры в будущем издании и всегда с удовольствием выслушают предложения по улучшению или включению важных разработок, не замеченных ими. Мы будем обновлять входящий в LaTeX'овский дистрибутив файл `grphcomp.err`, который содержит список замеченных опечаток, а также координаты авторов.

LaTeX и графика

Говорят, *каждая фотография — это целая история*. LaTeX умеет красиво расположить текст, но в руководствах по LaTeX'у обычно ничего или почти ничего не говорится о создании графического материала и включении его в ваш текст. В данной книге сделана попытка заполнить этот пробел, продемонстрировав средства и TeXнические приемы, разработанные за последнее десятилетие, которые позволят вам создавать, обрабатывать и включать графический материал в текст.

В эпоху мультимедийных персональных компьютеров графика может понадобиться где угодно. Вместе со многими программными продуктами поставляются готовые к использованию наборы рисунков, а в магазинах можно приобрести компакт-диски с «самыми лучшими фотографиями» интересных мест и т. д. Как мы увидим в этой и последующей главах, все изображения подобного рода могут быть включены в LaTeX'овский документ. Для этого они должны быть представлены в одном из подходящих форматов. К счастью, многие популярные графические форматы либо поддерживаются непосредственно, либо существуют конвертирующие программы, позволяющие нужным образом преобразовать изображения.

Если вы хотите самостоятельно стать иллюстратором, то часто пользуетесь специализированными графическими пакетами (например, xfig или idraw в операционной системе Unix, Adobe Illustrator, Corel Draw или FreeHand на Макинтоше или IBM-совместимом персональном компьютере и многими другими). Или, если вы работаете с современными вычислительными программами, такими, как Mathematica, Maple или MATLAB, то можете создавать изображения в каком-нибудь из их многочисленных графических выходных форматов. Если вы являетесь счастливым обладателем сканера или даже цифровой видеокамеры, то можете создавать оцифрованные копии фотографий, авторских рисунков или других графических объектов, используя эти устройства и сопутствующее программное обеспечение. Во всех этих случаях легко создать файлы, которые могут быть непосредственно включены в LaTeX'овский исходный документ с помощью пакета graphics, описанного в гл. 2.

Но во многих случаях требуется еще более тесное взаимодействие с системой набора, чем доступное в таких программах, и его тоже можно добиться с помощью LaTeX'а. Подобное взаимодействие необходимо, если вы хотите использовать одни и те же шрифты в тексте и в графике, или в более общем случае, если «стиль» графического материала должен зависеть от стиля всего документа. Тесная взаимосвязь графики с окружающим текстом очевидно требует создания графики средствами самой системы набора, поскольку в противном случае любое изменение стиля оформления документа потребует большого объема ручной работы и весь процесс будет сильно подвержен ошибкам.

* * *

В этой главе мы рассмотрим графические объекты под разными углами зрения. Мы начнем с различных требований, накладываемых на графические объекты в зависимости от используемого приложения. Затем проанализируем типы изображений, встречающиеся в документах, и стратегии, обычно используемые для создания, комбинирования и манипулирования такими объектами. После этого мы сменим тему и обсудим предоставляемые TeX'ом интерактивные средства для оперирования с графическими объектами и то, как эти средства правильно использовать.

Вооружившись этими знаниями, мы, наконец, совершим небольшое путешествие по графическим языкам, построенным на основе TeX'а и вокруг него. Все это поможет вам правильно выбрать из описанных в этой книге наборов средств то, которое соответствует стоящей перед вами задаче. На самом деле данная глава помимо всего перечисленного еще содержит примеры языков и приемов, которые больше нигде в книге подробно не рассматриваются. Это путешествие предоставит вам достаточно информации для принятия решения о том, следует ли воспользоваться приведенными указаниями и обзавестись таким пакетом для конкретной задачи.

1.1 Графические системы и типографский набор

Говоря о графических объектах, прежде всего нужно спросить себя, что мы понимаем под этим термином. Существует крайняя точка зрения, заключающаяся в том, чтобы рассматривать все, помещенное на бумагу, как графический объект, включая литеры используемых шрифтов. Эта весьма революционная точка зрения в действительности была воплощена при разработке языка описания страниц PostScript, в котором буквы можно набирать и манипулировать ими с помощью тех же самых функций, которые применяются к другим графическим объектам (мы увидим несколько примеров подобного рода в гл. 4).

Однако большинство систем типографского набора, включая TeX, не пытаются воплотить столь общую модель. Вместо этого они ограничивают свою область применения подмножеством графических объектов общего вида, например, предоставляя весьма сложные функции для размещения печатных

символов, разрешения лигатур и т. д., но исключая операторы для рисования произвольных линий, построения и закрашивания областей и т. п. В результате термин «графика» для большинства пользователей LaTeX'а служит синонимом слова «иллюстрация», а тот факт, что LaTeX уже обладает графическим языком — графическим режимом — часто упускается из виду.

При обсуждении графических возможностей идеальной системы типографского набора необходимо помнить, что различные приложения выдвигают разные, порой противоречивые требования:

- Одной крайностью является требование полной переносимости с одной платформы на другие; другой — учет даже таких деталей, как способ нанесения принтером краски на бумагу.
- Может потребоваться изменять размер изображения в зависимости от вида внешнего окружения, созданного системой типографского набора, например от ширины текста.
- Возможны ситуации, когда отдельные части изображения должны изменяться нелинейно, это может быть важно для сохранения удобочитаемости текстовых частей изображения, которые не должны становиться больше или меньше некоторого предела. Может также потребоваться, чтобы при увеличении размера изображения, скажем, на 10 процентов с целью соответствия строке любой содержащийся в нем текст не стал бы превосходить по размеру символы в основном теле документа.
- Может потребоваться, чтобы графический объект был тесно интегрирован с окружающим текстом, например, путем использования тех же шрифтов, что и в других частях документа, или в более общем случае путем включения объектов, изменяющих свой внешний вид при изменении общего стиля документа. (Последнее особенно важно, если документ описывается своим логическим содержанием, а не внешним обликом, с целью его повторного использования в разных контекстах и представлениях.)

Поскольку LaTeX является универсальной системой типографского набора, используемой во всевозможных приложениях, указанные выше и другие требования могут возникать в различных ситуациях. Как мы увидим в дальнейшем, большое количество таких ситуаций может быть разрешено если не идеально, то с большим изяществом. В некоторых случаях приемлемым решением оказываются наглядные и хорошо разработанные готовые макропакеты и программы, используемые уже целое десятилетие или даже дольше.

1.2 Типы графических объектов

Классификация графических объектов, с которой началась эта глава, сосредоточилась на вопросе интеграции с системой LaTeX и разделила графические объекты на созданные внешними и внутренними средствами. Другая перспектива может открыться, если начать с рассмотрения типов графических объ-

ектов, встречающихся в документах, и с обсуждения способов их создания и включения в документ. Поэтому давайте коротко перечислим различные типы изображений, которые могут встретиться в документе.

- Картинки могут быть нарисованы от руки без помощи компьютера, как, например, рисунок стеклянной бусины на рис. 1.1. Для использования в LaTeX'е такой рисунок должен быть преобразован в цифровую форму, например, с помощью сканера. Затем его можно вставить в документ и до некоторой степени преобразовывать, например, поворачивать с помощью команд, предоставляемых стандартным LaTeX'овским пакетом graphics, описанным в гл. 2. Подписи к рисунку и надписи на нем обычно создаются средствами самого LaTeX'а для обеспечения правильности ссылок внутри документа, но во всем остальном это изображение считается единым объектом, внутренняя структура которого недоступна, — «черным ящиком».
- «Художественная» графика, созданная на компьютере с помощью растровых средств, такая, как показано на рис. 1.2, до некоторой степени является компьютерным аналогом рисунков, выполненных пером и чернилами. Этот рисунок был сделан с помощью MacPaint: хорошо заметны низкое разрешение оригинала и слабая проработка диагональных линий. Характерной особенностью этого типа графических объектов является то, что выбранное в процессе создания разрешение не может быть легко изменено без ущерба качеству (или же без большого объема ручного труда). В остальных отношениях такое изображение сходно с рукотворным: обычно не возникает необходимости интегрировать рисунок с текстом или беспокоиться о согласованности гарнитур шрифтов — рисунок считается черным ящиком.
- Фотографии можно сканировать наподобие рисунков, выполненных от руки, в результате чего непрерывные тона фотографии преобразуются в индивидуальные цвета или оттенки серого из некоторого диапазона (черно-белые фотографии, полученные подобным способом, называются полутоновыми). Воспроизведение цвета в полном объеме требует сложных приемов печати, но эта проблема встает на стадии печати и обычно не влияет на сам типографский набор. Хотя качество изображения может быть улучшено или искажено (см. рис. 1.3), для TeXa такой объект остается черным ящиком.
- «Объектно-ориентированная» аппаратно-независимая художественная графика образует отдельный класс. Важная черта этого класса состоит в том, что изображение хранится в виде абстрактных объектов, не содержащих никакой зависящей от внешних устройств информации (в отличие от растровой графики, формат хранения которой содержит информацию о том, является ли данная точка черной или белой, делая тем самым эти объекты зависящими от разрешения). Эта независимость от устройств облегчает использование одного и того же изображения на разных устройствах вывода и, кроме того, позволяет манипулировать отдельными характеристиками изображения в процессе дизайна.

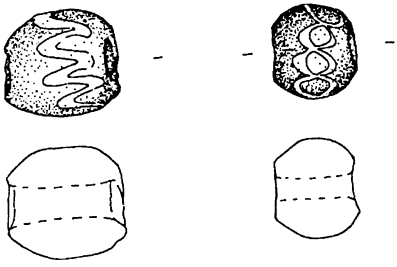


Рис. 1.1. Рисунок бусины, выполненный тушью.

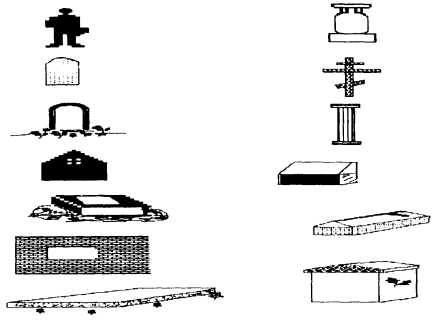


Рис. 1.2. Растровые изображения, созданные с помощью MacPaint.

Рис. 1.3. Изображение, подвергнутое цифровому преобразованию (растяжению по вертикали).

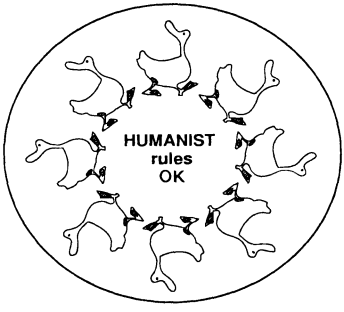


Рис. 1.4. Объектно-ориентированный рисунок.

На рис. 1.4, созданном с помощью Adobe Illustrator, изображение уток было получено путем многократного копирования и поворота изображения одной особи, нарисованной в виде набора кривых. Такой тип рисования часто имеет очень тесную связь с текстом и может содержать текстовые аннотации, соответствующие набираемому тексту. Хотя зачастую добавить текст в графический объект можно с помощью внешних программ, таких, как Illustrator, в общем случае использовать LaTeX для набора этого текста невозможно (решение, предложенное в разд. 11.7, годится лишь в определенных случаях).

Тогда как внешние графические языки не допускают столь тесной взаимосвязи, другие языки, разработанные на базе LaTeX'а, предоставляют такую важную функциональную возможность. Сам LaTeX уже снабжен простым объектно-ориентированным графическим языком достаточно низкого уровня — графическим режимом. Хотя и не всегда простой в использова-

нии, он позволил поколениям пользователей LaTeX'а создавать диаграммы удивительной сложности. На его базе было разработано несколько специализированных языков высокого уровня, предназначенных для решения задач, начиная с набора химических формул и заканчивая шахматной записью.

За прошедшее десятилетие были разработаны и нашли применение другие универсальные языки, не основанные на LaTeX'овском графическом режиме. Два очень важных из них, PSTricks и Xy-pic, подробно описаны в гл. 4 и 5. Основанные на разных парадигмах, они существенно различаются в используемых подходах, расстановке приоритетов и пользовательском интерфейсе, поэтому оба они нашли себе место в мире LaTeX'а. В последующих главах будут рассмотрены специализированные языки, предназначенные для более узких областей применения, таких, как набор физических и химических диаграмм (гл. 6), музыка (гл. 7) и игры (гл. 8).

Особенно важно использовать LaTeX для добавления текста к графическому файлу в случае, когда текст содержит формулы, что часто имеет место со следующими двумя типами графических объектов:

- Алгоритмически простроенная графика (гистограммы, графы, и т. п.) представляет собой изображения, созданные без участия человека, но часто содержащие текст, который должен соответствовать тексту документа. Масштаб и расстояние между элементами являются неотъемлемой частью изображения.

  Богатые возможности для построения графиков и диаграмм предоставляются многими LaTeX'овскими пакетами, основанными на графическом режиме, TeX'овскими пакетами общего назначения, наподобие PiCTeX (Wichura, 1988) и DraTex (Gurari, 1994), и языком PSTricks, описанным в гл. 4. Все эти решения позволяют применить всю мощь возможностей набора LaTeX'а к текстовым частям графических объектов и, таким образом, идеально интегрировать их с окружающими элементами документа.

- Наконец, существует алгоритмическая структурная графика, которая может быть получена из текстового представления. Однако, в отличие от предыдущей категории, часто важными являются лишь пространственные взаимосвязи между элементами, а не их точное расположение или размер. Примерами рисунков такого типа служат диаграммы категорий, деревья и блок-схемы алгоритмов. Такие графические объекты являются естественными кандидатами на то, чтобы быть созданными при помощи внутренних графических языков LaTeX'а, которые предоставляют интерфейс высокого уровня, позволяющий сосредоточить внимание на объектах и взаимосвязях между ними, тогда как окончательное решение о местоположении и внешнем виде принимается автоматически.

  Для графических объектов такого рода система Xy-pic, описанная в гл. 5, возможно, является наиболее гибкой среди универсальных языков, хотя PiCTeX, PSTricks и DraTex также могут оказаться удобными. Для специальных приложений, таких, как рисование деревьев, существует много

других LATEX'овских языков (см., например, Brüggemann-Klein and Wood (1989)).

Как мы видим, существует много различных типов графических объектов, каждый из которых предъявляет собственные требования. Первые три типа по отношению к LATEX'у представляют собой черные ящики и, таким образом, их использование в LATEX'овском документе требует всего лишь включения их в документ и в некоторых случаях манипуляций с ними как с единым целым. Необходимые функциональные средства подробно обсуждаются в гл. 2, но, как мы увидим далее в этой главе, TEX также предоставляет средства для включения графики, которые не используют описанного в гл. 2 интерфейса и представляют интерес в некоторых ситуациях.

Последние три типа являются гораздо более типичной разновидностью графических объектов в научных текстах. Их применение разнообразно, начиная с представления карт (Taupin, 1993b) и заканчивая химическими структурами и коммутативными диаграммами. Их объединяет объектно-ориентированный формат данных, т. е. они описываются (по крайней мере частично) абстрактным образом путем определения объектов и их взаимосвязей с помощью соответствующего языка. В большинстве случаев для таких объектов желательна тесная интеграция с окружающим текстом, которая может быть достигнута путем использования одного из графических языков, описанных в этой книге.

В некоторых случаях интерактивные программы рисования умеют представлять свои результаты на одном из графических языков, созданных на базе графического режима LATEX'а. Написанный Джорджем Хорном пакет TeXcad, распространяемый вместе с популярной реализацией emTEX для MSDOS и OS/2, возможно, наиболее известен; широко используемая программа xfig для Unix также может осуществлять преобразования во внутренний графический язык. Таким образом, рисунки подобного рода, несмотря на то, что они созданы с помощью внешних средств, тоже могут подчиняться форматирующим инструкциям, встречающимся внутри документа. Хотя такой автоматически полученный LATEX'овский код обычно не пригоден для последующего ручного редактирования и модификации и, таким образом, манипуляции с ним ограничены средствами форматирования, реализованными в графическом языке, тем не менее такой подход может в некоторых случаях обеспечить наилучший результат.

1.3 Интерфейсы, предоставляемые TEX'ом

Чтобы оценить достоинства разных подходов к графическим объектам, реализуемым разными пакетами, полезно рассмотреть еще одну тему: средства, предоставляемые TEX'ом для обеспечения взаимодействия с ними. Описание методов создания, включения и манипулирования графикой даст вам некоторое представление о таких важных вопросах, как мобильность, качество и

потребность в ресурсах каждого из решений. Предполагается, что читатель имеет некоторое представление о том, как работает TeX, т. е. о процессе превращения исходного файла в dvi-файл, используемый драйвером для печати результатов работы.

В изложении мы будем следовать принятой выше классификации: сначала мы рассмотрим способы включения графики, созданной другими приложениями, (иначе говоря, такой, которая с точки зрения TeX'а является черным ящиком), затем опишем средства, предоставляемые для манипуляций с такой графикой и, наконец, в заключение, рассмотрим средства, предоставляемые для построения графических языков внутри самого TeX'а (то, что может послужить базисом для последних трех пунктов в нашей классификации).

1.3.1 Методы интеграции

TeX предлагает два основных подхода для интеграции графики как единого целого: одно состоит в использовании команды \special, а второе — в использовании шрифтового интерфейса. Последний способ можно подразделить на два: во-первых, представление всего графического объекта в виде одного символа специального шрифта (или нескольких таких символов, если объект велик), во-вторых, создание изображения, разложив его каким-нибудь препроцессором и использовав шрифты, содержащие небольшие группы пикселов, передающих оттенки серого цвета. На практике применяются все три подхода, причем чаще всего предпочтение отдается первым двум.

Использование команд \special

Читая *The TeX book*, главный источник информации о языке TeX (Knuth, 1986a), мы не нашли способов непосредственного включения графики, созданной внешними приложениями. Единственной доступной командой оказывается команда \special, которая сама по себе ничего не делает, но позволяет воспользоваться возможностями, которые могут быть заложены в dvi-драйвер, или, цитируя Кнута:

> Команда \special помогает использовать имеющееся у вас специальное оборудование, например для печати книг знаменитым TeXниколором.

Другими словами, Кнут видел, что может возникнуть потребность в расширении языка TeX, но не захотел ввести примитивы для дальнейших графических операторов и структур данных или, в случае включения созданной внешними приложениями графики, хорошо определенный интерфейс.

Итак, команда \special позволяет использовать *специальные* свойства программы-драйвера, которая переводит результат работы TeX'а, представленный в формате dvi, на язык, понятный выходному устройству, и если такой драйвер содержит механизмы для включения графики, созданной внешними средствами, то она может быть импортирована. Расплачиваться приходится утратой мобильности, поскольку исходный файл содержит обращения

к нестандартному интерфейсу. Так и происходит на самом деле: поскольку авторы различных драйверов используют разные соглашения, то документы, содержащие прямые обращения к команде `\special`, поддаются обработке только при очень специфических настройках.

Поэтому в 1993 году Лесли Лэмпорт, Франк Миттельбах и Крис Роули разработали для LATEX'а интерфейс высокого уровня, который абстрагируется от лежащего в его основе синтаксиса низкого уровня, понятного отдельным драйверам. Этот интерфейс был реализован Дэвидом Карлайлом и Себастианом Ратцем в виде пакета graphics (Carlisle, 1995) для нового стандарта LATEX 2_ε (официально ставшего доступным в июне 1994 г.) который подробно обсуждается в гл. 2. Похожим образом применение команд `\special` для работы с цветом, поддерживаемым некоторыми драйверами, стало прозрачным с появлением интерфейса высокого уровня, предоставляемого пакетом color, который описывается в гл. 9.

Благодаря введению набора команд высокого уровня из документа эффективно устраняется зависимость от особенностей конкретно используемого драйвера. Единственным местом в документе, которое надо изменить в случае введения таких команд, оказывается одна строка в преамбуле, указывающая способ реализации этих команд (путем загрузки управляющего файла, соответствующего конкретному драйверу).

Хотя разработка подобных интерфейсов уменьшила потребность в полностью стандартизованном синтаксисе для использования команды `\special`, подобная стандартизация по-прежнему необходима, поскольку она упростила бы как реализацию существующих интерфейсов, так и дальнейшую разработку интерфейсов высокого уровня для конкретных приложений.

Шрифтовой интерфейс

Иной способ включения графики, созданной другими приложениями, вытекает из того, что в TEX'е *уже есть* встроенный стандартный механизм осуществления взаимосвязи с внешними графическими объектами — его система обращения к внешним шрифтам. Шрифт в TEX'овском представлении описывается своим внешним именем и `tfm`-файлом, содержащим информацию о знаках этого шрифта, необходимую для определения их положения. В этом и заключается суть — реальная форма глифа не имеет значения до тех пор, пока речь идет о TEX'е. Все остальное, т. е. реальное отображение этого глифа, выпадает на долю программы-драйвера. В настоящее время существует формат (`pk`- или `pxl`-файлы), который понимается всеми современными драйверами и может в некоторой степени служить форматом для обмена графическими объектами. Если изображение может быть преобразовано в такой формат, то его можно включить, загрузив этот «шрифт» с помощью стандартных LATEX'овских команд, выбрав этот шрифт и выбрав в нем символ, представляющий собой требуемое изображение. Эта техника эффективна и мобильна (подобные шрифты могут использоваться разными реализациями

TEX'а и почти всегда могут использоваться разными выходными устройствами). Главными недостатками являются то, что изменение масштаба не может быть сделано иначе, как повторным созданием шрифта с другим разрешением на основе исходного изображения, и то, что TEX может загрузить лишь ограниченное число шрифтов.

Рис. 1.5. Отсканированный мультипликационный кадр, преобразованный в шрифт с помощью формата pbm.

Существуют программы, преобразующие файлы из формата Microsoft Paint в формат pxl (программа bitpxl, созданная Густавом Нойманном) и из формата pbm[1] в формат pk (программа pbmtopk, созданная Ангусом Даггэном). Обе они работают, переводя одну или несколько имеющихся картинок в растровом формате в шрифтовый формат и называя последовательно идущие картинки буквами алфавита. Например, кадр из мультфильма Дуэйна Бибби был вставлен в эту книгу путем сканирования и преобразования полученного графического файла из формата gem в формат pbm. Потом из него с помощью pbmtopk был создан шрифт, названный lion и содержащий единственную букву «A», представляющую собой этот рисунок. После этого рисунок может быть вставлен в TEX'овский документ с помощью декларации вида \font\lion=lion, требующейся для загрузки шрифта, за которой следует команда {\lion A}, помещенная в той точке, где должен появиться рисунок.

Рис. 1.6. Цветной файл в формате gif, преобразованный в файлы TEX'овских шрифтов с помощью bm2font.

Наиболее сложной из подобных программ является bm2font, созданная Фридгельмом Совой, которая на входе получает изображения в форматах pcx или gif, а выдает один или более pk-файлов и соответствующих им tfm-файлов. Многие драйверы не умеют обращаться с символами большого размера, поэтому эта программа автоматически разбивает изображение на части и представляет их в виде отдельных символов. Для того чтобы суть происходящего была понятна пользователю, программа создает tex-файл, восстанавливающий из частей единое целое.

В рамках этого подхода можно осуществить обработку полутоновых изображений и даже

[1] Формат pbm (мобильное растровое представление (portable bitmap)) представляет интерес в качестве промежуточного графического формата при преобразовании одного формата в другой, поскольку существуют программы, умеющие получать многие графические форматы из формата pbm и преобразовывать в него. Пакет, в основном написанный Джефом Посканцером, входит в разряд общедоступных программ. Этот формат наряду со многими другими описан в книге (Murray and van Ryper, 1996).

цветоделение (Sowa, 1994). Недостатком, однако, является то, что при этом быстро создается большое число новых шрифтов, тем самым исчерпываются ресурсы TEX'а. По этой причине данный подход может применяться лишь в документах, содержащих мало графики. В приложении А.3 перечислены опции, указываемые в командной строке программы bm2font.

Обработка полутоновых изображений

Полутоновое изображение может быть вставлено в документ также путем создания шрифта, состоящего из блоков разных оттенков серого цвета и комбинирования их обычным TEX'овским способом. И Дональд Кнут (Knuth, 1987), и Адриан Кларк (Clark, 1987) продемонстрировали этот способ. Он позволяет изменять масштаб изображения, но не позволяет легко добавлять текст внутрь изображения, а само изображение зависит от исходного разрешения. Созданные Кларком шрифты и программу предобработки можно найти в TEX'овских архивах.

Разбиение на полутона можно выполнить средствами PostScript'а, что гораздо менее трудоемко, чем разбиение на полутона средствами TEX'а, но удобство достигается за счет утраты аппаратной независимости и гибкости.

1.3.2 Методы манипулирования

На первый взгляд средства TEX'а, предназначенные для манипулирования графикой (например, изменения масштаба или поворота) и применяемые в перечисленных выше методах, кажутся относительно бедными. Если графика включена в виде одного или нескольких символов шрифта, то изменение масштаба, в принципе, возможно: по крайней мере, с точки зрения TEX'а препятствий не существует, поскольку можно загрузить шрифты любого размера. Но на стадии печати программа-драйвер, наверное, пожалуется, что не может найти шрифт необходимого размера, что означает необходимость генерации шрифта с нужным кеглем, т. е. решения задачи (по меньшей мере) отнимающей время.

В качестве единственной альтернативы остается вновь прибегнуть к возможностям команд `\special` используемого драйвера. Для некоторых драйверов, особенно тех, которые преобразуют файлы из формата `dvi` в формат PostScript, этот метод предоставляет широкий выбор методов манипулирования, включая изменение масштаба, повороты и т. д. Однако до недавнего времени слабым местом этого подхода были различия в синтаксисе аргументов команды `\special`. Проблема была разрешена лишь с появлением пакета graphics, обсуждавшегося выше.

Когда выбирается подход разбиения на полутона, складывается немного другая ситуация, потому что в этом случае требуемые шрифты являются шрифтами METAFONT'а и они автоматически могут быть созданы любых размеров с помощью современного драйвера при наличии исходных файлов для их генерации.

1.3.3 Графический инструментарий TeX'а

Далеко не всегда графический материал имеет форму, готовую для включения в документ — часто сначала требуется его создать. В следующих разделах будут рассмотрены существующие в мире TeX'а средства для создания графики.

Использование встроенных команд TeX'а

Как было замечено ранее, TeX не обладает богатым набором графических примитивов. Все, чем мы располагаем, это встроенные функции, позволяющие рисовать горизонтальные или вертикальные прямые произвольной толщины, отсутствует даже возможность рисования наклонных линий. Кроме этого TeX позволяет с высокой точностью позиционировать объекты на странице. Последней разновидностью доступных функций являются примитивы для расположения объектов в матричных структурах, где точное положение определяется автоматически в соответствии с размерами объектов (например, LaTeX'овские окружения `tabular` и `array`). Но даже с таким минимальным набором оказывается возможным определить мощные графические языки, особенно если в качестве строительного кирпичика рассматривать символ «.», поскольку произвольные линии и кривые можно рисовать, помещая бок о бок сотни крошечных точек.

Использование шрифтов

Вместо использования символа «.» в качестве индивидуального графического объекта Лесли Лэмпорт снабдил LaTeX базовым набором предназначенных для рисования макро, в которых используются специальные шрифты, содержащие отрезки прямых, проведенных под разными углами, а также фрагменты окружностей и кривых разных размеров. Эти макро позволили программистам строить удивительно сложные изображения, не считая весьма достойных способов использования применения окружения `picture`, входящего в стандартный состав LaTeX'а. Были созданы и другие языки, основанные на умении TeX'а осуществлять типографский набор из фрагментов изображения (см. например, Ramek (1990) и Waldschmidt (1988)), но стандарт LaTeX'овского графического режима и его шрифты на сегодня можно считать устоявшимися. В пакете XY-pic (см. гл. 5) применен сходный подход, но используется свой набор шрифтов для изображения стрелок-указателей.

Подход, заключающийся в использовании шрифтов, не ограничивается создаваемыми METAFONT'ом шрифтами, содержащими фрагменты изображения. Существуют шрифты для нот, для шахмат и для игры го, которые позволяют набирать музыкальные партитуры, шахматные позиции и позиции в игре го, используя TeX в качестве наборного инструмента. Все они описаны в гл. 7 и 8.

Использование команд \special

Мы уже видели, что команда \special позволяет воспользоваться дополнительными возможностями драйвера (если они существуют), помогающими включать в документ графический материал. Понятно, что аналогичным образом можно получить доступ к дополнительным возможностям рисования, если драйвер обладает подобными способностями.

Чтобы понять, как работает команда \special, можно считать, что она создает невидимый пробел, который некоторым образом снабжен комментариями в виде текста ее аргумента. В зависимости от того, в каком месте исходного файла встретилась команда \special, этот «невидимый пробел» появляется в каком-то месте набранной страницы, точно так же, как любое слово из исходного файла в конце концов появляется где-то внутри абзаца. Это означает, что после получения набранного TEX'ом документа каждая команда \special сопоставляется некоторой точке на странице.

Когда драйвер встречает «невидимый пробел», созданный командой \special, он узнает положение этого пробела на странице. Затем он читает комментарии (т. е. аргумент команды \special) и (если он понимает их) выполняет нужную операцию. Следовательно, если драйвер обладает подобными возможностями, становится возможно отмечать точки и области с целью специальной обработки, начиная с проведения линии между двумя точками и вплоть до поворота некоторой области полученной после завершения набора TEX'ом. Именно с помощью этого механизма реализованы отдельные более сложные графические функции некоторых пакетов, описанных в этой книге. Еще в 1982 г. Дональд Кнут писал об использовании команды \special и соглашениях, касающихся синтаксиса ее аргумента:

> [...] автор предвидит, что после тщательных экспериментов в группах пользователей возникнут некоторые стандарты для общих графических операций TEX'а, а затем появится возможность для некоторого единообразия в использовании \special-расширений.

К сожалению, несмотря на неоднократные попытки (см. Vanderburg and Reid (1987), обсуждение данной проблемы в Rogers (1989) и Reid and Hosek (1989)), такой стандарт так и не появился. Для включения графики теперь существует интерфейс высокого уровня, абстрагирующийся от свойств конкретного драйвера. Однако ситуация с другими графическими возможностями программ-драйверов, реализуемых через команду \special, представляется менее удовлетворительной, так как не существует ни стандарта, ни интерфейса высокого уровня, скрывающего возможности разных драйверов. С другой стороны, поскольку эти возможности так сильно зависят от используемого на финальном этапе языка драйвера, такая стандартизация может так и не быть достигнута. Принимая во внимание доминирующее положение языка PostScript на рынке печати и типографского набора, возможно, наиболее полезным шагом, который следовало бы предпринять, могла бы стать стандартизация примитивов

\special среди PostScript'овских драйверов. В действительности, именно этот путь был выбран в пакете PSTricks, подробно описанном в гл. 4.

Есть только два набора соглашений, относящихся к аргументу команды \special, которые поддерживаются более чем на одной платформе. Первым является набор «tpic», изначально определенный Тимом Моргаком при реализации языка «pic» для использования его вместе с TeX'ом, который реализован в большом числе драйверов, включая xdvi, dvips, dviwin и dvipsone. Второй — это набор emTeX, который первоначально поддерживался всеми драйверами экрана и печати в реализации TeX'а под MSDOS и OS/2, выполненной Эберхардом Маттесом. Другие драйверы, включая dvips, тоже поддерживают этот набор. Хотя и не столь мощный как набор «tpic», набор emTeX сегодня можно использовать со многими типами устройств и следует считать минимально необходимым набором.

Оба набора, как «tpic», так и emTeX, использовались для реализации внутренних представлений объектно-ориентированных графических языков, построенных на основе TeX'а. Их интерфейсы описаны в приложениях A.1 и A.2.

1.4 Графические языки

Существует несколько различных способов создания графики, каждый из которых обладает своими достоинствами и недостатками в отношении простоты генерирования, гибкости, аппаратной независимости и возможности включения произвольного TeX'овского текста. Как обсуждалось в разд. 1.3.2, графическим объектом можно манипулировать как единым целым, используя различные стандартные операции, например изменение масштаба, поворот и т. д. Однако, для того чтобы манипулировать отдельными частями изображения, требуется, чтобы они были адресуемыми, т. е. исходное описание должно быть сделано на некотором абстрактном графическом языке.

В оставшейся части этой главы будет совершено «турне» по графическим языкам, описывающее, как они используют средства, предоставляемые TeX'ом, или, в случае внешних графических языков, как они могут взаимодействовать с LaTeX'овской системой для получения документов высокого качества. Некоторые из этих языков будут подробно описаны в последующих главах, описание других ограничивается лишь начальным введением.

1.4.1 Графические языки, основанные на TeX'е

За прошедшие годы было создано несколько графических языков, использующих средства, предоставляемые TeX'ом. Они различаются подходами и выбором приоритетов, поэтому степень их мобильности, потребность в ресурсах и гибкость изменяются в широких пределах.

Диаграммы и графики, набираемые символами

Изображения, построенные из шрифтов с символами одинаковой ширины, могут быть созданы на любом компьютере почти в любом редакторе и легко вставлены в LaTeX'овские файлы с помощью окружения `verbatim`. Их недостатками являются грубость, большая ограниченность, сложность создания и, в типичном случае, фиксированный размер, хотя обычные TeX'овские команды изменения размера шрифта позволяют «изменить их масштаб» (что и сделано на рис. 1.7 с помощью команды `\footnotesize`). Эта гистограмма иллюстрирует число похороненных на протестантском кладбище в Риме за десятилетия мужчин и женщин (демонстрируя постепенное увеличение количества похороненных женщин по сравнению с количеством мужчин — см. Rahtz (1987)), но добавить к диаграмме какие-нибудь существенные комментарии оказалось невозможно. Не так уж легко включить текст, набранный шрифтом переменной ширины. Это можно сделать с помощью окружения `alltt`, но за счет отказа от непосредственного использования при рисовании таких символов, как '\', '{' и '}'. Описание этого и сход-

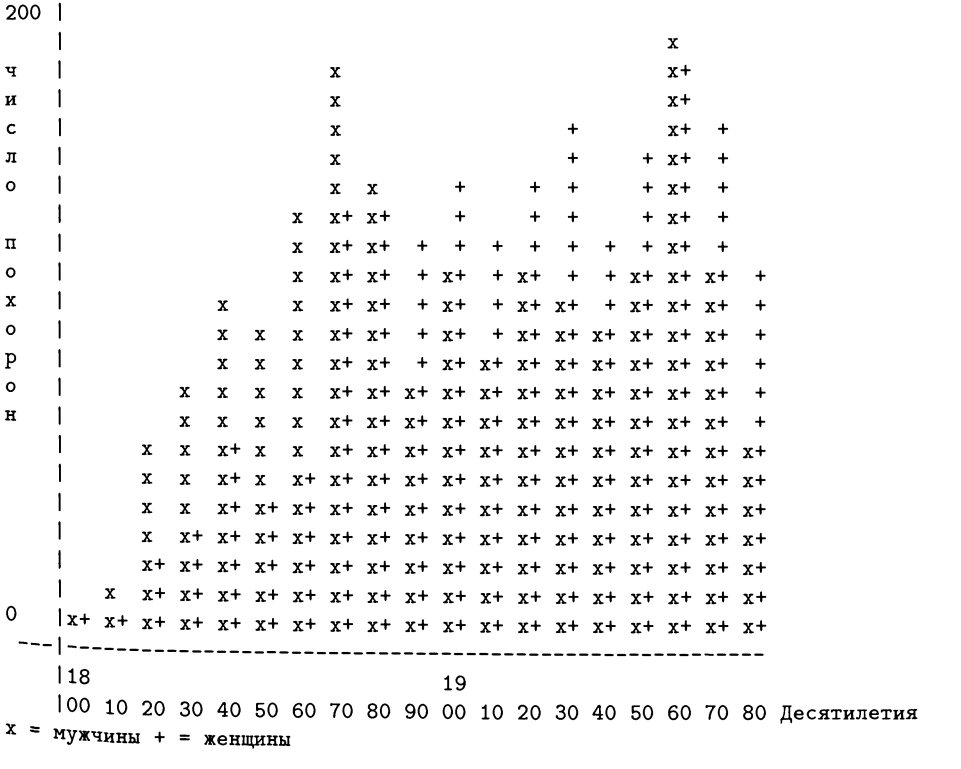


Рис. 1.7. Использование символов ASCII для построения гистограммы.

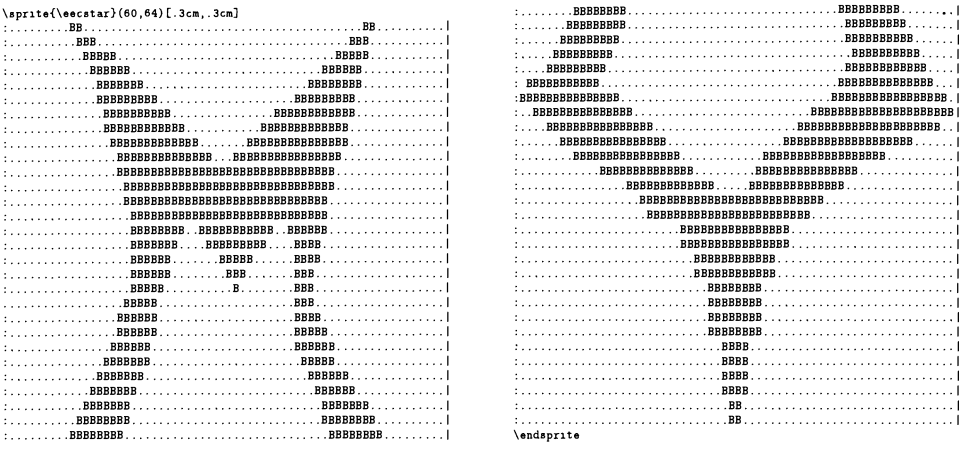


Рис. 1.8. Образец побитового кодирования, использованный для создания символа для эмблемы с рис. 1.9.

ных окружений см. в книге *The LaTeX Companion* (Goossens et al., 1994, с. 84)[1].

Похожая идея была изобретательно воплощена в пакете sprite, написанном Мартином Костаблем для создания новых TeX'овских символов путем задания побитового образа символа и изменения масштаба до требуемых размеров. Таким образом, исходный код с рис. 1.8 позволяет создать контур пятиконечной звезды в квадрате со стороной 0.3 см, который используется для создания эмблемы ЕЭС (рис. 1.9). Каждая из литер B задает крошечную линейку, размер которой определяется «разрешением» изображения, а требуемая высота и ширина символа «спрайт» задаются аргументами команды `\sprite`.

Хорошо видно, что очертания созданной звезды очень грубы. Если сторону квадрата, обрамляющего этот символ, выбрать равной 3 см, результат будет выглядеть ужасно (рис. 1.10). Применение данного метода очень ограничено, поскольку создание исходного «побитового» образа отнимает много времени, а рисование гладких кривых вызывает большие трудности. Этот метод требует также большого объема вычислений и памяти на диске.

Внешним интерфейсом пакета sprite служит программа bit2spr, написанная Марком Ровнером, которая преобразует уже существующие в X11 битовые образы в формат этого пакета[2]. Так как LaTeX'у надо знать размеры спрайта, программа bit2spr, взяв высоту и ширину образа, с помощью отношений вычисляет высоту и ширину спрайта в дюймах. По умолчанию отношения

[1] Здесь и далее страницы указаны по русскому переводу. — *Прим. ред.*

[2] Битовые образы X11 (X11 bitmaps) — растровый формат, используемый в графическом интерфейсе X Windows в компьютерах, работающих под управлением операционной системы Unix. — *Прим. ред.*

1-4-1

Рис. 1.9. Эмблема, созданная с помощью пакета «sprite».

1-4-2

Рис. 1.10. Увеличенный спрайт.

1-4-3

Рис. 1.11. Результат работы пакета bit2spr.

считаются равными 1.5, но они могут быть изменены при помощи необязательных параметров командной строки `-width` и `-height`. Отношения могут быть равны любому рациональному числу, но экстремальные значения с высокой вероятностью приведут к тому, что TEX'у не хватит памяти. На рис. 1.11 показан простой пример: созданная Дональдом Кнутом (Knuth, 1993) иконка в формате `dvi`.

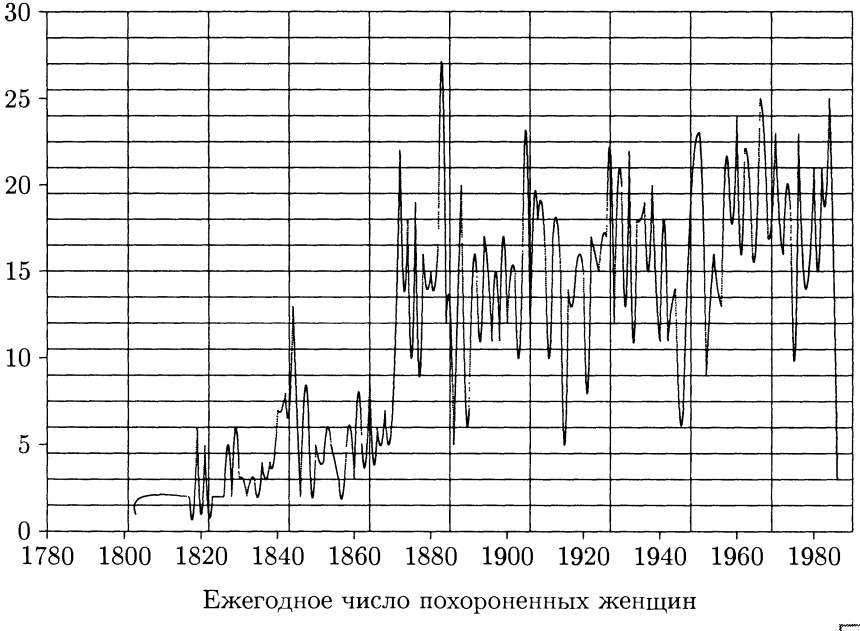
PICTEX

Комбинированный подход к рисованию без использования новых шрифтов был применен Майклом Вишурой в своем замечательном PICTEX'е (Wichura, 1987, 1988), в котором полный графический язык реализован, в основном, путем размещения мириадов точек, взятых из стандартного TEX'овского шрифта. Главный его недостаток состоит в чудовищном объеме вычислений и колоссальной потребности в памяти, что делает обязательным использование больших версий TEX'а (на небольших машинах расчет изображений может занять несколько минут и использование PICTEX'а совместно с другими пакетами без исчерпания TEX'овских переменных «размера») оказывается совсем непростой задачей![1] К его достоинствам относятся возможность изменения масштаба и легкость, с которой вставляется обычный TEX'овский текст, а сложность графического языка является скорее конструктивной особенностью, чем неизбежной особенностью данного подхода. При создании рис. 1.12 и 1.13 использовались данные, на основе которых был построен рис. 1.7, а над рисунками помещены авторские тексты программ для их создания![2]

[1] С последней проблемой можно справиться путем переопределения некоторых макро низкого уровня, но большинству пользователей такой подход вряд ли понравится.

[2] Чтобы продемонстрировать, что имеется возможность работать с русским языком, в эти тексты были добавлены 2 команды (строки 2 и 3), посредством которых вводятся русификация и кириллические шрифты. Такие же изменения внесены и в других аналогичных случаях. — *Прим. ред.*

```
\usepackage {graphics,rawfonts,pictex}
\usepackage [cp866]{inputenc}
\usepackage [LCY]{fontenc}
\beginpicture
\normalgraphs
\longticklength=3pt
\setcoordinatesystem units <.02in,.09in>
\setplotarea x from 1780 to 1990, y from 0 to 30
\axis bottom label {Ежегодное число похороненных женщин} ticks
      numbered from 1780 to 1980 by 20 /
\axis left ticks numbered from 0 to 30 by 5 /
\grid 10 20  \setquadratic  \plot "year.wom"
\endpicture
```



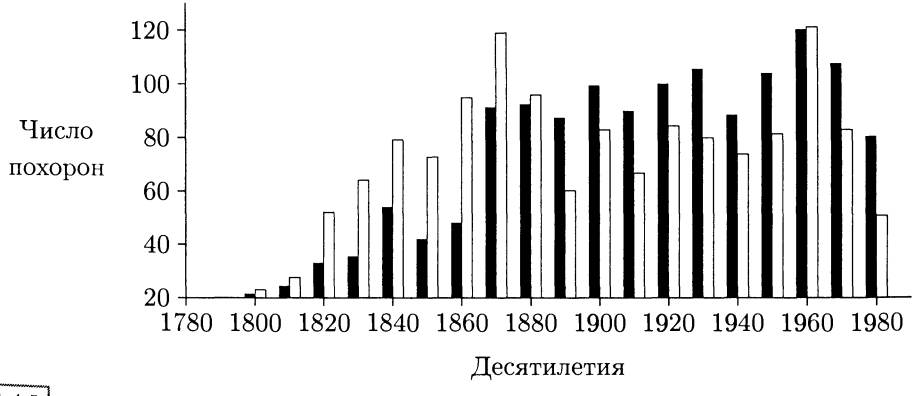
1-4-4

Рис. 1.12. Исходный код PiCTeX'а и полученный график.

Достоинства системы, подобной PiCTeX'у, иллюстрирует рис. 1.14, на котором приведена гистограмма. Заголовки обрабатываются обычным TeX'ом, поэтому они могут содержать математические формулы, и поворачиваются с помощью стандартного LaTeX'овского графического пакета, что значительно расширяет возможности пользователя по улучшению внешнего вида рисунков, сохраняя при этом совместимость с TeX'овскими шрифтами.

Изначально пакет PiCTeX был разработан для совместной работы с форматом plain TeX и поэтому в нем используется команда (`\fiverm`), которая не

```
\usepackage {graphics,rawfonts,pictex}
\usepackage [cp866]{inputenc}
\usepackage [LCY]{fontenc}
\beginpicture
\normalgraphs
\setcoordinatesystem units <.018in,.007in>
\setplotarea x from 1780 to 1990, y from 0 to 220
\longticklength=3pt
\axis bottom label {Десятилетия} ticks
numbered from 1780 to 1980 by 20 /
\axis left
label {\stack {Число\strut,\strut похорон}}
ticks withvalues 20 40 60 80 100 120 140 160 180 200 /
from 0 to 200 by 40 /
\setbars <-2pt,0pt> breadth <0pt> baseline at y = 0
\linethickness=4pt
\plot "decade.wom"
\setbars < 2pt, 0pt> breadth <4pt> baseline at y = 0
\linethickness=.25pt
\plot "decade.men"
\endpicture
```



1-4-5

Рис. 1.13. Исходный код PiCTeX'а и полученная гистограмма.

определена в LaTeX'е. В большинство версий TeX'а входит файл `pictex.sty`, позволяющий использовать PiCTeX как LaTeX'овский пакет. Если он отсутствует, проблему решат следующие строки:

```
\DeclareFixedFont{\fiverm}{OT1}{cmr}{m}{n}{5pt}
\input{prepictex}
\input{pictex}          % основная программа и немного документации
\input{postpictex}
\endinput
```

```
\usepackage {graphics,rawfonts,pictex}
\beginpicture
\setcoordinatesystem units <10pt,.03pt>
\setplotarea x from -1 to 24, y from 0 to 4500
\axis left label
\rotatebox{90}{Number of Coins ($\surd$)} ticks
numbered from 0 to 4500 by 500 /
\shaderectangleson
\setbars breadth <15pt> baseline at y = 0
baselabels ([tl] <2.5pt,-5pt>)
\plot "coins.dat"
\endpicture
```

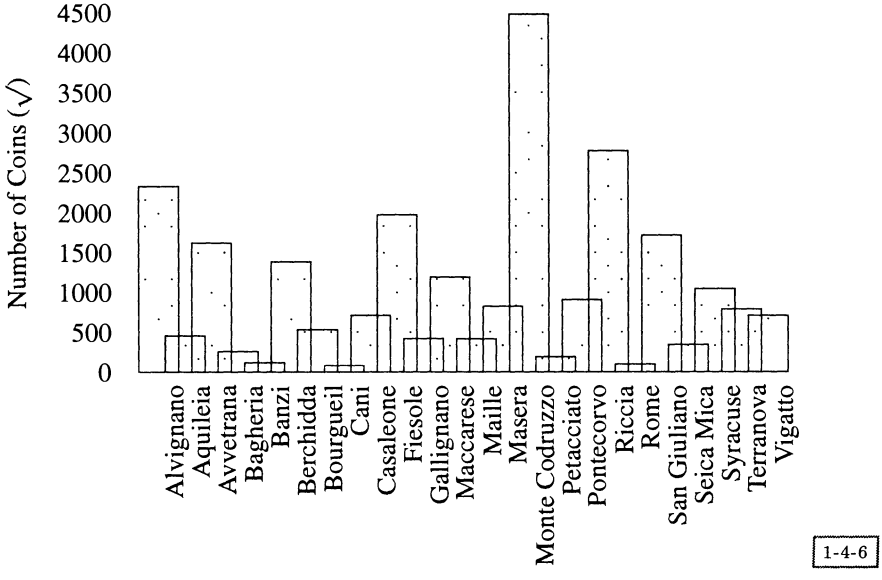


Рис. 1.14. Исходный код и гистограмма с повернутым текстом, нарисованная с помощью PICTEX'а (диаграмма предоставлена Крисом Локйером).

Пакеты DraTex и AlDraTex

Семейство основанных на TEX'е графических языков недавно пополнилось пакетом, разработанным Эйтаном М. Гурари (1994), при помощи которого удобно рисовать наиболее часто используемые диаграммы. В нем два уровня: один предназначен для базовых команд рисования (пакет DraTex), а другой — для конструкций высокого уровня (пакет AlDraTex). Они реализованы, в основном, с использованием примитивов TEX'а, но при необходимости обращаются к LATEX'овским шрифтам, содержащим части окружностей. Оба пакета подробно описаны в Gurari (1994). В качестве простого примера на рис. 1.15 показана круговая диаграмма.

Подходы с использованием окружения picture

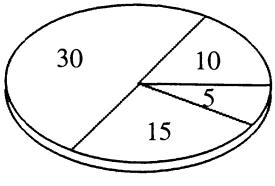
Находясь в режиме `picture`, LaTeX предоставляет базовый графический язык, полностью переносимый между различными реализациями TeX'а. Хотя и не всегда простой в использовании, он позволил поколениям пользователей LaTeX'а создавать диаграммы удивительной сложности.

Существуют различные расширения графических макро LaTeX'а. Наиболее широко применяется созданный Сьюнилом Подаром пакет epic (1986), команды которого расширяют графические возможности LaTeX'а и обеспечивают более дружелюбный и мощный пользовательский интерфейс путем сокращения вычислений, требуемых для задания внешнего вида объектов. Пакет epic, а также принадлежащий Йоахиму Блезеру макро bar, предназначенный для рисования гистограмм, и расширение графического режима LaTeX'а под названием curves описаны в Goossens et al. (1994).

Недостатки ограниченных возможностей LaTeX'овских шрифтов для рисования очевидны, даже когда используются улучшения, сделанные в epic. Размеры окружностей и углы наклона прямых могут принимать фиксированные значения, контуры имеют зигзагообразный вид (поскольку строятся из маленьких сегментов), отсутствуют средства для закрашивания или затемнения областей. К достоинствам этого метода можно отнести высокую мобильность и хорошую интеграцию с остальным текстом.

Одно важное дополнение к LaTeX'овскому графическому режиму состояло в переопределении графических макро так, чтобы сделать их аппаратно-зависимыми, преобразовывая их непосредственно в команды `\special` языка PostScript. Это освобождает пользователя от ограничений в виде фиксированного диапазона размеров окружностей и углов наклона прямых. Ранние версии описаны в Clark (1989) и Arbortext Inc. (1986), но в будущем функционирование будет обеспечиваться пакетом pict2e, как частью, входящей в стандартный дистрибутив LaTeX'а. В настоящее время пакет pspicture, созданный Дэвидом Карлайлом, уже выполняет некоторые из этих функций.

Существует повторная реализация пакета epic, называющаяся eepic и выполненная Конрадом Куоком с помощью команд `\special` программы tpic, ко-



1-4-7

```
\usepackage {DraTex,AlDraTex}
\Draw
   \Scale(1,0.6)
   \PieChartSpec(1,50,25)()
   \PieChart(10 & 30 & 15 & 5)
   \Move(0,-6)
   \DrawOvalArc(50,50)(180,360)
\EndDraw
```

Рис. 1.15. Круговая диаграмма, созданная при помощи пакета AlDraTex.

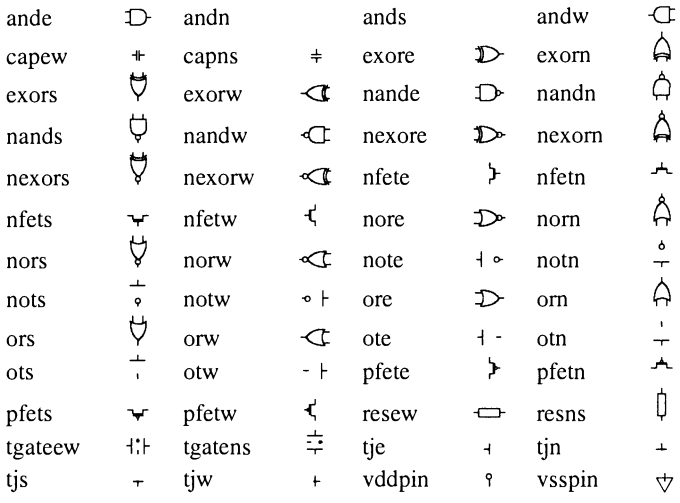


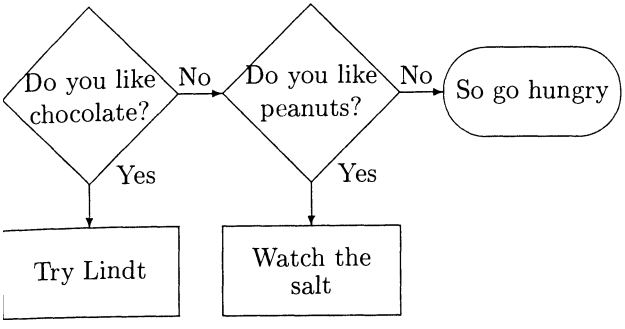
Рис. 1.16. Библиотека элементов lcircuit.

торые распознаются несколькими драйверами (см. разд. 1.2 и приложения А.1 и А.2.)

Было предложено несколько подходов к созданию интерфейсов высокого уровня для графического режима. Типичным представителем первого подхода является разработанный Адрианом Джонстоном пакет lcircuit, представляющий собой библиотеку элементов электрических схем для графического режима LaTeX'a. В этот набор входят все основные логические элементы в четырех ориентациях: полевые транзисторы, выводы для подключения источников питания, пропускающие вентили, конденсаторы, резисторы и Т-образные соединения проводников (см. рис. 1.16). Каждый символ определен при помощи команды `\savebox` и хранится во внешнем файле, позволяя таким образом без проблем использовать символы многократно. Однако не предпринималось никаких попыток для повышения функциональности, т. е. построение большей схемы из этих символов должно выполняться вручную или с применением такой программы, как TeXcad, распространяемой вместе с системой emTeX.

Второй и на сегодняшний день наиболее широко распространенный подход заключается в определении небольшого языка целиком в рамках LaTeX'a. Примером такого подхода может служить пакет nassflow Йоханнеса Брамса, предназначенный для рисования блок-схем или диаграмм Насси-Шнайдермана с помощью окружения `picture`. В последующих главах приведено несколько примеров такого подхода.

Третий подход состоит в том, чтобы наряду со специализированным внешним языком определить и программу предварительной обработки, преобразующую описанные на этом языке входные данные в примитивы TeX'a. Наи-




```
Right
Choice . . No Yes
  Do you like
  chocolate?
Tag
Choice . . No Yes
  Do you like
  peanuts?
Tag
Right
Oval
  So go hungry
ToTag
Down
Box
  Try Lindt
ToTag
Down
Box
  Watch the
  salt
```

Рис. 1.17. Пример на языке flow.

более сложной из программ подобного рода, пожалуй, является описанная ниже программа gpic, но существует много менее сложных примеров, например, программа flow Терри Брауна, преобразующая простые файлы описаний блок-схем в команды LaTeX'овского окружения `picture`. Этот язык позволяет соединять стрелками выбранного направления боксы, заключенные в рамки разной формы. Задание нескольких направлений осуществляется путем определения «меток» (tags), помещения их в стек и последующего возвращения к ним. На рис. 1.17 показан пример диаграммы, построенной с помощью этого языка.

XY-pic

XY-pic — это язык для построения графов и диаграмм с помощью TeX'а, первоначально разработанный Кристофером Х. Роузом с помощью Росса Мура. При обычных настройках эта система пользуется стандартным TeX'ом и METAFONT'ом, т. е. она следует подходу на основе шрифтов, используя свои собственные шрифты для изображения стрелок, и т. д., но результат может быть создан для конкретного драйвера (наподобие dvips). С помощью этой программы можно набирать сложные диаграммы теории категорий, теории автоматов, алгебры, нейронных сетей и теории баз данных. Система построена вокруг объектно-ориентированного *графического языка ядра*. Каждому «объекту» на

рисунке сопоставлен «Метод», описывающий, как этот объект надо набирать, растягивать и т. д. В работе ядра было сделано несколько улучшений, которые получили название *расширения*. Язык Xy-pic будет подробно рассмотрен в гл. 5.

PSTricks

Пакет PSTricks позволяет реализовать возможности чрезвычайно мощного языка описания страниц PostScript, который используется во многих принтерах и наборных устройствах. С помощью команд `\special` в исходный TeX'овский документ вставляются фрагменты PostScript'овского кода, которые передаются далее соответствующими драйверами (включая dvips, dvipsone и Textures). Это позволяет реализовать различные эффекты, начиная с рисования линий, использования цвета, теней, преобразования символов (такого, как размещение текста вдоль кривой) и заканчивая трехмерной псевдографикой. Пакет PSTricks позволяет получить доступ к большинству возможностей языка PostScript, используя знакомый синтаксис TeX'а, а функциональность низкого уровня расширяется пакетами высокого уровня для таких случаев, как, например, рисования деревьев или графов. Пакет PSTricks будет подробно рассмотрен в гл. 4.

1.4.2 Внешние графические языки и программы для рисования

Существует много широко распространенных графических языков и интерактивных пакетов, начиная с программ, функционирующих на очень низком уровне, и заканчивая узко специализированными программами, а созданные трансляторы представляют их результаты в таком виде, в котором их может обрабатывать TeX, будь то LaTeX'овская графика, набор команд `\special` или даже код METAFONT'а. В этом разделе описаны некоторые из них, но эта выборка ни в коей мере не является исчерпывающей. Многие из широко известных пакетов для рисования и черчения располагают широким выбором форматов вывода; некоторые из них могут быть преобразованы к виду, совместимому с TeX'ом, поэтому количество вариантов огромно. Составляющие отдельный класс программы преобразования формата графики (такие как Hijaak, Image Alchemy или Adobe File Utilities) имеют дело с преобразованиями другого рода (по большей части совершаемыми над растровыми изображениями).

HPGL

Очень многие из универсальных пакетов для рисования и построения графиков могут выводить данные в формате HPGL (Hewlett Package Graphics Language), который может быть переведен на язык METAFONT с помощью пакета hptomf, написанного Дирком Донатом, или преобразован во множество совместимых с

TEX'ом форматов с помощью созданного Хайнцем Вернтгесом пакета hp2xx. Он преобразует в форматы METAFONT, TEXcad, epic, EPS, «em» (команды \special emTEX'a), в векторные форматы HPGL и растровые форматы PCX, PIC, IMG и PBM, обеспечивая контроль таких характеристик как масштаб, размер пера и цветовая гамма. Во многих случаях преобразования, осуществляемые над такими TEX-ориентированными форматами, как «epic», «em» и TEXcad, не могут быть выполнены из-за ограниченности объема доступной TEX'у памяти или особенностей пакета. Опция METAFONT'а также может приводить к результатам, переполняющим память METAFONT'а или программы gftopk (для создания примера, показанного на рис. 1.18, потребовалось построить расширенную версию), но позволяет достигнуть хорошего качества. Рисунок 1.18 был получен путем преобразования вывода графопостроителя из AutoCAD'а в METAFONT. Детальные контурные линии на карте острова Коморан состоят примерно из 10000 сегментов.

Однако HPGL является языком низкого уровня и после преобразования в другой формат результат вряд ли можно будет легко редактировать или обрабатывать. Кроме того, это уже немолодой язык и его возможности, можно сказать, ограничиваются требованиями совместимости с графопостроителями и печатающими устройствами, созданными в более ранние времена.

Пакет для построения графиков gnuplot

Пакет gnuplot (созданный Томасом Уильямсом, Колином Келли и др.) является универсальной программой для рисования двумерных (и трехмерных) графиков, обладающей собственным языком описания графиков и диаграмм. К ней прилагаются драйверы для многих выводных «устройств», включая pic, LATEX (на основе emTEX'овских команд \special), METAFONT, PostScript, PSTricks, TEXdraw и eepic. Типичный график на основе информации, уже встречавшейся на рис. 1.7, представлен на рис. 1.19, на котором также указаны команды языка gnuplot, использованные для его создания.

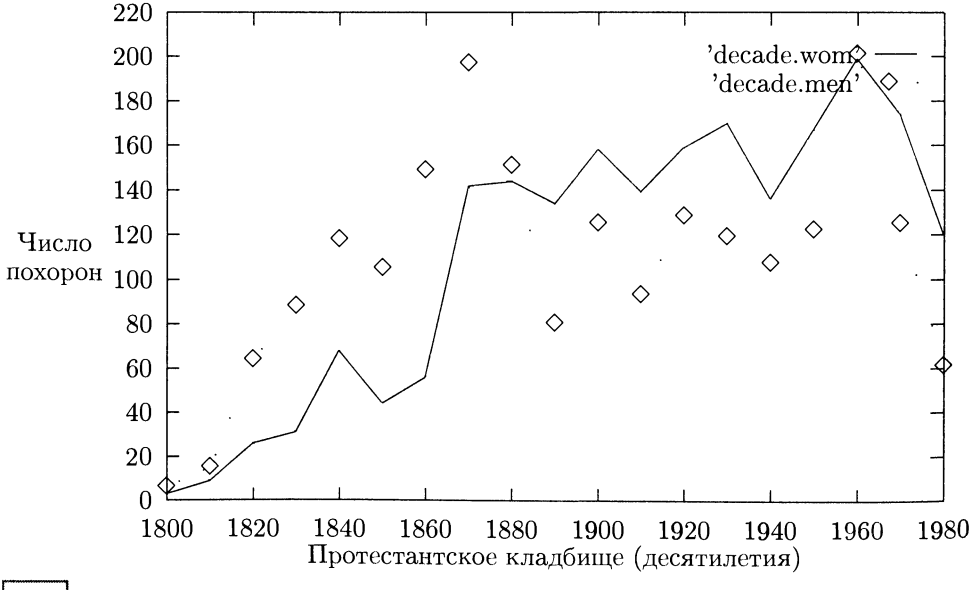
Язык pic

Те, кто привык заниматься типографским набором в операционной системе Unix, знакомы с программой troff. Для нее Брайан Керниган разработал графический язык низкого уровня и создал программу предварительной обработки (pic) для преобразования файлов, написанных на этом языке, к виду, понятному для программы troff. Еще одна версия препроцессора pic — tpic — выполняет аналогичные операции над командами языка pic, включенными в TEX'овский документ, но заменяет их на команды \special, которые вызывают графические примитивы, реализованные в dvi-драйвере. Синтаксис команд \special для tpic описан в приложении А.2 на с. 538.

Программа troff и некоторые из ее препроцессоров были заново переписаны Джеймсом Кларком в ходе его работы на Free Software Foundation. Входящая в этот набор программа gpic обладает всеми возможностями исходного про-

Рис. 1.18. Вывод графопостроителя, преобразованный из AutoCAD'а в METAFONT.

```
set terminal latex
set xlabel "Протестантское кладбище (десятилетия)"
set ylabel "Число\strut \\похорон"
plot 'decade.wom' with lines, 'decade.men' with linespoints
```



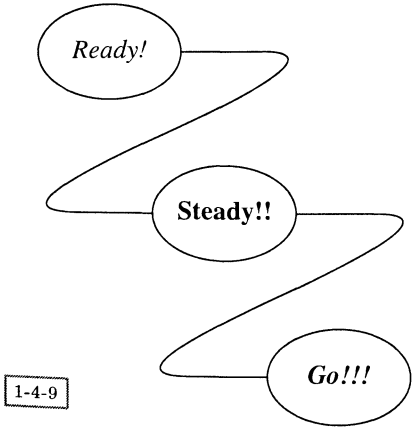
1-4-8

Рис. 1.19. График, построенный с помощью пакета gnuplot, и команды LaTeX'овского графического режима, использованные для его построения.

цессора pic, но в отличие от него, gpic также может выводить TEX'овский код, содержащий команды `\special` в соответствии с синтаксисом программы tpic, при вызове ее с опциями `-t` или `-c`.

Программа gpic читает TEX'овский файл и оставляет все строки неизменными до тех пор, пока не встретит строку, содержащую единственную команду `.PS`. С этого момента она считает, что далее следуют команды на языке pic, и преобразовывает их до тех пор, пока не встретит строку, содержащую единственную команду `.PE`.

В языке pic определены основные графические объекты, такие как `line`, `box`, `arc`, `arrow`, `ellipse`, `circle`, `spline` и текстовые строки. Для конструирования более сложных объектов могут создаваться именованные блоки. В языке есть циклы, условные переходы, переменные величины и элементарные функции (такие, как `sin`, `cos`, `atan`, `sqrt` и `rand`). В каждый момент времени текущее местоположение и направление рисования являются заданными. Положение объекта может быть задано в абсолютных величинах или относительно ранее определенных объектов или фиксированных точек. Для демонстрации рассмотрим следующие команды:

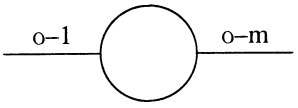


1-4-9

```
.PS
ellipse "\textit{Ready!}"
spline right .85i then down 0.85i \
       left 1.85i then right .85i
ellipse "\textbf{Steady!!}"
spline right .85i then down 0.85i \
       left 1.85i then right .85i
ellipse "\textbf{\itshape Go!!!}"
.PE
```

Эллипсы рисуются в текущих позициях, выбранных по умолчанию, а направление и размеры соединяющих их линий задаются явным образом. Текст может содержать любые команды LATEX'а. Другие примеры программирования на языке pic приведены на рис. 1.20 и 1.21.

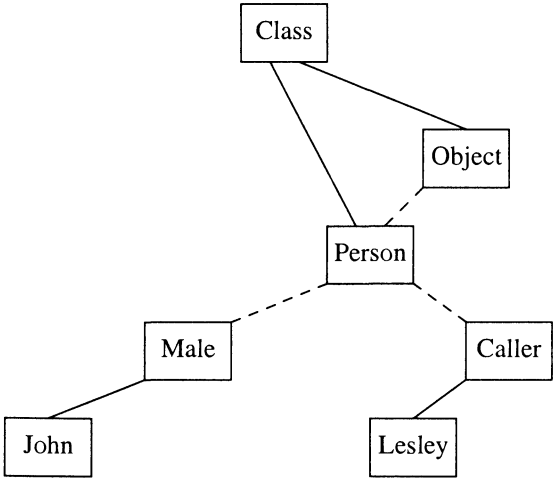
В языке pic простые графические отношения выражаются интуитивно понятным образом, в связи с чем его применение кажется более предпочтительным, чем, например, окружения `picture`. Однако связанные с подходом на основе пакета gpic проблемы носят тройственный характер. Во-первых, требуется стадия предварительной обработки, т. е. отдельная программа, откомпилированная для той компьютерной платформы, на которой она используется, во-вторых, в ней используются команды `\special` программы tpic, которые поддерживаются не всеми драйверами и, в-третьих, препроцессор ничего не




```
.PS
line "o--1" above ; circle
line "o--m" above ; move
box invis "relationships"
.PE
```

1-4-10

Рис. 1.20. Простой пример программы на языке pic.




```
.PS
define rect X
   box width 0.48i height 0.3i X
A: rect "Class" ;
   move down right from A.se
B: rect "Object"; move down left
C: rect "Person"; move down left
D: rect "Male"  ;
   move down right from C.c
E: rect "Caller"; move down left
G: rect "Lesley";
   move down left from D.c
F: rect "John"
line from 2/3 <A.sw,A.se> to B.n
line from 1/3 <A.sw,A.se>    \
       to 1/3 <C.nw,C.ne>
line dashed from B.sw        \
            to 2/3 <C.nw,C.ne>
line dashed from C.sw to D.ne
line dashed from C.se to E.nw
line from D.sw to F.n
line from E.sw to G.n
.PE
```

1-4-11

Рис. 1.21. Пример на языке pic, демонстрирующий применение сохраненных имен.

знает о том, как набирается текст, и по этой причине не может, например, точно расположить рамку вокруг формулы. Используя набор команд `\special` с существенно более широкими возможностями, можно было бы реализовать в рамках самого TeX'а язык, похожий на pic, так чтобы различные устройства смогли бы интерпретировать эти команды наилучшим возможным способом. Попыткой такого подхода, осуществленной Рольфом Олейничак-Буркертом, является пакет texpic (Olejniczak-Burkert, 1989), реализующий большую часть возможностей пакета pic. Эта система обладает значительными преимуществами, но ее язык не столь полон, как язык pic, и синтаксис вводимых команд (похожий на TeX'овские макро) не выглядит столь «элегантным», как синтаксис языка pic. Но более тесная интеграция с TeX'ом означает, что некоторые

вещи делаются лучше. Например, размер рамки, содержащей текст, может быть сделан в точности соответствующим ширине ее содержимого. Добиться этого средствами языка pic невозможно, поскольку он не имеет никакого представления о метриках шрифта.

Специализированные внешние графические языки

В некоторых случаях можно рассмотреть возможность добавления в графическую систему еще одного уровня: нового языка, разработанного специально для данного приложения, и транслятора на один из существующих аппаратно-независимых языков, с которого уже осуществляется перевод в формат TEX'а. Подобный подход описания небольших языков, предназначенных для специальных целей, обсуждается в (Bentley, 1986).

В исходный пакет pic (Kernighan, 1984), работающий в операционной системе Unix, входит сопутствующий набор языков высокого уровня, вывод которых производится в формате pic. Есть языки, предназначенные для построения графиков общего вида (программа grap, описанная в (Bentley and Kernighan, 1984)), химических формул (программа chem) и музыкальных партитур (программа music). В качестве примера были взяты данные о кладбище в Риме, на основе которых с помощью языка grap очень быстро была построена содержательная гистограмма (похожая на рис. 1.7, суммирующая количество похорон в каждом десятилетии). Инструкции для программы grap и полученный результат показаны на рис. 1.22.

Не предполагая давать здесь описание языка grap, мы лишь познакомили читателя с некоторыми главными чертами этого языка. Обратите внимание, что в тексте меток диаграмм может использоваться TEX'овский набор математических формул и что масштаб графика может быть изменен (выражение `ht 3 wid 4` в начальной команде `frame`; в качестве единицы измерения выбран дюйм).

METAFONT и METAPOST

У TEX'а есть сопровождающая его программа для создания шрифтов — это METAFONT. Хотя METAFONT был разработан для производства красивых изображений глифов, он может также очень успешно использоваться при создании рисунков для TEX'а. Он располагает мощными приемами и структурами данных, хорошо приспособленными для многих видов графики, и обладает теми преимуществами, что доступен везде, где есть TEX, и что его выходные данные (превращающиеся после преобразований в pk-шрифты) понимаются всеми dvi-драйверами. Он обладает двумя недостатками, каждый из которых может быть устранен:

1. Синтаксис языка METAFONT абсолютно не похож на синтаксис TEX'а, и некоторые пользователи находят, что его трудно освоить. Существует оригинальная «программа-упаковщик», доступная в виде LATEX'овского пакета mfpic, которая позволяет описать изображение в привычном

```
frame invis ht 3 wid 4 left solid  bot solid
label bot "Число мужчин и женщин, похороненных за каждые десять лет ($n \approx 4300$
ticks left out from 50 to 400 by 50
ticks bot out from 1800 to 1986 by 20
copy "decade.dat" thru X
tot=$2+$3
line from $1,0 to $1,tot
line from $1,tot to $1+10,tot
line from $1+10,tot to $1+10,0
X
```

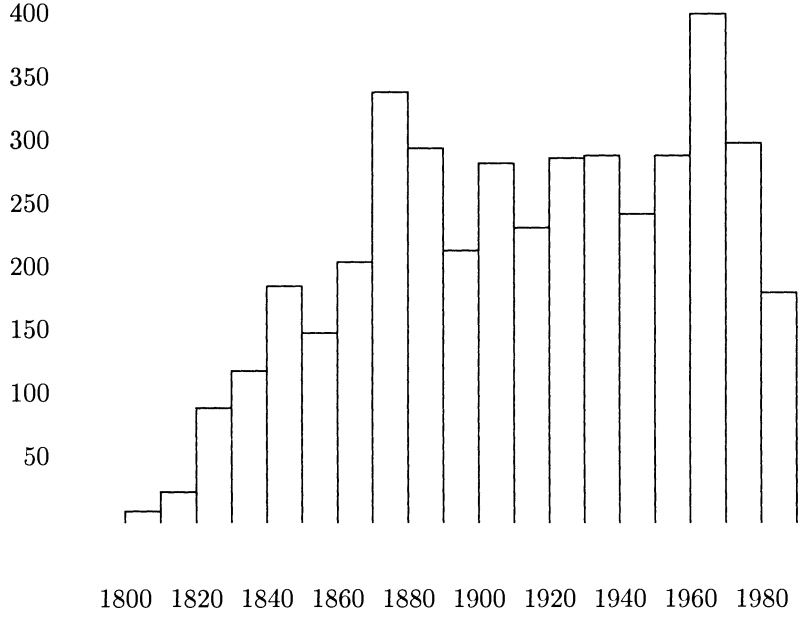


Рис. 1.22. Пример вывода программы grap.

LaTeX'овском синтаксисе и преобразовать ее в формат METAFONT'а для последующей генерации рисунка.

2. Вывод METAFONT'а в растровом формате с большим трудом поддается обработке некоторыми драйверами и делает необходимой повторную генерацию для любого нового выводящего устройства. Он очень плохо выглядит при просмотре его популярной программой Adobe Acrobat. Помимо этого в METAFONT'е отсутствует поддержка некоторых важных свойств, например цвета. Замечательным решением для многих пользователей служит METAPOST, созданная Джоном Хобби другая версия METAFONT'а, производящая аппаратно-независимый код на языке PostScript. Кроме этого добавлены поддержка цветов и некоторые средства высокого уровня для рисования графиков.

Языки METAFONT, METAPOST и пакет mfpic будут рассмотрены в гл. 3.

1.5 Выбор пакета

Большинство людей выбирают графическую программу или макропакет не на основе метода, используемого TEX'ом для манипуляций с полученным результатом. Главным критерием обычно служит *тип* изображения, которое требуется создать, а не *способ*, которым это делается. Выбирать можно из множества коммерческих интерактивных пакетов, разработанных с учетом вычислительных возможностей пользователей для соответствующих предметных областей (таких, как AutoCAD для разработчиков, ChemDraw для химиков, Adobe Illustrator для художников, SPSS для статистиков, Mathematica для математиков, ArcInfo для географов, xfig для специалистов в области информатики, Excel для бизнесменов и т. д.). Однако для других более естественным решением будет описание требуемых изображений специальными средствами.

Если вы *действительно* абсолютно свободны в выборе графического пакета, то можете принять во внимание следующие соображения:

- Решение, которое вы принимаете, зависит от взаимосвязи между графикой и текстом. Если изображение представляет собой «черный ящик», то создавайте его отдельно, а потом включите в документ на стадии печати. Если его содержимое должно быть в том же стиле, что и текст, тогда изучите графические пакеты, интегрированные с TEX'ом.
- Насколько важна для вас *полная* мобильность в мире TEX'а? Если она очень важна, то воспользуйтесь пакетами, которые используют для рисования крохотные точки или TEX'овские шрифты. Вы также можете остановиться на степени мобильности, обеспечиваемой широко распространенными (но не стандартными) примитивами `\special`, мобильности, обеспечиваемой языком PostScript или основанной на использовании растровой графики. Здесь на ваш выбор влияет предполагаемое применение результатов в будущем.
- Если вы хотите создавать требующую интерактивного рисования «художественную» графику, то остановитесь на пакете, который соответствует вашей предметной области и способен взаимодействовать с выбранным вами принтером. Если вы располагаете TEX'овским драйвером принтера, то, наверное, он позволит вам автоматически включать файлы, предназначенные для такого принтера. На практике наибольшую гибкость предоставляет огромный массив программного обеспечения, написанный для операционных систем Macintosh и Microsoft Windows, но разумно выбрать такой пакет, который позволяет получать результаты в формате инкапсулированного PostScript'а, поскольку он является наиболее мобильным форматом в издательском мире.
- Если вы хотите включать PostScript'овские картинки, используйте стандартный графический пакет, обсуждаемый в гл. 2, чтобы ваши документы не зависели от капризов и синтаксиса конкретного драйвера.

- Если вам требуется воспроизводить полутоновые изображения, их легко можно отсканировать и включить в документ с помощью TeX'а. Сканируйте с максимальным разрешением, какое вы только можете себе позволить (в смысле возможностей оборудования и занимаемого дискового пространства). Но не забудьте удостовериться, что разбираетесь в вопросах, связанных с изменением масштаба растровых изображений: сканирование с разрешением 1200 dpi и последующая печать с разрешением 1270 dpi могут привести к неприглядным результатам из-за незначительного изменения масштаба, которое необходимо выполнить.
- Если вы хотите представлять данные в относительно простой форме, выбор программного обеспечения будет зависеть от вашего обычного рабочего окружения. Если ваши данные содержатся в электронной таблице, то, вероятно, там же есть и соответствующие инструменты. Если вы пользуетесь базой данных, то может оказаться проще написать обрабатывающие программы, которые будут кодировать изображения на графическом языке наподобие LaTeX'овского окружения `picture`, pic или PiCTeX. Эти программные продукты вряд ли потребуется переносить куда-нибудь за пределы вашего рабочего окружения.
- Если вы строите алгоритмические изображения, в которых положение объектов определяется вашими данными, а не содержанием, и если вывод содержит много текстового материала (в частности, математику или непростые шрифты), то вам надо рассмотреть макропакеты, реализующие рисование непосредственно в TeX'е, чтобы ваш текст обрабатывался самим TeX'ом.

Слабое место при включении материала наподобие диаграмм в виде изображений, созданных внешними средствами, состоит в том, что рисунок никак не взаимодействует с TeX'ом. Хотелось бы иметь возможность добавлять произвольный TeX'овский текст (такой, как математическая метка на диаграмме), набранный тем же шрифтом, что и остальной текст. Какое-то время эта проблема обсуждалась, пока К. Карни (Karney, 1988) не предложил схему, в соответствии с которой автору следовало помечать подписи под рисунками и другие элементы маркированными точками на рисунке, результат передавать на обработку TeX'у в виде команд `\special` и поручать `dvi`-драйверу вставку этих подписей на этапе печати. Обсуждаемый в разд. 11.7 на с. 534 пакет PSfrag предлагает интересное альтернативное решение.

В этой книге мы пытаемся показать сильные стороны многих программ и макропакетов LaTeX'а, но не можем сказать, что лучше всего подходит именно для вашего случая. Мы лишь выражаем надежду, что разнообразие и многочисленность примеров в последующих главах продемонстрируют, что средства для работы с графикой в LaTeX'е существуют, хорошо развиты и могут удовлетворить почти любые потребности.

Стандартные графические средства LaTeX'а

LaTeX 2ε предоставляет универсальный аппаратно-независимый интерфейс для включения созданного внешними средствами графического материала, а также изменения масштаба и вращения LaTeX'овских боксов. Помимо этого существует универсальный пакет для управления цветом, который подробно обсуждается в гл. 9.

Сначала мы рассмотрим процесс включения графических файлов. LaTeX предлагает как простой интерфейс (graphics), который может быть скомбинирован с самостоятельными командами вращения и изменения масштаба, так и более сложный интерфейс (graphicx), обладающий собственным арсеналом средств манипулирования. Затем мы изучим, как работают отдельно выполняемые изменения масштаба и вращения, и завершим главу кратким описанием некоторых дополнительных пакетов, применяемых вместе со стандартным набором графических средств и обеспечивающих использование альтернативного синтаксиса и интеграцию с другими пакетами.

2.1 Загрузка графических пакетов

Графические средства не заложены в ядро LaTeX 2ε, а поддерживаются стандартными пакетами-расширениями color, graphics и graphicx. Эти пакеты опираются на свойства, которые отсутствуют в самом TeX'е, но должны быть реализованы в «драйвере», используемом для вывода на печать dvi-файла. К сожалению, драйверы не обладают одинаковыми свойствами и различаются даже внутренним способом обращения к этим расширениям. Поэтому все эти пакеты распознают опции наподобие «dvips», указывающие на то, какой драйвер используется.

Оба пакета graphics и graphicx могут быть использованы для изменения масштаба, вращения и зеркального отражения LaTeX'овского материала или для включения графических файлов, подготовленных другими программами. Разница между этими пакетами заключается в том, что в пакете graphics ис-

пользуется комбинация макро со «стандартным» или TeXоподобным синтаксисом, тогда как в «расширенном и улучшенном» пакете graphicx для указания необязательных параметров команд `\includegraphics` и `\rotatebox` применяется интерфейс типа «параметр—значение».

2.1.1 Опции пакета, определяющие драйвер устройства

Важно отметить, что при использовании LaTeX'овских графических пакетов после включения графического файла или совершения каких-либо геометрических преобразований на выводимой странице резервируется необходимое место. Однако реальное включение или выполнение требуемых преобразований и затем вывод корректного результата являются задачей *драйвера устройства* (dvips, xdvi, dvipsone и т. д.). Некоторые драйверы, такие, как программы предварительного просмотра, не способны выполнять отдельные функции и поэтому могут показывать набранный материал перекрывающим окружающий текст. Поэтому имя используемого вами драйвера должно быть передано графическому пакету в виде *необязательного параметра* (опции), например:

```
\usepackage[dvips]{graphics}
```

если вы пользуетесь программой dvips Тома Рокицки, или

```
\usepackage[textures]{graphicx}
```

если вы работаете на Макинтоше и пользуетесь программой Textures фирмы Blue Sky. В табл. 2.1 перечислены поддерживаемые в данный момент драйверы и присущие им ограничения. Список поддерживаемых драйверов постепенно расширяется, поэтому, чтобы узнать о поддержке драйвера, отсутствующего в этой таблице, стоит заглянуть в документацию, прилагающуюся к пакету.

Используемый по умолчанию драйвер можно указать в *конфигурационном* файле `graphics.cfg` следующим способом:

```
\ExecuteOptions{emtex}
```

В данном примере, если при вызове графических пакетов необязательный параметр драйвера не указан, то они воспользуются инструкциями для TeX'овской реализации emTeX для PC.

Помимо необязательных параметров, указывающих тип драйвера, графические пакеты поддерживают некоторые другие параметры, позволяющие контролировать наличие (или отсутствие) отдельных свойств:

`draft` Подавляет все «специальные» свойства, такие, как включение внешних графических файлов в окончательный результат. Внешний вид страницы не изменится, поскольку LaTeX'у по-прежнему будет известна информация о размере бокса, обрамляющего графический материал. Этот параметр особенно полезен, когда процесс создания документа еще не завершен и вы не хотите каждый раз загружать (часто очень большие) графические файлы. В режиме `draft` изображение заменяется боксом соответствующего размера, содержащим имя внешнего файла.

Таблица 2.1. Обзор цветовых и графических возможностей различных драйверов устройств.

Опция	*Автор*	*Свойства*
`dvips`	Т. Рокицки	все функции
`dvialw`	Н. Бийби	только включение файла с изменением масштаба
`dvipdf`	С. Лесенко	все функции
`dvilaser`	Arbortext	только включение файла с изменением масштаба
`dvipsone`	Y&Y	все функции
`dvitops`	Дж. Кларк	все функции, но не допускаются вложенные вращения
`dviwin`	Х. Сендукас	включение файлов
`dviwindo`	Y&Y	все функции
`dvi2ps`	встроено	только включение файла с изменением масштаба
`emtex`	Э. Маттес	только включение файла без изменения масштаба
`ln`	Б. Келли	включение файлов для принтера LN03 фирмы DEC
`oztex`	Э. Треворов	включение файлов, цвет, вращение
`pctexps`	PCTeX	включение файлов, цвет, вращение
`pctexwin`	PCTeX	включение файлов, цвет, вращение
`pctex32`	PCTeX	все функции
`pctexhp`	PCTeX	только включение файлов
`psprint`	Э. Треворов	только включение файлов
`pubps`	Arbortext	вращение, включение файлов
`truetex`	Р. Кинч	включение графики и частично цвет
`tcidvi`	Р. Кинч	TrueTeX с дополнительной поддержкой программы Scientific Word
`textures`	Blue Sky	все функции для программы Textures

`hiresbb` В PostScript'овских файлах вместо стандартных комментариев, указывающих размер обрамляющего бокса и имеющих вид `%%BoundingBox` (которые должны иметь целочисленные значения), ищет комментарии, имеющие вид `%%HiResBoundingBox` (которые обычно имеют действительные значения).

`final` Эффект противоположен `draft`. Эта опция может оказаться полезной, например, в том случае, когда режим «draft» был задан в качестве глобального параметра командой `\documentclass`, например, для того, чтобы увидеть переполненные боксы, но вместе с тем вы не хотите отказываться от включения графики. В пакете graphicx этот и предыдущий параметры могут задаваться локально в отдельных командах `\includegraphics`.

`hiderotate` Не показывается материал, подвергнутый вращению (например, в том случае, если программа предварительного просмотра не умеет поворачивать материал и выдает сообщения об ошибках).

`hidescale` Не показывается материал, масштаб которого был изменен (например, когда программа предварительного просмотра не поддерживает изменение масштаба).

2.2 Включение графических файлов

Для включения графических файлов любого формата, поддерживаемого различными драйверами, LaTeX 2ε предоставляет универсальный синтаксис. В общем случае он зависит от расширения файла, которое указывает его тип. В конфигурационных файлах драйверов перечисляются расширения, с которыми этот драйвер может оперировать. Этот список может быть расширен с помощью деклараций, описанных в разд. 2.2.4. Хотя в большинстве примеров в данной книге используется включение PostScript'овских файлов, система также обрабатывает файлы в форматах TIFF, PCX, PICT и др. (конечно, в каждом случае должны быть установлены соответствующие драйверы).

2.2.1 Синтаксис команды `\includegraphics` в пакете graphics

Включить графический файл *file* средствами пакета graphics можно с помощью команды:

`\includegraphics*[`*llx,lly*`][`*urx,ury*`]{`*file*`}`

При отсутствии необязательных параметров размер изображения определяется путем считывания внешнего файла *file* (содержащего само изображение или его описание, см. ниже).

Если аргументы [*urx,ury*] заданы, они должны описывать координаты верхнего правого угла изображения в виде пары значений, выраженных в единицах, поддерживаемых TeX'ом. По умолчанию в качестве единицы измерения выбираются большие PostScript'овские пункты, т. е. `[1in,1in]` и `[72,72]` эквивалентны. Если задана лишь одна пара необязательных параметров, то левый нижний угол изображения считается расположенным в точке `[0,0]`. В противном случае координаты этой точки задаются парой [*llx,lly*].

Вариант команды `\includegraphics` со звездочкой «вырезает» из графического изображения часть, соответствующую обрамляющему боксу. Обычный вариант (без *) позволяет любой части изображения, выступающей за границы обрамляющего бокса, накладываться на окружающий текст.

В приведенных ниже примерах команда `\includegraphics` используется вместе с другими командами из пакета graphics, чтобы продемонстрировать различные способы манипулирования включенными изображениями. (Синтаксис этих команд подробно обсуждается в разд. 2.3.) Для этой цели применяется небольшая программа на PostScript'е (помещенная в файл `w.eps`), рисующая большую прописную букву W и несколько линий. Текст программы приведен на рис. 2.1 на развороте. Обратите внимание на команду `BoundingBox`, определяющую, что изображение простирается от точки `100, 100` (в больших пунктах) до точки `172, 172`, т. е. его естественные размеры равны одному дюйму в ширину и одному дюйму в высоту.

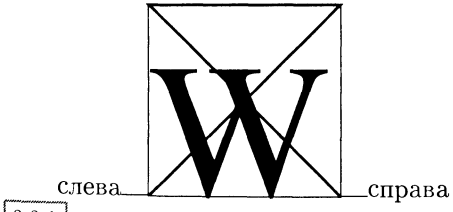
В последующих примерах команда `\fbox` используется для визуализации области, которую LaTeX резервирует для включаемой графики (во всех примерах, хотя явно это указано только в первом, параметр `\fboxsep` равен нулю

```
%! Рисунок в формате Encapsulated PostScript, используемый в примерах
%%BoundingBox:100 100 172 172
100 100       translate % начало координат помещаем в точку 100 100
  0   0       moveto    % определяем текущее положение
 72  72       rlineto   % намечаем диагональную линию
 72 neg 0     rlineto   % намечаем горизонтальную линию
 72  72 neg   rlineto   % намечаем другую диагональную линию
                        % рисуем (проводим) намеченные линии
stroke
  0   0       moveto    % переопределяем текущее положение
/Times-Roman findfont   % выбираем шрифт Times-Roman
 72           scalefont % размер шрифта выбираем равным 1 дюйму
              setfont   % назначаем его текущим шрифтом
(W) show                % рисуем прописную букву W
```

Рис. 2.1. Содержимое файла `w.eps`.

для подавления дополнительных пробелов, добавляемых в обычных условиях командой `\fbox`, а базовая линия рисуется горизонтальными линиями, создаваемыми командой `\HR`, представляющей собой сокращенную запись команды `\rule{1em}{0.4pt}`).

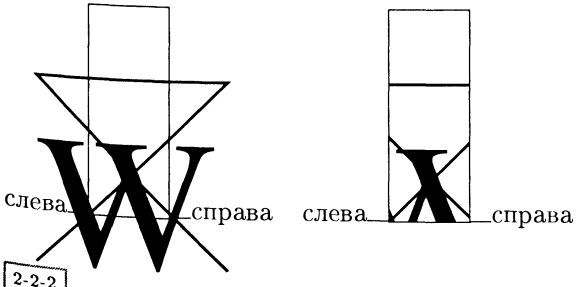
В первом примере показано включение графического файла `w.eps` в натуральную величину. В данном случае картинка и ее обрамляющий бокс идеально совпадают.



2-2-1

```
\usepackage {graphics}
\setlength\fboxsep{0pt}
слева\HR
  \fbox{\includegraphics{w.eps}}%
\HR справа
```

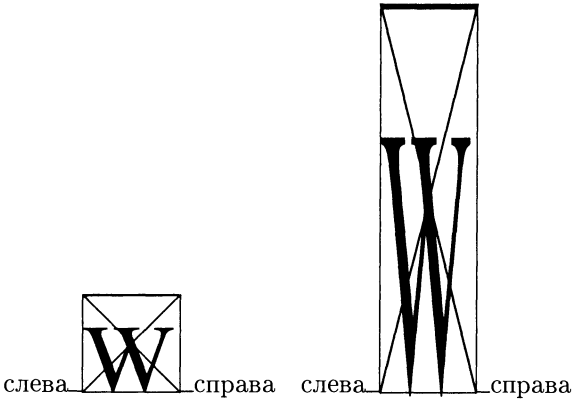
Теперь определим бокс так, чтобы часть его соответствовала изображению, а часть лежала вне его, т. е. так, чтобы некоторые части изображения оказались за пределами бокса, перекрывая, таким образом, материал, окружающий текст. Если применяется вариант команды со звездочкой, то из картинки вырезается соответствующая боксу часть изображения, как это показано на рисунке справа.



2-2-2

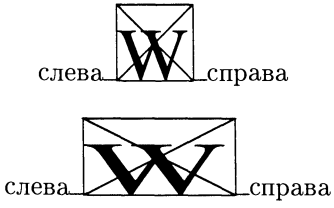
```
\usepackage {graphics}
слева\HR
 \fbox{\includegraphics
 [120,120][150,200]{w.eps}}%
\HR справа \quad
слева\HR
 \fbox{\includegraphics*
 [120,120][150,200]{w.eps}}%
\HR справа
```

Следующие примеры показывают, как можно использовать команды `\scalebox` и `\resizebox` совместно с `\includegraphics`. В обоих случаях можно либо явно указать изменение одного размера, позволив другому изменяться пропорционально, либо явно указать изменения обоих размеров при деформации образа. Синтаксис этих команд подробно обсуждается в разд. 2.3.




```
\usepackage {graphics}
слева\HR
 \fbox{\scalebox{.5}{%
 \includegraphics{w.eps}}}%
\HR справа
\quad
слева\HR
 \fbox{\scalebox{.5}[2]{%
 \includegraphics{w.eps}}}%
\HR справа
```

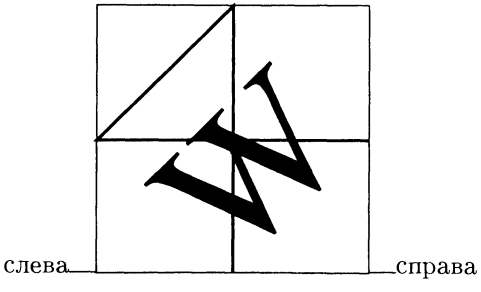
2-2-3




```
\usepackage {graphics}
слева\HR
  \fbox{\resizebox{10mm}{!}{%
    \includegraphics{w.eps}}}%
\HR справа
\bigskip\par
слева\HR
  \fbox{\resizebox{20mm}{10mm}{%
    \includegraphics{w.eps}}}%
\HR справа
```

2-2-4

Добавление вращений делает все еще более интересным:




```
\usepackage {graphics}
слева\HR
  \fbox{\rotatebox{45}{%
    \includegraphics{w.eps}}}%
\HR справа
```

2-2-5

2.2.2 Синтаксис команды \includegraphics в пакете graphicx

Предоставляемый расширенным графическим пакетом graphicx синтаксис для включения созданных внешними средствами графических файлов является немного более прозрачным и дружественным.

`\includegraphics*[`*key val list*`]{`*file*`}`

Вариант этой команды со звездочкой существует только для обеспечения совместимости со стандартной версией пакета `\includegraphics`, описанной разд. 2.2.1. Он эквивалентен заданию ключа `clip`.

Список *key val list* является списком перечисляемых через запятую пар *key*=*value*, обозначающих параметры и принимаемые ими значения. Для булевых параметров простое указание параметра равносильно *key*=`истина`, а отсутствие его упоминания — *key*=`ложь`. Список возможных параметров:

`bb` «Обрамляющий бокс» изображения. Его поле, соответствующее значению, должно содержать четыре величины, разделенные пробелами.

`bbllx` Нижняя левая координата x (устаревшая форма[1]).

`bblly` Нижняя левая координата y (устаревшая форма[1]).

`bburx` Верхняя правая координата x (устаревшая форма[1]).

`bbury` Верхняя правая координата y (устаревшая форма[1]).

`hiresbb` Этот параметр заставляет LaTeX вместо обычных `%%BoundingBox` искать комментарии в виде `%%HiResBoundingBox`. Некоторые приложения используют его для более точного определения обрамляющих боксов, так как эти значения в обычном случае могут быть лишь целыми числами. Это булев параметр, принимающий значение либо «`истина`», либо «`ложь`».

`viewport` У этого параметра (как и у `bb`) четыре аргумента, но в данном случае точка отсчета выбирается относительно указанного в файле обрамляющего бокса. Так, чтобы увидеть квадрат со стороной 20 больших пунктов в нижней левой части рисунка, надо указать `viewport=0 0 20 20`.

`trim` Параметр похож на `viewport`, но четыре аргумента соответствуют областям, удаляемым из левой, нижней, правой и верхней частей включаемого изображения.

`natheight` Естественная высота изображения[2].

`natwidth` Естественная ширина изображения[2].

`angle` Угол поворота (в градусах, против часовой стрелки).

[1] Существует только для обеспечения обратной совместимости. Параметры `[bbllx=a, bblly=b, bburx=c, bbury=d]` эквивалентны параметру `[bb = a b c d]`, поэтому следует использовать более современный вариант.

[2] Эти аргументы могут быть использованы для задания левой нижней координаты, равной `(0 0)`, а верхней правой координаты — равной (`natwidth natheight`), и, таким образом, эквивалентны `bb=0 0 w h`, где `w` и `h` являются величинами, указанными для этих двух параметров.

`origin` Центр вращения, сходный с параметром `origin` команды `\rotatebox`, описанной на с. 68 и на рис. 2.2 на с. 68.

`width` Требуемая ширина (ширина изображения делается равной этой величине).

`height` Требуемая высота (высота изображения делается равной этой величине).

`totalheight` Требуемая полная высота (полная высота изображения делается равной этой величине). Этот параметр следует использовать вместо `height`, если изображения вращаются более, чем на 90 градусов, поскольку высота может исчезнуть (и превратиться в `depth`) и LaTeX'у будет трудно выполнить требования пользователя.

`keepaspectratio` Булева переменная, которая принимает значения «`истина`» или «`ложь`» (замечание относительно значений по умолчанию см. выше). В случае значения «истина» задание параметров `width` и `height` не деформирует изображение, но его масштаб меняется таким образом, чтобы ни `width`, ни `height` *не превысили* заданных границ.

`scale` Масштаб.

`clip` Извлечение части изображения и помещение ее в обрамляющий бокс. Это булев параметр, принимающий значения «`истина`» или «`ложь`».

`draft` Локальное переключение в режим draft. Как и `clip`, это булев параметр.

`type` Тип графики.

`ext` Расширение файла, содержащего изображение.

`read` Расширение файла, «считываемого» LaTeX'ом.

`command` Любая операция, которая должна быть выполнена над файлом.

Если в первых семи ключах (начиная с `bb` и заканчивая `trim`) размеры задаются без указания единиц измерения, то предполагается, что это TeX'овские «большие пункты» (равные пунктам в PostScript'е).

Первые девять ключей (начиная с `bb` и заканчивая `natwidth`) определяют размер изображения. Он должен быть указан в том случае, когда графический файл не может быть прочитан TeX'ом, или содержит неверную метрическую информацию, или если вы хотите вырезать из рисунка некоторый прямоугольник.

Следующие семь ключей (начиная с `angle` и заканчивая `scale`) относятся к изменению масштаба и повороту включаемого материала. Сходные результаты могут быть получены с помощью пакета graphics и команды `\includegraphics`, помещаемой в аргумент команд `\resizebox`, `\rotatebox` или `\scalebox` (см. примеры выше в разд. 2.2.1 и детальное описание этих команд в разд. 2.3).

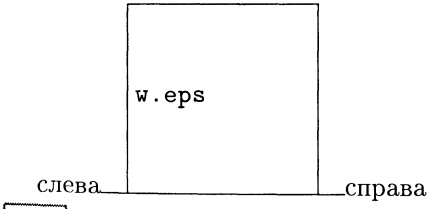
Важно отметить, что параметры считываются слева направо, поэтому `[angle=90, totalheight=2cm]` означает повернуть на 90 градусов и затем установить высоту равной 2 см, тогда как `[totalheight=2cm, angle=90]` привело бы к тому, что равной двум сантиметрам стала бы итоговая *ширина*.

По умолчанию LaTeX резервирует место для рисунка на основе информации, указанной либо в файле, либо в необязательных аргументах. Если какая-нибудь часть рисунка в действительности выступает за границы этого пространства, она накладывается на окружающий текст. Если используется вариант команды со звездочкой или указан параметр `clip`, то выступающая часть изображения не будет напечатана.

Последние четыре ключа (`type`, `ext`, `read`, `command`) подавляют синтаксический анализ имени файла. При их использовании обязательный аргумент *file* не должен содержать расширения (см. описание команды `\DeclareGraphicsRule` ниже).

Ниже повторно приведены несколько примеров из разд. 2.2.1, иллюстрирующих синтаксис пакета graphicx, а также показаны предоставляемые им дополнительные возможности. В большинстве случаев синтаксис новой версии пакета более понятен, чем синтаксис предыдущей. В простейшем случае, когда не указаны никакие необязательные аргументы, синтаксис команды `\includegraphics` в обоих пакетах совпадает.

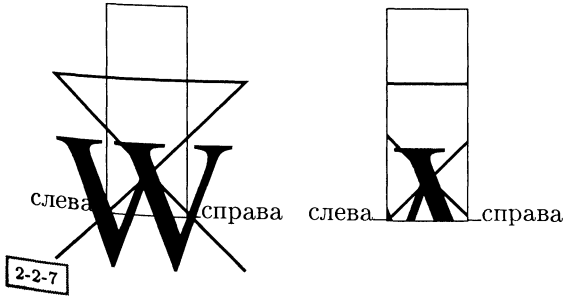
Если указан параметр `draft`, будет видна лишь рамка, обозначающая обрамляющий бокс. На уровне отдельного изображения такая возможность в пакете graphics отсутствует.



2-2-6

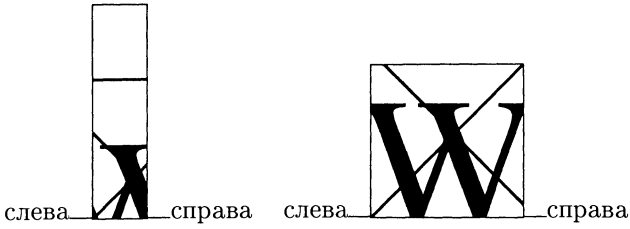
```
\usepackage {graphicx}
слева\HR
 \includegraphics[draft]{w.eps}%
\HR справа
```

Результаты применения параметров `bb`, `clip`, `viewport` и `trim` видны на следующих примерах.



2-2-7

```
\usepackage {graphics}
слева\HR\fbox{%
 \includegraphics
 [bb=120 120 150 200]%
 {w.eps}}\HR справа
\quad
слева\HR\fbox{%
 \includegraphics
 [bb=120 120 150 200,clip]%
 {w.eps}}\HR справа
```




```
\usepackage {graphics}
слева\HR\fbox{%
 \includegraphics
   [viewport=20 20 40 100,
    clip]%
{w.eps}}\HR справа
\quad
слева\HR\fbox{%
 \includegraphics
   [trim=5 5 10 10,clip]%
   {w.eps}}\HR справа
```

2-2-8

Если вы хотите изменить масштаб изображения, воспользуйтесь параметром `scale`. С его помощью масштаб вдоль разных направлений изменяется одинаково.


```
\usepackage {graphics}
слева\HR
  \fbox{\includegraphics
  [scale=.5]{w.eps}}%
\HR справа
```

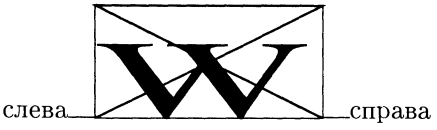
2-2-9

Чтобы сделать размеры изображения равными заданным значениям, воспользуйтесь параметрами `width` или `height`:


```
\usepackage {graphics}
слева\HR
  \fbox{\includegraphics
  [width=10mm]{w.eps}}%
\HR справа
```

2-2-10

Если вы укажете оба параметра одновременно, то сможете деформировать рисунок таким образом, чтобы он соответствовал указанному прямоугольнику, как иллюстрирует следующий пример:

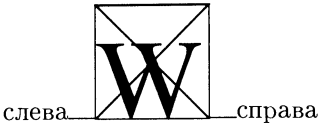



```
\usepackage {graphics}
слева\HR
  \fbox{\includegraphics
  [height=15mm,width=30mm]{w.eps}}%
\HR справа
```

2-2-11

Однако при этом вы можете обеспечить неизменность пропорций рисунка, дополнительно указав ключ `keepaspectratio`. Тогда LaTeX постарается

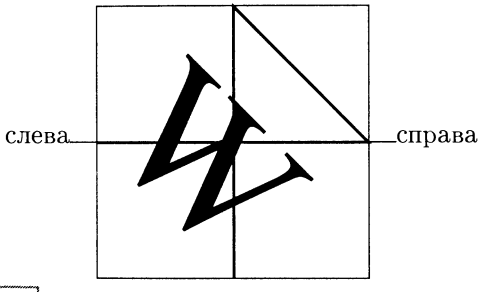
сделать так, чтобы размер изображения наилучшим образом соответствовал указанному вами прямоугольнику.



2-2-12

```
\usepackage {graphics}
слева\HR
  \fbox{\includegraphics
  [height=15mm,width=30mm,%
   keepaspectratio]{w.eps}}%
\HR справа
```

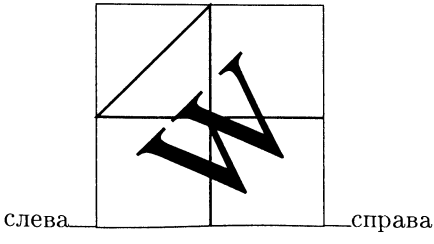
Еще один уровень сложности добавляет параметр `angle`. Обратите особое внимание, насколько больше становится повернутый LaTeX'овский бокс и где изображение размещается по отношению к базовой линии. Как видно из следующих примеров, в зависимости от используемого значения параметра `angle` изображение может сдвинуться вверх или вниз.



2-2-13

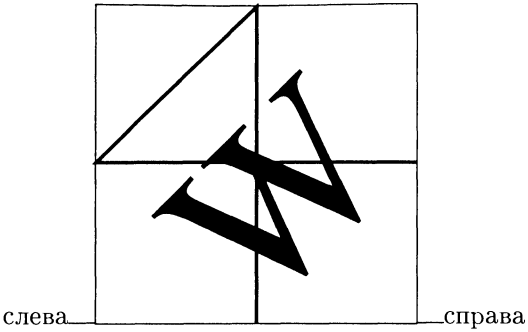
```
\usepackage {graphics}
слева\HR
  \fbox{\includegraphics
  [angle=-45]{w.eps}}%
\HR справа
```

Но самое интересное начинается, если вы зададите одновременно и размер рисунка, и угол его поворота, так как очередность их задания имеет значение. Пакет graphicx обрабатывает параметры *слева направо*. Вы должны быть особенно внимательны, если хотите повернуть изображение и сделать его определенной высоты. В следующих примерах демонстрируется разница между случаями задания угла поворота до и после команды изменения размера: в первом случае изображение поворачивается, а затем изменяется масштаб полученного результата, тогда как во втором случае сначала изменяется масштаб изображения, а потом выполняется его поворот.



2-2-14

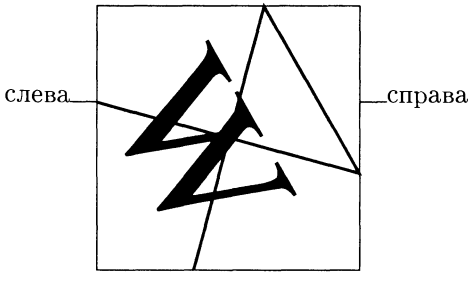
```
\usepackage {graphics}
слева\HR\fbox
   {\includegraphics
      [angle=45,width=30mm]%
      {w.eps}%
   }%
\HR справа
```




```
\usepackage {graphics}
слева\HR\fbox
   {\includegraphics
      [width=30mm,angle=45]%
      {w.eps}%
   }%
\HR справа
```

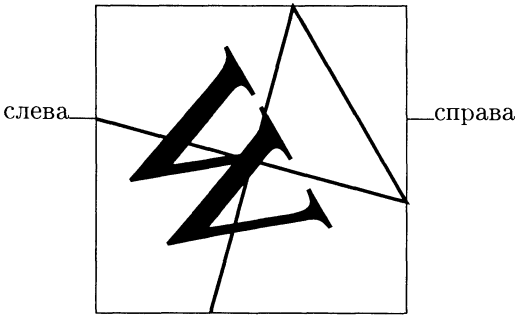
2-2-15

LaTeX рассматривает высоту и глубину поворачиваемого обрамляющего бокса отдельно, а `height` относится только к высоте, т. е. его действие не распространяется на глубину. В общем случае, когда хочется, чтобы совокупная высота повернутого изображения равнялась заданному значению, вы должны указать параметр `totalheight` (определения различных размеров, описывающих LaTeX'овский бокс см. на рис. 2.2 на с. 68). Конечно, для достижения специальных эффектов вы можете манипулировать поворотами и комбинациями параметров `height` и `width` по своему желанию. Ниже показаны лишь некоторые комбинации параметров и получаемые с их помощью результаты.



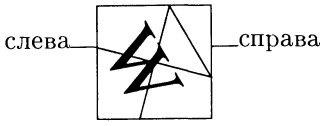

```
\usepackage {graphics}
слева\HR\fbox{%
  \includegraphics[angle=-60]
     {w.eps}}\HR справа
```

2-2-16



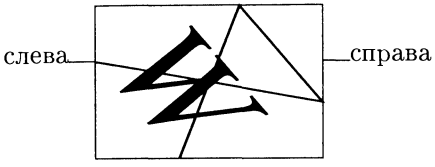

```
\usepackage {graphics}
слева\HR\fbox{%
  \includegraphics
        [angle=-60,%
         height=15mm]%
     {w.eps}}\HR справа
```

2-2-17



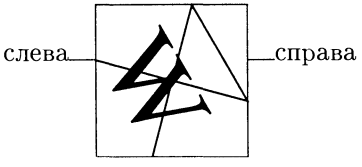
2-2-18

```
\usepackage {graphics}
слева\HR\fbox{%
  \includegraphics
        [angle=-60,%
         totalheight=15mm]%
    {w.eps}}\HR справа
```



2-2-19

```
\usepackage {graphics}
слева\HR\fbox{%
  \includegraphics
        [angle=-60,%
         totalheight=20mm,%
         width=30mm]%
    {w.eps}}\HR справа
```



2-2-20

```
\usepackage {graphics}
слева\HR\fbox{%
  \includegraphics
        [angle=-60,%
         totalheight=20mm,%
         width=30mm,%
         keepaspectratio]%
    {w.eps}}\HR справа
```

2.2.3 Присвоение параметрам пакета graphicx значений по умолчанию

Вместо того чтобы снова и снова вводить одно и то же множество пар «параметр—значение» в каждой отдельной команде `\includegraphics`, для соответствующих этим парам параметров можно глобально задать значения по умолчанию с помощью декларации `\setkeys`, доступной всегда, когда используется пакет graphicx.

`\setkeys{`*identifier*`}{`*key val list*`}`

identifier — это произвольная строка, определяемая создателем макро. Например, для команды `\includegraphics` была выбрана строка `Gin`. Список *key val list* представляет собой перечисление через запятую пар «параметр—значение».

В качестве примера рассмотрим случай, когда используется пакет graphicx и ширина всех рисунков должна быть сделана равной ширине текста. Тогда вы должны указать следующее:

```
\setkeys{Gin}{width=\linewidth}
```

В результате ширина всех рисунков, включенных с помощью команды `\includegraphics`, будет автоматически установлена равной текущей ширине строки. (В этом случае обычно предпочтительнее использовать коман-

ду \linewidth, а не \columnwidth, поскольку первая меняет свое значение в зависимости от окружающей обстановки, например в окружении quote и т. д.).

Аналогичным образом вы можете указать значения по умолчанию для любого ключа, используемого с командой \rotatebox (еще одной командой из пакета graphicx, синтаксис которой имеет вид «параметр—значение»). Определяющий ее идентификатор *identifier* имеет вид Grot. Таким образом, команда

```
\setkeys{Grot}{origin=ct}
```

определяет, что в качестве начала координат origin во всех командах \rotatebox следует использовать точку ct, если локально не задано другое.

2.2.4 Декларации, управляющие включением рисунков

Тогда как пары «параметр—значение» можно задавать только при использовании пакета graphicx, описываемые в данном разделе декларации можно применять как с graphics, так и с graphicx.

Список директорий, в которых LaTeX'у следует искать графические файлы, может быть определен командой \graphicspath, синтаксис которой имеет вид:

\graphicspath{*dir-list*}

где *dir-list* — это список директорий, каждая из которых заключена в фигурные скобки {} (даже если этот список состоит только из одной директории). Например, команда \graphicspath{{./eps/}{./tiff/}} указывает LaTeX'у искать в поддиректориях eps и tiff текущей директории. По умолчанию LaTeX ищет графические файлы в тех же директориях, в которых он ищет остальные файлы.

\DeclareGraphicsExtensions{*ext-list*}

Команда \DeclareGraphicsExtensions позволяет вам определить поведение системы в том случае, когда в аргументе команды \includegraphics не указано расширение файла. Аргумент {*ext-list*} представляет собой разделенный запятыми список расширений файлов. Полные имена файлов получаются путем добавления к имени файла каждого расширения из списка *ext-list* до тех пор, пока не будет найден файл, чье имя совпадает с вычисленным.

Поскольку предыдущий алгоритм проверяет, существует ли файл, чтобы выяснить, какое расширение использовать, если в команде \includegraphics не указано расширение, во время работы LaTeX'a, графический файл обязательно должен существовать. Однако если расширение файла *указано*, например, не \includegraphics{gr}, а \includegraphics{gr.eps}, то существова-

ние графического файла во время работы LaTeX'а не обязательно[1]. Так как LaTeX'у необходимо знать размер изображения, он должен быть указан в аргументах команды `\includegraphics` или в файле, реально прочитанном LaTeX'ом (это может быть сам графический файл или другой файл, указанный параметром `read` или построенный на основе списка расширений файлов. В последнем случае этот файл должен существовать во время работы LaTeX'а).

В приведенном ниже примере команда `\includegraphics` сначала будет искать файл `file.ps`, а если такого файла не существует, то `file.ps.gz`:

```
\DeclareGraphicsExtensions{.ps,.ps.gz}
  \includegraphics{file}
```

Если вы хотите быть уверенными в том, чтобы всегда указывалось полное имя файла, можете воспользоваться следующей декларацией. Кроме этого, обратите внимание, что в представленных ниже примерах размер рисунка в команде `\includegraphics` в каждом случае указывается явным образом.

```
\DeclareGraphicsExtensions{{}}
  \includegraphics[1in,1in]{file.pcx}
  \includegraphics[75pt,545pt][50pt,530pt]{file.pcx}
  \includegraphics[bb=75 545 50 530]{file.pcx}
```

Действие, которое должно быть выполнено в случае нахождения файла с указанным расширением, задается следующей командой:

`\DeclareGraphicsRule{`*ext*`}{`*type*`}{`*read-file*`}{`*cmd*`}`

Можно указать любое количество таких деклараций. Ниже объясняется значение каждого из аргументов.

ext Расширение файла. Оно может быть задано явно или, если аргумент команды `\includegraphics` не имеет расширения, может быть определено из списка расширений, указанного в аргументе *ext-list* команды `\DeclareGraphicsExtensions`. Для задания действий по умолчанию, предпринимаемых относительно всех явно не указанных расширений, может быть использована звездочка (`*`), т. е. команда

```
\DeclareGraphicsRule{*}{eps}{*}{}
```

указывает считать все неописанные расширения eps-файлами, а соответствующие графические файлы просматриваются в поисках комментариев, имеющих вид `%%BoundingBox`.

type «Тип» используемого файла. Все файлы одного и того же типа вводятся с помощью одной и той же внутренней команды (которая должна быть определена в соответствующем файле драйвера). Например, все файлы с расширениями `.ps`, `.eps` или `.ps.gz` должны принадлежать одному и тому же типу `eps`.

[1] Например, файл может быть создан «на лету» путем указания *управляющего* аргумента в команде `\DeclareGraphicsRule`.

read-file Расширение файла, который должен быть считан для определения размера графического материала. Оно может совпадать с аргументом *ext*, но в случае использования сжатых или бинарных изображений, которые LaTeX не умеет распознавать, метрическая информация (размер обрамляющего бокса) обычно помещается в отдельный файл. Например, PostScript'овским файлам, сжатым программой gzip и имеющим расширение `.ps.gz`, может соответствовать читаемый файл с расширением `.ps.bb`. Если аргумент *read-file* пуст, т. е. имеет вид `{}`, то для определения размера система не будет искать внешний файл и размер должен быть указан в аргументе команды `\includegraphics`. Если в драйвере предусмотрена процедура прочтения файлов, содержащих размеры, для файлов типа *type*, то она и реализуется, в противном случае используется процедура прочтения `.eps`-файлов. Следовательно, если никакой другой формат не указан, вы можете выбрать размер растрового изображения, используя синтаксис для PostScript'овских рисунков, т. е. из строки `%%BoundingBox`.

cmd Эта команда подставляется в качестве аргумента команды `\special` вместо имени файла. Обычно аргумент *cmd* пуст, но в случае сжатых файлов вам может захотеться разархивировать графический файл перед тем, как включать его в предназначенный для печати документ, если такая операция допускается драйвером. Например, при работе с драйвером dvips можно задать

```
\DeclareGraphicsRule{.ps.gz}{eps}{.ps.bb}{`gunzip -c #1}
```

где аргумент `#1` обозначает полное имя файла. В этом случае последний аргумент заставляет драйвер dvips выполнить команду gunzip для разархивации файла перед тем, как вставить его в выходной файл на PostScript'е.

Различные возможности, предоставляемые аргументами команды `\DeclareGraphicsRule`, показаны в табл. 2.2.

Таблица 2.2. Аргументы команды `\DeclareGraphicsRule`.

	ext	*type*	*read-file*	*cmd*
базовый PostScript	`.ps`	eps	`.ps`	
	`.eps`	eps	`.eps`	
динамическая разархивация	`.pz`	eps	`.bb`	`` `gunzip -c #1 ``
	`.ps.gz`	eps	`.ps.bb`	`` `gunzip -c #1 ``
	`.eps.gz`	eps	`.eps.bb`	`` `gunzip -c #1 ``
форматы, связанные с DOS	`.tif`	tiff		
	`.pcx`	bmp		
	`.bmp`	bmp		
	`.msp`	bmp		
форматы, связанные с Macintosh	`.pict`	pict		
	`.pntg`	pntg		

Система, описанная до сих пор, может привести к некоторым проблемам, если расширение *ext* не соответствует аргументу *type*. Например, может встретиться набор PostScript'овских файлов с именами `file.1`, `file.2`, ... Пакет graphics не умеет автоматически определять, что это — файлы на PostScript'е, если только для каждого из них в команде `\includegraphics` не задано соответствующее значение параметра `type=eps`. Для того, чтобы справиться с такой ситуацией, вы можете задать принятый по умолчанию тип с помощью определения `\DeclareGraphicsRule` для типа `*`, как это было описано выше.

2.3 Манипулирование графическими LaTeX'овскими объектами

Помимо команды `\includegraphics` в пакетах graphics и graphicx реализован некоторый набор команд для операций с графикой. За исключением команды `\rotatebox`, которая в пакете graphicx имеет несколько иной синтаксис, синтаксис остальных команд в обоих пакетах идентичен.

2.3.1 Изменение масштаба LaTeX'овского бокса

Команда `\scalebox` позволяет вам увеличивать или уменьшать размер текста или другого LaTeX'овского материала на величину масштабного множителя:

`\scalebox{`*scalefact*`}{`*material*`}`

Первый из двух обязательных аргументов (*scalefact*) задает множитель, на который умножаются оба размера аргумента *material*. Следующий пример демонстрирует, как она работает:[1]

Этот текст крупный.
Этот текст нормальный.
Этот текст мелкий

2-3-1

```
\usepackage {graphics}
\scalebox{2}{%
 Этот текст крупный.}\\
 Этот текст нормальный.\\
\scalebox{0.5}{%
 Этот текст мелкий.}
```

Дополнительный необязательный аргумент, если он присутствует, используется для задания отдельного масштабного множителя по вертикали:

`\scalebox{`*h-scale*`}[`*v-scale*`]{`*material*`}`

[1] В этом и последующих примерах для правильной работы с кириллицей следует ввести команды, упомянутые на с. 7. — *Прим. ред.*

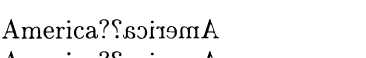
Это демонстрируется в следующем примере, где также показано, как с помощью стандартной LaTeX'овской команды \parbox может быть изменен масштаб сразу нескольких строк.


```
\usepackage {graphics}
\framebox{\scalebox{2}{%
\parbox{.56in}{Америка и\\
                Европа}}}
\framebox{\scalebox{2}[1.5]{%
\parbox{.56in}{Америка и\\
                Европа}}}
```

2-3-2

\reflectbox{*material*}

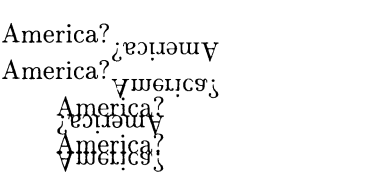
Это — удобная аббревиатура для команды \scalebox{-1}[1]{*material*}, действие которой видно из следующего примера:




```
\usepackage {graphics}
America?\reflectbox{America?}     \\
America?\scalebox{-1}[1]{America?}
```

2-3-3

Кроме этого могут быть получены и более интересные специальные эффекты. В следующем примере обратите особое внимание на использование команд \makebox с нулевой шириной, которые скрывают от LaTeX'а свое содержимое и, таким образом, предоставляют возможность тонкой регулировки положения набранного материала.




```
\usepackage {graphics}
America?\scalebox{-1}{America?}   \\
America?\scalebox{1}[-1]{America?}\\
America?\makebox[0mm][r]{%
          \scalebox{-1}{America?}}\\
\makebox[0mm][l]{America?}%
   \scalebox{1}[-1]{America?}
```

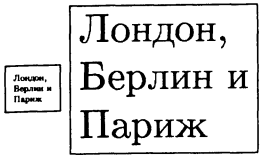
2-3-4

2.3.2 Подгонка под требуемый размер

Следующая команда дает возможность указать LaTeX'у набрать некоторый материал таким образом, чтобы он имел определенный горизонтальный или вертикальный размер:

\resizebox*{*h-dim*}{*v-dim*}{*material*}

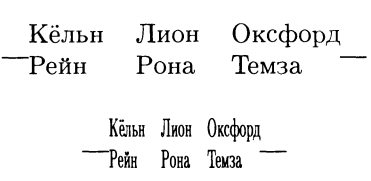
Если при этом необходимо сохранить пропорции изображения, то достаточно указать только один размер, заменив другой символом «!».




```
\usepackage {graphics}
\framebox{\resizebox{5mm}{!}{%
  \parbox{15mm}{Лондон, \\
                Берлин и\\Париж}}}
\framebox{\resizebox{!}{10mm}{%
  \parbox{15mm}{Лондон, \\
                Берлин и\\Париж}}}
```

2-3-5

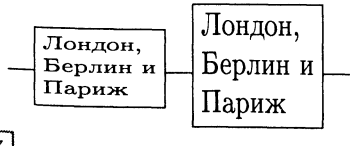
Если оба аргумента *h-dim* и *v-dim* указаны явно, то содержимое рисунка может быть деформировано. В следующем примере базовую линию задает горизонтальная линейка, нарисованная с помощью команды \HR.




```
\usepackage {graphics}
\HR\begin{tabular}{@{}lll@{}}
    Кёльн & Лион & Оксфорд \\
    Рейн  & Рона & Темза
 \end{tabular}\HR\par\bigskip
\HR\resizebox{2cm}{.5cm}{%
 \begin{tabular}{@{}lll@{}}
    Кёльн & Лион & Оксфорд \\
    Рейн  & Рона & Темза
 \end{tabular}}\HR
```

2-3-6

Как обычно, LaTeX'овские команды, содержащие размеры боксов, могут использовать в качестве параметров естественные длины \depth, \height, \totalheight и \width:

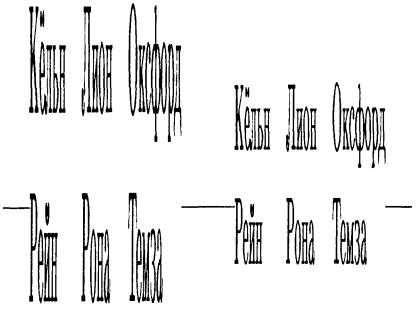



```
\usepackage {graphics}
\HR\framebox{\resizebox{\width}
                       {.7\height}{%
   \parbox{15mm}{Лондон, \\
                 Берлин и\\Париж}}}\H
\framebox{\resizebox{\width}
                     {.7\totalheight}{
  \parbox{15mm}{Лондон, \\
                Берлин и\\Париж}}}\HR
```

2-3-7

Вариант команды \resizebox без звездочки основывается в своих вычислениях на высоте набранного LaTeX'ом материала, тогда как вариант со звездочкой использует суммарную высоту (глубину плюс высоту) LaTeX'овского бокса.

В следующих примерах это различие демонстрируется для текста, имеющего значительную глубину.



```
\usepackage {graphics}
\HR
\resizebox{20mm}{30mm}{%
  \begin{tabular}{@{}lll@{}}
    Кёльн & Лион & Оксфорд \\
    Рейн  & Рона & Темза
  \end{tabular}}\HR
\HR
\resizebox*{20mm}{30mm}{%
  \begin{tabular}{@{}lll@{}}
    Кёльн & Лион & Оксфорд \\
    Рейн  & Рона & Темза
  \end{tabular}}\HR
```

2-3-8

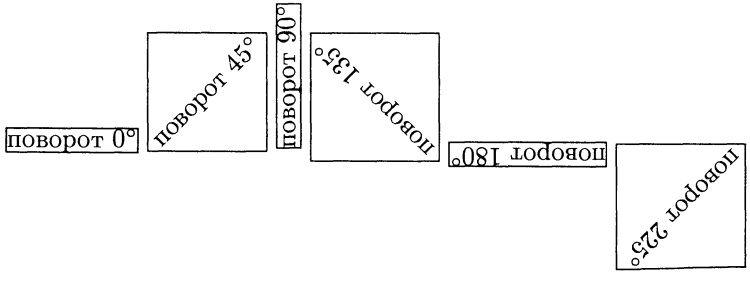
2.3.3 Поворот LaTeX'овского бокса

С помощью команды \rotatebox набранный LaTeX'ом материал может быть повернут на произвольный угол. Содержание данного раздела можно сравнить с описанным в разд. 11.4 книги Goossens et al. (1994) пакетом rotating, созданным Себастианом Ратцем. Хотя пользователям рекомендуется использовать вместо этого пакета описанные в данном разделе команды, он все еще распространен, поскольку предоставляет дополнительные возможности для поворота плавающих объектов.

\rotatebox{*angle*}{*material*}

Аргумент *material* набирается внутри LaTeX'овского бокса, который поворачивается на *angle* градусов против часовой стрелки относительно контрольной точки.

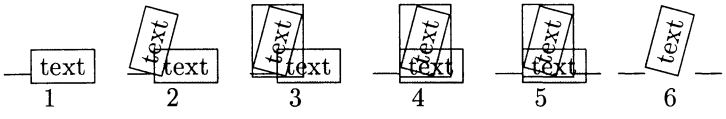
```
\usepackage {graphics}
\setlength{\fboxsep}{0mm}
\newcommand{\MyRot}[1]{\fbox{\rotatebox{#1}{поворот $#1^\circ$}}}
\MyRot{0} \MyRot{45} \MyRot{90} \MyRot{135} \MyRot{180} \MyRot{225}
```



2-3-9

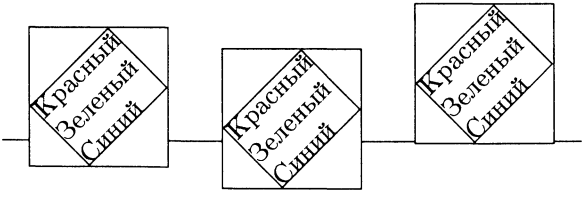
Чтобы понять, где на странице будет размещаться поворачиваемый материал, необходимо рассмотреть используемый алгоритм. Ниже показаны от-

дельные шаги, выполняемые при повороте текста \fbox{text} на 75 градусов. На первом шаге показано положение текста до выполнения поворота. Горизонтальная линия слева указывает положение базовой линии. Сначала аргумент *material* (в данном случае \fbox{text}) помещается в бокс. На пересечении базовой линии и левой границы блока находится контрольная точка, вокруг которой в стандартной ситуации выполняется поворот. Это показано на шаге 2 (для сравнения также показано положение текста до выполнения поворота). Затем алгоритм вычисляет новый обрамляющий бокс, т. е. область, резервируемую для поворачиваемого материала, которая показана на шаге 3. После этого материал горизонтально сдвигается так, чтобы левые границы нового и старого обрамляющих боксов совпадали (шаг 4). Затем текущая позиция TeX'а, в которой будет набираться последующий материал, устанавливается справа от обрамляющего бокса, что показано на 5-м шаге линией, обозначающей базовую линию. На 6-м шаге с помощью базовой линии, проведенной по обе стороны от повернутого материала, показан окончательный результат.



В случае поворота более сложного материала важно помнить, где находится контрольная точка получающегося бокса. В следующем примере показано, как она может быть смещена при помощи необязательного параметра позиционирования команды \parbox.

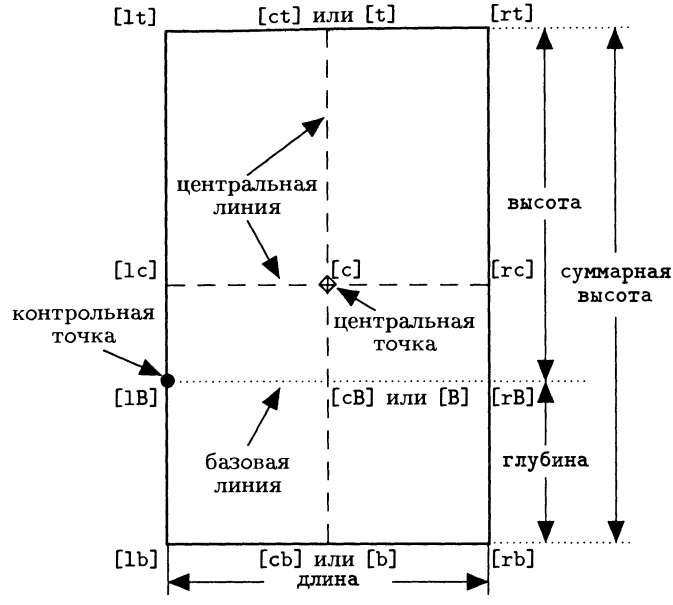
```
\usepackage {graphics}
\setlength{\fboxsep}{0mm}
\providecommand{\HR}{\rule{1em}{0.4pt}}
\HR\fbox{\rotatebox{45}{\fbox{\parbox{4.5em}{Красный\\Зеленый\\Синий}}}}%
\HR
\HR\fbox{\rotatebox{45}{\fbox{\parbox[t]{4.5em}{Красный\\Зеленый\\Синий}}}}%
\HR
\HR\fbox{\rotatebox{45}{\fbox{\parbox[b]{4.5em}{Красный\\Зеленый\\Синий}}}}%
\HR
```



2-3-10

Расширенный графический пакет graphicx предоставляет бóльшую гибкость при задании точки, вокруг которой выполняется поворот, путем указания пар: параметр—значение.

\rotatebox[*key val list*]{*angle*}{*material*}

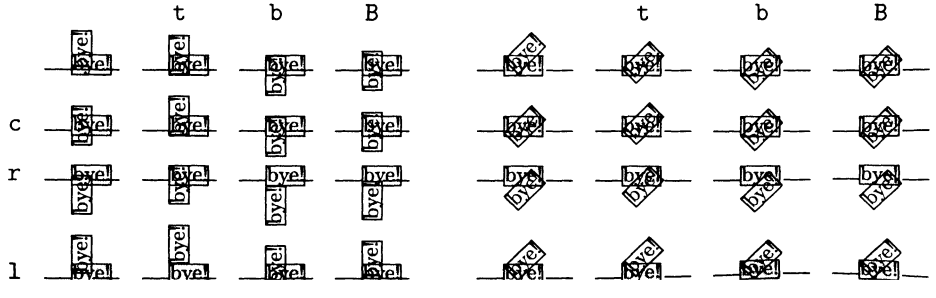


2-3-11

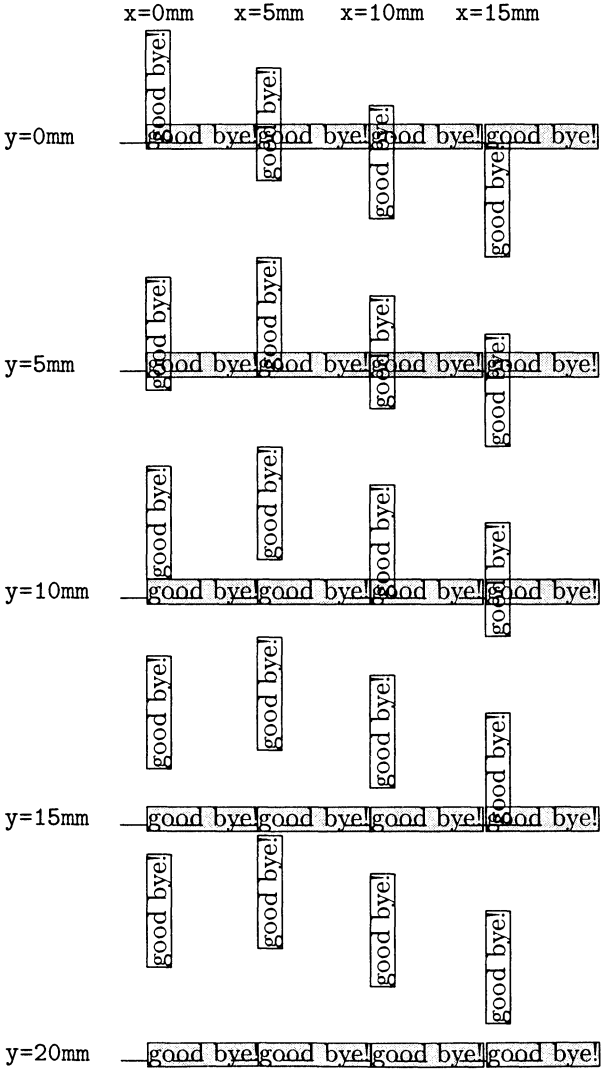
Рис. 2.2. LaTeX'овский бокс и возможные положения контрольных точек `origin`.

В данном случае четырьмя возможными параметрами могут быть `origin`, `x`, `y` и `units`. Возможные значения параметра `origin` показаны на рис. 2.2 (может быть выбрано по одному значению для горизонтального и вертикального выравниваний), на котором также показаны реальные положения контрольных точек относительно LaTeX'овского бокса.

Влияние всех возможных комбинаций параметров `origin` на положение LaTeX'овского бокса показано на рисунке ниже; две приведенные матрицы результатов отвечают повороту на 90° и 45°. Для лучшего понимания происходящего результат показан на фоне исходного положения бокса, который закрашен серым.



Если спецификация `origin` не дает желаемого результата, существует дополнительная возможность указать координаты x и y точки, вокруг которой должен происходить поворот; отсчет координат ведется по отношению к контрольной точке. Для этого используются параметры `x` и `y` и формат `x=`*dim*`, y=`*dim*. Ниже показаны матрица из некоторых выбранных для примера значений параметров и их влияние на бокс, повернутый на 90°.

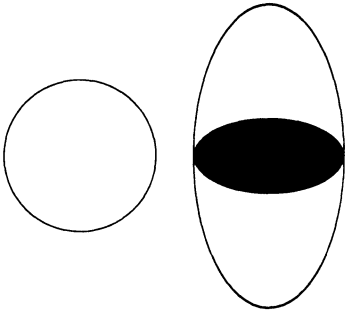


Интерпретацией аргумента *angle* команды `\rotatebox` можно управлять с помощью ключевого слова `units`, указывающего число отсчитываемых против

часовой стрелки единиц измерения, которые составляют полный круг. Значение по умолчанию равно 360, так что команда `units=-360` будет означать, что углы отсчитываются по часовой стрелке. Аналогичным образом команда `units=6.283185` определяет единицей измерения радианы вместо градусов. Вместо того чтобы задавать значение `units` в каждой отдельной команде `\rotatebox`, вы можете указать его значение по умолчанию с помощью декларации `\setkeys`, как описано в разд. 2.2.3.

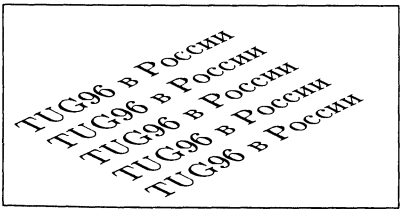
2.3.4 Комбинированные эффекты

Команды изменения масштаба и поворота можно комбинировать и вкладывать *неограниченное число раз*. Их также можно применять к объектам LaTeX'овского окружения picture.



```
\usepackage {graphics,pspicture}
\setlength{\unitlength}{1mm}
\begin{picture}(50,40)
\put(15,20){\circle{20}}
\put(40,20){%
 \scalebox{1}[2]{\circle{20}}}
\put(40,20){%
 \scalebox{1}[.5]{\circle*{20}}}
\end{picture}\qquad
```

2-3-12




```
\usepackage {graphics}
\fbox{\resizebox{5cm}{20mm}{%
    \rotatebox{45}{\parbox{3cm}%
{\raggedright
 TUG96 в России   TUG96 в России
 TUG96 в России   TUG96 в России
 TUG96 в России}}}}
```

2-3-13

2.3.5 Некоторые приложения

Поворот содержимого таблиц

Типичным кандидатом на то, чтобы подвергнуться повороту, является содержимое таблицы, как всей целиком, так и отдельных клеток. Если сделать первое, обычно, достаточно просто, то для осуществления второго часто приходится прибегать к LaTeX'овским «трюкам», чтобы быть уверенным, что отве-

денное для этой клетки пространство соответствует визуально необходимому, а не занимаемому в результате механического поворота. Эту задачу обычно выполняют линейки нулевой ширины или высоты. Так сделано в следующей таблице (переписанной с помощью пакета graphics на основе примера со с. 363 книги Goossens et al. (1994)), в которой заголовки колонок повернуты и занимают слишком много места:

Колонка 1	Колонка 2	Колонка 3
1	2	3
4	5	6
7	8	9

```
\usepackage {graphics}
\begin{tabular}{rrr}
\rotatebox{45}{ Колонка 1} &
\rotatebox{45}{ Колонка 2} &
\rotatebox{45}{ Колонка 3}    \\\hline
1& 2& 3 \\ 4& 5& 6 \\ 7& 8& 9 \\\hline
\end{tabular}
```

2-3-14

Если мы воспользуемся командой \makebox для того, чтобы скрыть ширину повернутых заголовков, и отцентрируем их относительно интервалов фиксированной ширины, помещенных над колонками, то получим гораздо более приемлемый результат:

Колонка 1	Колонка 2	Колонка 3
1	2	3
4	5	6
7	8	9

```
\usepackage {graphics}
% Zero height rule of width of one quad
\newcommand{\IR}[1]{\rule{1em}{0pt}%
  \makebox[0cm][c]{\rotatebox{45}{\ #1}}}
\begin{tabular}{rrr}
\IR{Колонка 1} & \IR{Колонка 2}
                & \IR{Колонка 3} \\\hline
1& 2& 3 \\ 4& 5& 6 \\ 7& 8& 9 \\\hline
\end{tabular}
```

2-3-15

Также существует возможность вложенных поворотов внутри таблиц, когда осуществляются боковые смещения как единого целого. Примеры из табл. 11.3 и 11.4 на с. 365–366 книги Goossens et al. (1994) можно легко переписать с помощью описанных в этой главе стандартных пакетов и пакета высокого уровня rotating, который использует graphics.

Охватывающие скобки

Как мы только что видели, при желании содержимое окружений tabular или array может быть повернуто. Этой возможностью можно воспользоваться для создания (повернутых) скобок, охватывающих несколько столбцов или строк.

В следующих примерах строительным блоком является макро \Bpara, определение которого приведено ниже. Идея состоит в том, чтобы создать скобку (}) определенной высоты (аргумент 4), повернуть ее настолько, насколько необходимо (аргумент 3), но LaTeX'у сообщить, что для набора полученного результата отводить место не требуется. Вот для чего применяется окружение picture с нулевыми высотой и шириной. Аргументы 1 и 2 используются в

команде \put, предоставляя нам средства для тонкой настройки положения скобки путем горизонтального или вертикального смещения ее контрольной точки (середины) на некоторое число пунктов.

Как мы видели в предыдущих примерах, при повороте материал сдвигается вверх или вниз в зависимости от угла поворота и размеров поворачиваемого материала. Во избежание этого побочного эффекта сделаем так, чтобы скобка не занимала никакого места ни в каком направлении, применив команды \raisebox и \makebox и задав нулевые размеры в качестве всех аргументов. Реальная скобка создается в математическом режиме с помощью команд \left и \right, окружающих невидимую линейку \rule, размер которой контролирует высоту скобки. Похожие команды могут быть построены и для других расширяемых символов.

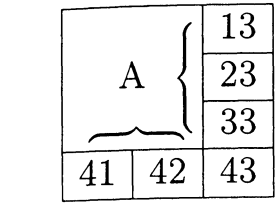
```
% #1,#2 x и y координаты контрольной точки скобки в pt
% #3 угол поворота большой скобки в градусах
% #4 суммарная высота скобки в pt
\newcommand\Bpara[4]{%
 \begin{picture}(0,0)%
  \setlength{\unitlength}{1pt}%
  \put(#1,#2){\rotatebox{#3}{\raisebox{0mm}[0mm][0mm]{%
            \makebox[0mm]{$\left.\rule{0mm}{#4pt}\right\}$}}}}%
 \end{picture}}
```

Применение приведенного выше определения означает необходимость задания значений четырех аргументов вручную, но эта задача является относительно простой: при первом запуске вы указываете некоторые приближенные значения и, так как данные в вашей таблице фиксированы, вы измеряете ее линейкой и корректируете значения аргументов. Поскольку результат выполнения команды не занимает места, он не меняет положения материала таблицы.

$$
\begin{array}{ccccc}
a_{11} & & & & \\
a_{21} & a_{22} & & & \\
a_{31} & a_{32} & a_{33} & & \\
a_{41} & a_{42} & a_{43} & a_{44} & \\
a_{51} & a_{52} & a_{53} & a_{54} & a_{55}
\end{array}
\Bpara{0}{7}{61}{65}
\Bpara{-8}{-6}{-90}{58}
$$

```
\usepackage {graphics}
\[
\begin{array}{ccccc}
 a_{11}                                  \\
 a_{21} & a_{22}                         \\
 a_{31} & a_{32} &
  a_{33}\Bpara{0}{7}{61}{65}             \\
 a_{41} & a_{42} &
  a_{43}      & a_{44}                   \\
 a_{51} & a_{52} &
  a_{53}\Bpara{-8}{-6}{-90}{58}
              & a_{54} & a_{55}
   \end{array}
\]
```

2-3-16



2-3-17

```
\usepackage {graphics}
\Large
\begin{tabular}{|c|c|c|}     \hline
\multicolumn{2}{|c|}{}& 13 \\\cline{3-3}
\multicolumn{2}{|c|}{
        A\Bpara{15}{7}{180}{24}}
                        & 23 \\\cline{3-3}
\multicolumn{2}{|c|}{
 \Bpara{5}{0}{90}{22}}& 33 \\\hline
41      & 42            & 43 \\\hline
\end{tabular}
```

2.4 Другие пакеты, входящие в набор стандартных графических средств

2.4.1 Пакеты epsfig и rotating

Написанные Себастианом Ратцем стилевые файлы epsfig и rotating для LaTeX 2.09 (см. разд. 10.3 и 10.4 в книге Goossens et al. (1994)) были переписаны в виде интерфейсов для пакета graphicx, чтобы пользователи могли продолжать пользоваться привычным синтаксисом. Для создания новых документов рекомендуется применять новую систему графических команд LaTeX'a, описанную в данной главе. Однако следует отметить, что пакет rotating предоставляет некоторые дополнительные средства высокого уровня, такие как окружения `sidewaystable` и `sidewaysfigure`, которые отсутствуют в стандартных графических пакетах. Помимо этого пакет rotating позволяет пользователю контролировать направление поворота с помощью необязательных параметров `clockwise` (используемому по умолчанию) и `counterclockwise`.

2.4.2 Пакет lscape

Созданный Дэвидом Карлайлом пакет lscape, первоначально написанный для LaTeX'a 2.09, также был переписан так, чтобы использовать стандартные графические возможности. В нем используются те же параметры, что и в пакете graphics. В пакете определяется окружение `landscape`, с помощью которого содержимое страницы поворачивается на 90°. При этом верхние и нижние колонтитулы остаются без изменений: они появляются в стандартном (портретном) положении.

2.4.3 Пакет trig

Пакет trig не предназначен для непосредственного использования в документах. С его помощью вычисляются значения тригонометрических функций синус, косинус и тангенс, которые требуются для подсчета размеров пространства, занимаемого повернутым боксом. Этот пакет используется в пакете fontinst Алана Джеффри, который преобразует метрические файлы PostScript'овских шрифтов в форму, понимаемую TeX'ом (см. с. 457).

2.4.4 Пакет keyval

Этот пакет тоже предназначен для использования в других пакетах и полное описание его работы выходит за рамки данной главы. В нем реализован общий метод задания «параметров», используемых пакетом graphicx, и разбиения на части разделенных запятой списков, состоящих из пар вида *key=value*.

Работа с METAFONT'ом и METAPOST'ом

В процессе создания системы типографского набора TEX Дональд Кнут быстро понял, что дополнительно потребуется отдельная программа для разработки шрифтов. Он создал METAFONT — язык для описания форм знаков и программу для интерпретации этого языка и превращения описаний форм в наборы точек, предназначенных для вывода на печать или просмотра. Результатами работы Дональда Кнута являются TEX, METAFONT и обширное семейство шрифтов Computer Modern, запрограммированных на языке METAFONT. METAFONT также был использован для разработки специализированных символьных шрифтов и некоторых других семейств шрифтов.

Разработка METAFONT'а как языка описания шрифтов до некоторой степени дублирует язык PostScript, который тоже очень элегантно описывает форму знаков. Однако если стратегия PostScript'а состоит в том, чтобы отложить воспроизведение формы знаков до финальной стадии вывода на печать, то стратегия METAFONT'а — в том, чтобы заранее получить результат в растровом виде и печатать его на любом простом печатающем устройстве.

Разработка шрифтов несомненно является искусством, для занятий которым большинство из нас плохо подготовлено. Но METAFONT является очень мощной системой, позволяющей справиться с большинством графических задач. А теперь появилась и программа METAPOST, которую можно считать родной сестрой METAFONT'а и которая использует по существу тот же язык, но вместо растрового результата выдает результат на языке PostScript. В этой главе будут рассмотрены вопросы применения METAFONT'а и METAPOST'а (поэтому мы будем использовать термин META для обозначения «как METAFONT, так и METAPOST») для рисования графических образов и форм, отличных от символов в шрифтах. Язык META необыкновенно хорошо подходит для изображений, графически описывающих некоторый математический объект (такой, как кривые Гильберта или Серпинского). Это не удивительно, если учесть, что целью Дональда Кнута было математическое описание формы знаков шрифта.

Мы начнем с короткого обзора языка META. Нашей главной целью будет дать читателям, незнакомым с этим языком, некоторое представление о его возможностях и качестве изображений, которые могут быть созданы с его помощью. Мы постараемся давать описания команд по мере их появления, но некоторые примеры могут содержать тексты программ на языке META, которые явно не объясняются.

Затем мы обсудим, как запускать программы METAFONT и METAPOST, и более или менее подробно рассмотрим два пакета высокого уровня, предназначенных для того, чтобы обуздать мощь METAPOST'а, предоставив средства для рисования боксов и диаграмм. После этого мы рассмотрим программы, создающие для вас команды на языке META, сосредоточившись, главным образом, на (LA)TEXовском пакете mfpic. В заключение, мы кратко опишем, как заставить METAFONT создавать поддающиеся редактированию файлы в формате инкапсулированного PostScript'а.

3.1 Язык META

Все хитрости METAFONT'а весьма подробно описаны в Knuth (1986c). В руководстве по METAPOST'у (Hobby, 1992) не только изложены различия между этими двумя системами, но оно и само по себе служит хорошим введением в язык META. Книга Алана Хёнига *TEX Unbound* (1997) содержит массу полезного материала о приемах METAFONT'а. Статьи, публикуемые на протяжении многих лет в журнале *TUGboat*, тоже служат важным источником информации для тех, кто хочет глубже погрузиться в METAFONT.

Язык META позволяет вам задавать осуществление заливки, изменения масштаба, поворотов, отражений, растяжений и сдвигов объектов, а также других сложных преобразований. В каком-то смысле, программы META можно считать специализированными системами для решения уравнений, побочным продуктом которых оказывается создание изображений.

Эти программы поддерживают множество сложных типов данных: `boolean`, `numeric`, `pair`, `path`, `pen`, `picture`, `string` и `transform`. Давайте рассмотрим некоторые из них подробнее:

pair На языке META «точки» в двумерном пространстве представляются с помощью типа `pair`. Константы типа `pair` имеют вид (x, y), где обе величины x и y являются константами типа `numeric`. Переменная p типа `pair` эквивалентна парному выражению (`xpart` p, `ypart` p).

path Тип `path` есть непрерывная кривая, составленная из цепочки *сегментов*. Форма каждого сегмента определена четырьмя *контрольными точками*. Две из них, *ключевые* точки, являются конечными точками сегмента. Отыскание двух других очень часто предоставляется самому META.

pen Перья, отличительная особенность языка META, представляют собой заполненные выпуклые фигуры, двигающиеся вдоль траекторий `path` и влияющие на то, как в конечном итоге будут нарисованы линии. В META

изначально заданы два пера: `nullpen` и `pencircle`. Перо `nullpen` — это просто точка $(0,0)$. Оно не содержит пикселов и может использоваться для заполнения некоторой области, не меняя ее границы. В противоположность ему, перо `pencircle` является круглым, причем точки $(\pm 0.5, 0)$ и $(0, \pm 0.5)$ лежат на его границе. Другие перья конструируются в виде выпуклых многоугольников с помощью команды `makepen` c, где c — это замкнутая траектория `path`. Ключевые точки траектории c становятся вершинами этого пера. Сами перья можно трансформировать.

picture Тип данных `picture` может использоваться для сохранения последовательности команд рисования языка METAFONT. Результат целой программы на языке METAFONT часто получается благодаря взаимодействию нескольких изображений. Например, в METAFONT'е значением выражения $v + w$ служит изображение, каждый пиксел которого равен сумме двух пикселов, занимающих то же самое положение, соответственно, на изображениях v и w.

transform Аффинные преобразования, т. е. линейные преобразования, дополненные сдвигами, являются естественными преобразованиями евклидовой геометрии. Язык METAFONT позволяет построить любое аффинное преобразование и предоставляет семь примитивных преобразований (Knuth, 1986c, p. 141): *shifted*, *scaled*, *xscaled*, *yscaled*, *slanted*, *rotated* и *zscaled*. Результат большинства этих преобразований очевиден. Последнее преобразование *zscaled* использует пару чисел, интерпретируемых как комплексное число в декартовых координатах, т. е. представляет собой комплексное умножение.

Поскольку значительная часть работы при написании программ на METAFONT заключается в описании геометрических форм, в языке реализован богатый набор вычислительных операций, например, непосредственно поддерживаются операции пифагорова сложения (`++`) и вычитания (`+-+`). В список полезных вычислительных функций входят `length` x (абсолютная величина x), `sqrt` x (квадратный корень из x), `sind` x (синус угла x градусов), `cosd` x (косинус угла x градусов), `angle` (x, y) (арктангенс величины y/x), `floor` x (наибольшее целое число $\leq x$), `uniformdeviate` x (равномерно распределенная случайная величина, лежащая между 0 и x) и `normaldeviate` (нормально распределенная случайная величина со средним 0 и стандартным отклонением 1).

Контроль за ходом выполнения программы осуществляется с помощью конструкции `for ...endfor` с обычными условными переходами. Вы можете писать макро с параметрами или подпрограммы, есть возможность использовать локальные переменные и группирование для того, чтобы ограничить область видимости изменений значений.

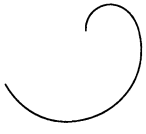
Наконец, язык METAFONT известен своей способностью решать линейные уравнения, включая уравнения, содержащие координаты точек. Например, вы можете задать точку с помощью других точек; так, выражение `z3=1/2[z1,z2]` задает `z3` как точку, расположенную в середине линии, соединяющей `z1` и `z2`.

3.1.1 Примеры программ на языке META

Ниже приведены несколько примеров текстов на языке META и полученные с их помощью изображения, нарисованные программой METAPOST. Вам не составит большого труда таким же образом обработать их программой METAFONT за исключением того, что вы столкнетесь с проблемой при использовании выходных устройств с высоким разрешением, так как при создании больших изображений METAFONT'у не будет хватать памяти. Английский оригинал книги был набран с разрешением 2400 dpi и некоторые METAFONT'овские примеры было невозможно создать с таким высоким разрешением. Выходом из ситуации является либо работа с более низким разрешением (например, 300 dpi), либо разбиение изображения на отдельные «символы» некоторого шрифта и соединение их вместе средствами LaTeX'а. Если вы располагаете PostScript'овским принтером, вам почти несомненно лучше использовать METAPOST.

Здесь мы не приводим тексты программ-оболочек, которые запускают эти примеры на исполнение. Об этом говорится в разд. 3.2.1 на с. 81 при описании того, как METAFONT создает символ, и в разд. 3.2.2 на с. 85 при описании того, как METAPOST создает рисунок.

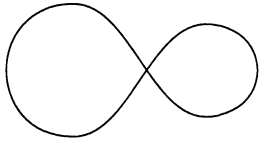
Простейшей инструкцией META является конструкция `draw`, которая получает последовательность точек, разделенных `..`, и соединяет их кривыми:



```
draw (0,0)..(50,20)..
 (40,30)..(30,20)
```

3-1-1

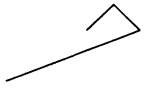
Принятой по умолчанию единицей измерения здесь и далее во всех примерах служит PostScript'овский пункт (1/72 дюйма, TeX'овский «большой пункт»). Для того чтобы гладко соединить первую и последнюю точки, последовательность следует закончить словом `cycle`:



```
draw (0, 50)..(0,0)..
  (60,40)..(60,10)..cycle;
```

3-1-2

Прямые линии рисуются, если между точками указать `--` вместо `..` (в действительности прямые реализованы как кривые с наложенными специальными ограничениями):



```
draw (0,0)--(50,20)--
        (40,30)--(30,20)
```

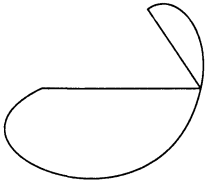
3-1-3

Существует несколько способов управления рисованием кривой: можно менять угол наклона в точках начала и конца кривой с помощью команды `direction` (сокращенно `dir`), менять точки, являющиеся экстремальными (крайняя верхняя, крайняя левая и т. д.), и менять изгиб кривой (с помощью команд `tension` и `curl`). Следующая программа рисует грубую спираль путем продуманного применения инструкции `dir`:

3-1-4

```
u=2.5cm;
path p;
p= (0,0) {dir 130}..
   {dir -130}(0.25u,0){dir 130}..
   {dir -130}(0.5u,0){dir 130}..
   {dir -130}(0.75u,0){dir 130}..
   {dir -130}(u,0);
draw p rotated -90;
```

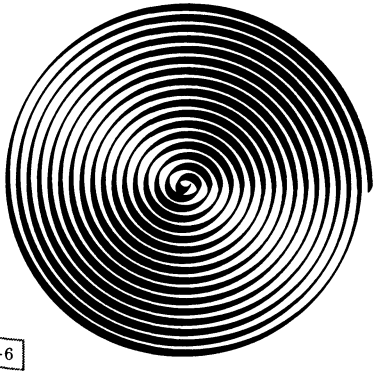
Следующая программа демонстрирует эффект применения команды `curl`: сначала три точки соединяются прямыми линиями, а затем эти же точки соединяются кривыми с присвоенными им значениями `curl`:



3-1-5

```
path p,q;
u=.35cm;
q=(0u,0u)--(6u,0u)--(4u,3u);
draw q;
p=(0u,0u){curl 4000}..(6u,0u)
  ..{curl 4000}(4u,3u);
draw p;
```

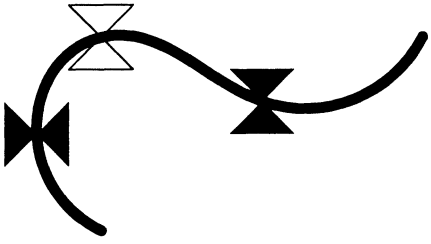
Чтобы продемонстрировать используемые в языке METAFONT необычные «перья», мы аппроксимировали спираль, нарисованную с помощью нестандартного «пера». Вариант этого рисунка, выполненный в цвете, приведен в цветной вклейке I(a).



3-1-6

```
pickup pencircle scaled 3.2pt
   yscaled .2pt rotated 60;
n:=4;
for i := (n*17) step -(n) until (n):
 draw (i,0)..(0,i)..(-i,0)
      ..(0,-(i-n))..(i-n,0);
endfor
```

Типичным для META приемом являются создание траектории и ее последующее многократное использование с различными преобразованиями. В следующем примере представлен фрагмент рисунка хвоста бумажного змея:

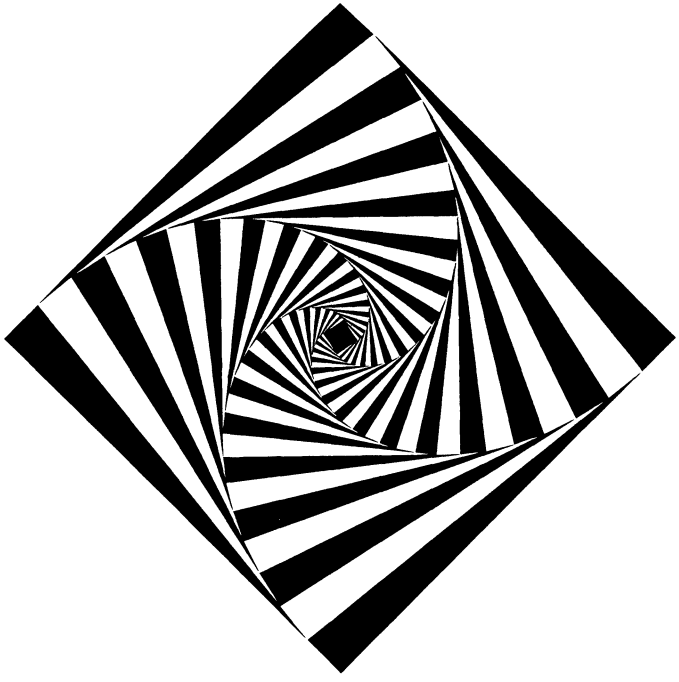


```
u=.85cm;
path p[];
p1:=(.5u,.5u)--(1.5u,.5u)--(.5u,1.5u)
 --(1.5u,1.5u)--(.5u,.5u)--cycle;
fill (p1 shifted (0,2.5u))
 rotatedaround ((u,3.5u),90);
draw p1 shifted (u,4u);
fill p1 shifted (3.5u,3u);
p2 =(2u,2u)..(u,3.5u)..(2u,5u)
 ..(4.5u,4u)..(7u,5u);
pickup pencircle scaled 4pt;
draw p2;
```

3-1-7

Более сложная картина, созданная Аланом Хёнигом для своей книги *TEX Unbound* (1997), демонстрирует работу команд управления циклами. Боксы постепенно уменьшающегося размера поочередно закрашены белым и черным и каждый немного повернут относительно предыдущего.

```
boolean timetofillbox; timetofillbox := true;
partway := 0.9;
l := .75in; u := 1.75in; n := 4;
theta := 360/n;
z1 = (0,u);
for i := 2 upto n: z[i] = z1 rotated ((i-1)*theta); endfor
forever:
 path p;
 p := z1
 for j := 2 upto n: --z[j] endfor  --cycle;
 if timetofillbox:
  fill p;
  timetofillbox := false;
 else:
  unfill p;
  timetofillbox := true;
 fi
 pair Z[];
 for j := 1 upto n:  Z[j] := partway[z[j-1],z[j]];  endfor
 Z1 := partway[z[n],z1];
 for j := 1 upto n:  x[j] := xpart Z[j]; y[j] :=
ypart Z[j];  endfor
 if not timetofillbox: l := abs(z1); fi
 exitif l < .05u;
endfor
```



3-1-8

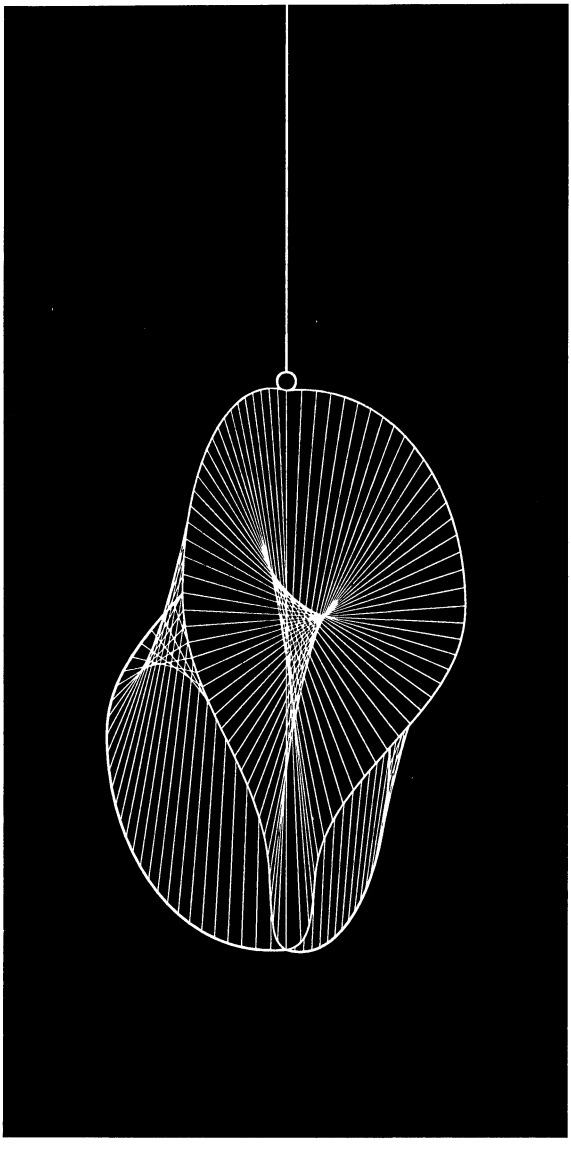
Наконец, приятным побочным эффектом способности языка METAFONT рисовать кривые линии оказалась возможность использования его в качестве отличного инструмента художниками. Кес ван дер Лан демонстрирует[1] на рис. 3.1, смоделированном в стиле конструктивистского направления Наума Габо, что METAPOST с одной стороны подходит для трехмерной графики, а с другой — для графического искусства.

3.2 Применение программ METAFONT

3.2.1 Запуск METAFONT'а

До некоторой степени программа METAFONT похожа на TEX. Она обладает языком низкого уровня, с помощью которого пользователь может писать макро. Большинство пользователей предпочитают начинать не с чистого листа, а загружают предварительно созданную библиотеку полезных команд («базовый» файл), наподобие формата «plain», написанного Дональдом Кнутом. Когда в системе установлен «METAFONT», обычно можно считать, что установлена и библиотека «plain».

[1] Текст программы для этого рисунка, а также для других примеров из этой книги можно найти на CTAN.



3-1-9

Рис. 3.1. Созданная с помощью META картина Кеса ван дер Лана в стиле Наума Габо.

В METAFONT'овском файле программа, создающая каждый символ или объект, заключается между командами начала и конца группы `beginchar...endchar`. Команда `beginchar` имеет четыре параметра: имя создаваемого символа, его ширину, высоту и глубину. Следующая программа,

состоящая из очень простой строки, создает «букву» в позиции, соответствующей букве «X» в данном шрифте:

```
beginchar ("X", 50, 50, 0);
draw (0,0)..(50,50)..(50,0)
endchar;
end
```

Вы создаете исходное описание шрифта, просто записывая METAFONT'овский файл, содержащий набор определений символов. Теперь обработайте его программой, чтобы получить результат в растровом формате gf (подробнее об этом и других форматах шрифтов см. разд. 10.2.1 на с. 422). Интерактивная сессия обработки METAFONT'ом файла с описанием шрифта может выглядеть примерно так (на машине, работающей под операционной системой Unix):

```
 > mf
This is METAFONT, Version 2.718 (C version 6.1)
**\mode=qms;input logo10
(/cdrom/texmf/fonts/source/public/mflogo/logo10.mf
(/cdrom/texmf/fonts/source/public/mflogo/logo.mf [77] [69] [84] [65]
[70] [80] [83] [79] [78]) )
Font metrics written on logo10.tfm.
Output written on logo10.300gf (9 characters, 1124 bytes).
Transcript written on logo10.log.
```

Строка с приглашением вида «**» говорит нам, что METAFONT ожидает ввода. В этот момент мы с помощью команды mode указываем выходное устройство, для которого предполагается создать растровые шрифты (см. ниже), и просим METAFONT прочесть исходное описание шрифта и создать растровый шрифт. Полученный в результате gf-файл преобразуется в формат pk для того, чтобы им воспользовалось устройство предварительного просмотра или принтер (хотя существует возможность пользоваться непосредственно форматом gf):

```
 > gftopk -v logo10.300gf
This is GFtoPK 2.3 (C version 6.1)
[./logo10.300gf->logo10.300pk]
'METAFONT output 1996.11.08:2035'
1124 bytes packed to 516 bytes.
```

Можно также попросить METAFONT показать на экране конструируемый символ по мере его создания.

Команда *mode* очень важна. Именно с ее помощью вы сообщаете METAFONT'у о вашем принтере или экране. Поскольку METAFONT создает растровые образы, ему необходимо знать, сколько пикселов содержится в дюйме и как они организованы. Например, есть существенная (концептуальная) разница между принтерами, начинающими печатать на белом листе бумаги и наносящими на него черные точки, и принтерами, начинающими печатать на черном

листе и удаляющими с него те области, которые должны быть белыми. Описание режимов для большого числа устройств содержится в METAFONT'овском файле `modes.mf`, обновление которого осуществляется Карлом Берри (этот файл входит в большинство дистрибутивов METAFONT'а и может быть найден на CTAN'е). Типичными именами являются «cx» (используемый во многих системах лазерный принтер Canon с разрешением 300 dpi), «ljfour» (принтер HP LaserJet IV с разрешением 600 dpi) и «linoone» (наборное устройство Linotronic с разрешением 1270 dpi). При настройке на любое устройство происходит присвоение значений большому числу внутренних переменных METAFONT'а, как показано в следующем примере для принтера Laserjet IV:

```
mode_def ljfour =
  mode_param (pixels_per_inch, 600);
  mode_param (blacker, .25);
  mode_param (fillin, 0);
  mode_param (o_correction, 1);
  mode_common_setup_;
enddef;
```

Все это объяснено в Knuth (1986c) на с. 90–93. После задания режима следует ввести команду `mode_setup`, которая осуществляет внутреннюю настройку в соответствии с выбранным режимом.

Величина *dpi* (dots per inch (точек на дюйм)) в описании режима наряду с масштабом изображения используется для присвоения имени выходному файлу. Результатом деятельности METAFONT'а является `gf`-файл (*generic font* (*шрифт общего вида*)), который может быть сжат в эквивалентный `pk` (*packed* (*упакованный*)) формат с помощью вспомогательной программы `gftopk`. Если пользователи не указывают никакого режима или указывают неизвестный режим, то файлы будут иметь расширение вида `.2602gf`. В этом случае METAFONT переходит в режим «`proof`», используемый создателями шрифтов для просмотра шрифтов в увеличенном виде с разрешением 2601.72 dpi — 36 пикселов на пункт (откуда файл и получил расширение 2602).

Если запустить METAFONT еще раз, на этот раз в режиме, соответствующем наборному устройству Linotronic со средним разрешением, то выходной файл будет иметь другое имя, а `.gf`-файл окажется в три раза больше:

```
 > mf
This is METAFONT, Version 2.718 (C version 6.1)
**\mode=linoone;input logo10
(/cdrom/texmf/fonts/source/public/mflogo/logo10.mf
(/cdrom/texmf/fonts/source/public/mflogo/logo.mf [77] [69] [84] [65]
[70] [80] [83] [79] [78]) )
Font metrics written on logo10.tfm.
Output written on logo10.1270gf (9 characters, 3900 bytes).
Transcript written on logo10.log.
 > gftopk -v logo.1270gf
This is GFtoPK 2.3 (C version 6.1)
```

```
[./logo10.1270gf->logo10.1270pk]
'METAFONT output 1996.11.08:2036'
3900 bytes packed to 1568 bytes.
```

Помимо gf-файла METAFONT обычно создает метрический файл с расширением tfm (см. разд. 10.2.1 на с. 422). Метрический файл всегда должен получаться одинаковым вне зависимости от текущего режима или выбранного увеличения. TEX умеет масштабировать информацию, содержащуюся в tfm-файлах, но символы в растровых файлах не могут масштабироваться. Если вам нужен символ большего размера, вам придется вновь запустить METAFONT, чтобы создать растровое изображение нужного размера и с нужным разрешением.

Поскольку в TEX'е размеры шрифтов возрастают в геометрической прогрессии (так называемые «magsteps» (шаги увеличения), которые растут с шагом 1.2), вы можете сообщить величины этих шагов METAFONT'у и создать символы большего размера, например,

```
 > mf
This is METAFONT, Version 2.718 (C version 6.1)
**\mode=cx; mag=magstep(1); input logo10;
(/cdrom/texmf/fonts/source/public/mflogo/logo10.mf
(/cdrom/texmf/fonts/source/public/mflogo/logo.mf [77] [69] [84] [65]
[70] [80] [83] [79] [78]) )
Font metrics written on logo10.tfm.
Output written on logo10.360gf (9 characters, 1280 bytes).
Transcript written on logo10.log.
```

Команда mag=magstep(1) означает, что при взятии за основу отвечающего режиму значения (режиму CX соответствует 300 dpi) создается шрифт с разрешением 1.2 × 300 dpi, т.е. .360gf.

Между шрифтом, содержащим символы высотой 10 пунктов (например, cmr10), обработанным METAFONT'ом с увеличением 1.2 с тем, чтобы были созданы символы высотой 12 пунктов, и шрифтом, содержащим символы высотой 12 пунктов (например, cmr12), обработанным METAFONT'ом без увеличения, существуют различия. В первом случае просто увеличивается масштаб символов, разработанных для размера 10 пунктов, тогда как во втором случае используются символы, специально разработанные для большего размера. Однако в случае, когда речь идет о создании картинок, а не о создании шрифтов, подобное различие делается редко. Изображения обычно сразу создаются «правильного» размера путем указания METAFONT'у нужного масштаба.

3.2.2 Альтернатива METAFONT'у — METAPOST

Созданный Джоном Хобби на основе исходных текстов для METAFONT'а METAPOST понимает язык, почти идентичный языку METAFONT'а, но выдает PostScript'овские файлы. Полученные в результате файлы можно включать в

LaTeX'овский документ с помощью стандартного графического пакета. METAPOST создан не для производства шрифтов, а для рисования изображений и графиков общего вида, поэтому имеет несколько важных отличий от METAFONT'а:

1. Поскольку на выходе появляется не растровое изображение, а программа на PostScript'е, он аппаратно *независим* и вам не надо заботиться о *режимах*.
2. Программа является программой общего назначения, поэтому позволяет вставлять обычный LaTeX'овский текст в рисунок в качестве меток, подписей и т. д.
3. Добавлена поддержка цвета.

Более подробно различия между языками METAFONT и METAPOST перечислены в руководстве по METAPOST'у (Hobby, 1992). Почти все осуществляющие рисование команды, которые работают в METAFONT'е, также работают и в METAPOST'е, но последний содержит дополнительные команды. Одна особенно полезная METAPOST'овская макробиблиотека, облегчающая рисование и соединение боксов, описана в разд. 3.3.1 на с. 92.

К сожалению, METAPOST не позволяет создать высококачественные PostScript'овские шрифты в формате Type 1 из исходных программ на METAFONT'е, хотя он может быть использован для создания шрифтов в менее сложном формате Type 3.

В отличие от программы на языке METAFONT, которая обычно состоит из пар команд `beginchar` ... `endchar`, программа на языке METAPOST обычно состоит из пар `beginfig` и `endfig` и командного слова `end`, следующего за последней парой. Команда `beginfig` имеет параметр, задающий расширение выходного файла, соответствующего каждому изображению. Размер изображения указывать не требуется: METAPOST сам определит его и вставит в выходной файл корректную командную строку `%%BoundingBox`. Принятой по умолчанию единицей измерения являются PostScript'овские пункты (которые в TeX'е называются «большими пунктами», в одном дюйме их 72, а не 72.27), хотя, конечно, длины можно задавать и явным образом. Следующая очень простая METAPOST'овская программа рисует квадрат со стороной один дюйм:

```
beginfig(1);
draw (0,0)--(0,72)--(72,72)--(72,0)--cycle;
endfig;
end
```

Если ее назвать `test.mp` и обработать METAPOST'ом, то будет получен PostScript'овский файл `test.1`, имеющий следующий вид:

```
%!PS
%%BoundingBox: -1 -1 73 73
%%Creator: MetaPost
%%CreationDate: 1996.11.08:2155
%%Pages: 1
%%EndProlog
```

```
%%Page: 1 1
 0 0.5 dtransform truncate idtransform setlinewidth pop [] 0 setdash
 1 setlinejoin 10 setmiterlimit
newpath 0 0 moveto
0 72 lineto
72 72 lineto
72 0 lineto
 closepath stroke
showpage
%%EOF
```

Это — обычный файл в формате EPS[1], который можно включить в ваш LaTeX'овский документ с помощью пакета graphics или любого другого приложения. Однако, если вы используете текстовые метки, вся процедура может стать более сложной (см. следующий раздел).

Цвет в METAPOST'е представляется в виде тройки значений, явно определяющих его в терминах красного, зеленого и синего, или же через одну из констант `black`, `white`, `red`, `green`, `blue`. Тройка `color` состоит из трех действительных чисел (которые лежат между 0 и 1 и соответствуют красному, зеленому и синему цветам) и может использоваться в выражениях типа `pair`. Это делает возможным задание цвета в виде `.3blue` или в виде арифметического выражения `(.4,.2,.5) - (.9,.7,.3)`. Если значение цветовой компоненты становится меньше 0 или больше 1, то оно автоматически преобразуется так, чтобы попасть в допустимый диапазон.

Добавление текста

Одно из самых больших преимуществ METAPOST'а над METAFONT'ом состоит в том, что вы можете снабжать иллюстрации текстовыми комментариями. METAPOST может набирать текст, используя стандартные PostScript'овские шрифты, или вы можете указать ему использовать TeX'овские шрифты. Подобные манипуляции с текстом могут быть весьма нетривиальными, поэтому обсуждение этой темы в деталях мы отложим до следующего раздела.

Наиболее распространенный способ добавления текста заключается в том, чтобы воспользоваться командой `label` следующим образом:

`label`.*suffix* (*string expression*, *pair expression*)

Выражение *string expression* печатается в позиции, координаты которой задаются выражением *pair expression*. Выражение *suffix* является необязательным и может принимать одно из следующих значений: `lft` (*left*), `rt` (*right*), `top` (*top*), `bot` (*bottom*), `ulft` (*upper left*), `urt` (*upper right*), `llft` (*lower left*), `lrt` (*lower right*). Оно определяет положение метки по отношению к координате *pair expression*. Расстояние от этой точки до метки устанавливается с помощью переменной `labeloffset`.

[1] Обратите внимание, что обрамляющий бокс больше, чем вы могли бы ожидать, за счет того, что изображающая бокс линия имеет ширину.

Выражение *string expression* может иметь одну из двух форм:

1. Если это выражение имеет тип «строка» языка META (т. е. в простейшем случае ограничено двойными кавычками), то METAPOST набирает текст непосредственно в PostScript'е, используя шрифт, указанный в переменной `defaultfont` (обычно это `cmr10`). Для этого шрифта должен существовать файл с TEX'овской метрикой. Обратите внимание, что при этом лигатуры и керны из `.tfm`-файлов *не* используются.
2. В противном случае, если вы помещаете текст (на этот раз не заключая его в кавычки) между командами `btex ... etex`, он передается на обработку TEX'у. Это позволяет вам пользоваться любыми командами TEX'а или LATEX'а, работать в математическом режиме и т. д. [1]

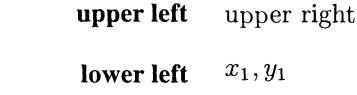
В любом случае, размер шрифта равен размеру базового шрифта (обычно 10pt), умноженному на величину `defaultscale` (значение которой по умолчанию равно 1).

Специальная форма команды `label` вместе с меткой печатает в позиции с указанными координатами точку:

`dotlabel.`*suffix* (*string expression*, *pair expression*)

Эти команды демонстрируются в следующем примере:

```
u=1.75pt;
labeloffset:=10pt;
defaultfont:="ptmb8r";
dotlabel.ulft ("upper left",(100u,100u));
label.urt (btex upper right etex,(100u,100u));
label.llft ("lower left",(100u,100u));
label.lrt (btex $x_1,y_1$ etex,(100u,100u));
draw (50u,100u)--(150u,100u);
draw (100u,125u)--(100u,75u);
```



3-2-1

Две из этих меток имеют тип «строка» языка META и набраны шрифтом `ptmb8r` (преобразованным в шрифт Times-Bold — использованные здесь соглашения, касающиеся имен шрифтов, см. в разд. 10.1.2), а две другие были переданы на обработку TEX'у и набраны выбранным по умолчанию шрифтом из семейства Computer Modern.

[1] В том числе использовать кириллические шрифты. — *Прим. ред.*

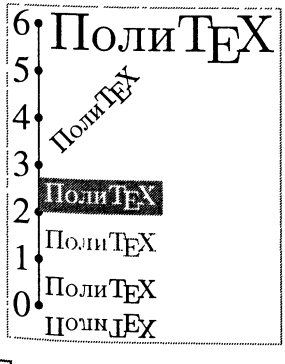
Существует также команда dotlabels, позволяющая присваивать метки целому набору ранее созданных точек:

dotlabels.*suffix* ($pair_1 \ldots pair_n$)

В каждой перечисленной в списке позиции печатается точка; она помечается именем, хранящимся в соответствующей переменной, а сама надпись размещается в соответствии с переменной *suffix*. Эта команда применяется в рассмотренном ниже примере.

Метке, определенной как «строка», соответствуют принятые по умолчанию шрифт и размер. Однако метка также может быть передана на обработку TEX'у в формате btex *выполняющие набор команды* etex. Если используется такая форма, то полученный результат больше не является полностью переносимым, так как зависит от специальных структурированных комментариев, включаемых в выходной PostScript'овский файл, который понимают лишь немногие dvi-драйверы (примером драйвера, обладающего подобными возможностями, служит dvips).

Для того чтобы сохранить метку в «графической» переменной языка METAFONT, вы можете воспользоваться командой thelabel. Она позволяет использовать одну и ту же метку несколько раз при различных преобразованиях. Другой полезный прием состоит в том, чтобы применить к сохраненной метке команду bbox, которая возвращает траекторию в виде прямоугольника, охватывающего картинку METAFONT. В следующем примере этот прием используется для того, чтобы набрать белый текст на зеленом фоне, а также демонстрируются некоторые другие преобразования, которые могут оказаться полезными. Кроме того, в этом примере показано, как применить команду bbox к создаваемому рисунку, чтобы легко окружить его рамкой:



3-2-2

```
defaultscale:=1.4;          u=1pt;
picture p,q;                gap:=18u;
for i := 0 upto 6: z[i]=(0u,i * gap * u);
endfor;
draw z0--z6;
dotlabels.lft(0,1,2,3,4,5,6);
q=thelabel.urt(btex {}Поли\TeX etex ,z0);
draw q;
draw q reflectedabout(z0,z0+(2u,0));
draw q shifted(0,gap) withcolor red;
fill bbox q shifted(0,2*gap*u)
                          withcolor green;
draw q shifted(0,2*gap*u) withcolor white;
draw q scaled 2 shifted(0,5*gap*u);
p=q rotatedabout(z0,45);
draw p shifted(10u,3*gap*u);
draw bbox currentpicture withcolor green;
```

Добавление текста — некоторые трудоемкие подробности

В предыдущем разделе текст передавался на обработку TEX'у с помощью команд `btex ... etex`, но под «TEX'ом» в этом случае понимался не LATEX, а plain TEX. В этом разделе мы рассмотрим этот процесс более подробно и покажем, как использовать LATEX. Кроме того, будет подробно разобрано манипулирование шрифтами.

Когда METAPOST впервые встречает команду `btex`, он вызывает процедуру makempx. Она запускает программу, называющуюся mptotex, которая просматривает `mp`-файл и выделяет все заключенные внутрь `btex... etex` фрагменты во временный файл, окружая их соответствующими макро так, чтобы каждый фрагмент представлял собой отдельную «страницу» в этом TEX'овском файле. После этого она вызывает TEX для обработки собранного материала и передает результирующий `dvi`-файл другой программе (dvitomp), которая преобразует его в `mpx`-формат, воспринимаемый METAPOST'ом. Когда, после завершения работы программы makempx, METAPOST продолжает работу, он вставляет взятое из `mpx`-файла описание изображения, соответствующее каждому заключенному в `btex ... etex` фрагменту. В некоторых вариантах установки METAPOST также есть возможность получать аппаратно-независимый вывод в формате troff путем запуска METAPOST'а с параметром `-T` в командной строке, при котором вызываются другие внутренние процедуры.

Вы можете контролировать этот процесс тремя способами:

1. Любой текст, расположенный между командами `verbatimtex` и `etex`, переносится во внешний файл, но не используется в процессе рисования. Это дает возможность определять макро, задавать LATEX'овский класс документа, осуществлять загрузку пакетов или менять принятые по умолчанию значения переменных.
2. В процессе работы данная процедура ищет переменную окружения, называемую `TEX`, чтобы выяснить, какую именно программу следует вызывать (`tex`, `latex`, `amstex` и т. д.). Вы должны задать значение этой переменной соответствующим образом, если, например, вы уже воспользовались конструкцией `verbatimtex ... etex` для записи в файл команды `\documentclass{article}`.
3. Вы можете задать METAPOST'овскую переменную `prologues`, определяющую способ управления шрифтами. В случае значения, равного 0, получающийся результат зависит от приложения (например, dvips), обрабатывающего обращения к шрифтам TEX'а, которые должны быть загружены и включены в выходной файл. Если значение больше 0, METAPOST старается создать аппаратно-независимый выходной файл на PostScript'е, используя либо troff (при значении, равном 1), либо TEX (любое положительное значение, отличное от 1, например, 2).

В качестве примера, предположим, что в виде METAPOST'овской «метки» вы хотите набрать пару сложных уравнений, требующих $\mathcal{AMS}$-LATEX'овского

математического пакета.[1] На следующем рисунке показано, что надо делать. Когда LaTeX обрабатывает данный фрагмент, он находится в *горизонтальном режиме*, поэтому невозможно непосредственно вывести уравнения. Вместо этого мы помещаем его в окружение `minipage`:

3-2-3

```
verbatimtex
\documentclass{article}
\usepackage{times,amsmath}
\begin{document}
etex;
picture p; path q;
p:=thelabel(btex \large
\begin{minipage}{4in}
\begin{gather}
\iint\limits_A f(x,y)\,dx\,dy\qquad
\iiint\limits_A f(x,y,z)\,dx\,dy\,dz\\
\iiiint\limits_A f(w,x,y,z)\,dw\,dx\,dy\,dz
\qquad\idotsint\limits_A f(x_1,\dots,x_k)
\end{gather}
\end{minipage}
etex,(0,0));
q:=fullcircle scaled 2in;
fill q withcolor blue;
clip p to q;
draw p rotated (90) withcolor white;
```

Этот пример также показывает, как форма одного изображения может быть использована в качестве границы другого (видна только та часть уравнения, которая попадает внутрь круга).

Вам не нужно добавлять `\end{document}`, так как METAPOST делает это вместо вас. Перед запуском METAPOST'а переменной окружения TEX должно быть присвоено значение «latex».

Когда переменная `prologues` установлена равной 0, METAPOST не старается создать чистый код на PostScript'е, но для каждого шрифта вставляет в выходной файл специальные комментарии, содержащие имена шрифтов и необходимые размеры, как в этом примере:

```
%*Font: ptmr8r 9.96265 9.96265 28:c06
%*Font: cmsy10 9.96265 9.96265 01:8
%*Font: cmr7 6.97385 6.97385 31:8
```

Когда рисунок вставляется в LaTeX'овский документ, программе, преобразующей `dvi` в PostScript, требуется прочесть эти команды и найти необходимые шрифты либо в растровом формате, либо в PostScript'овском формате Type 1. В настоящий момент только программа dvips Тома Рокицки (см. разд. 11.2) понимает это соглашение.

[1] Конечно, надо помнить, что номера этих уравнений и их метки *не* видны из основного LaTeX'овского файла, содержащего этот рисунок.

Чтобы заставить METAPOST создавать текст на PostScript'е в таком виде, который распознается любым приложением и может быть отредактирован такими программами, как Adobe Illustrator, установите переменную `prologues` равной 2 (но не 1, что соответствует программе troff). Теперь каждый раз, когда вы используете некоторый шрифт (либо в переменной типа «строка», либо внутри фрагмента на TEX'е), METAPOST ищет его в файле `psfonts.map` программы dvips (см. разд. 11.2.4), чтобы выяснить настоящее имя PostScript'овского шрифта, и соответствующим образом записывает текст на PostScript'е. Он *не* включает шрифт в выходной файл, поэтому если только используемые шрифты не встроены в принтер, вы должны загрузить их вместе с вашим файлом. Если вы пользуетесь шрифтами типа Type 1 из семейства Computer Modern, то должны удостовериться, что они указаны в файле `psfonts.map` и загружены в принтер.

Недостаток такого подхода заключается в том, что, используя PostScript'овские шрифты, вы не можете контролировать кодировку. Например, в описании шрифта «`ptmb8r`» в файле `psfonts.map` указано перекодировать его в нестандартную кодировку. Однако в настоящее время начальные настройки METAPOST'а игнорируют подобные инструкции, поэтому любые символы, не входящие в стандартный диапазон ASCII, с высокой вероятностью будут неверно интерпретированы. Если вы хотите, чтобы результат деятельности METAPOST'а содержал нестандартные символы, мы можем порекомендовать вам использовать при наборе текста имена шрифтов и метрические файлы TEX'а, использующие стандартную кодировку Adobe (Adobe Standard Encoding) (см. разд. 10.3.3). Для набора математики вам следует убедиться, что изменение кодировки не требуется. Здесь может помочь использование написанного Аланом Джеффри LATEX'овского пакета mathptm (см. разд. 10.1.1), хотя в нем используются шрифты из семейства Computer Modern, например, для набора знака квадратного корня. В качестве последнего средства вы можете использовать создание самодостаточных PostScript'овских файлов путем включения рисунка в LATEX'овский документ, не содержащий больше ничего, и превращения его в PostScript (обсуждение этого приема см. в разд. 11.6 на с. 529).

3.3 Макробиблиотеки METAPOST'а

3.3.1 Макро для создания боксов

Одним из побудительных мотивов написания METAPOST'а было желание создать инструменты, сравнимые с теми, которые доступны пользователям программы troff. Одним из лучших подобных инструментов является язык pic (см. разд. 1.4.2 на с. 39 и Kernighan (1984)), часто используемый для рисования и соединения боксов. Написанная Джоном Хобби библиотека boxes макрокоманд на языке METAPOST в сочетании со стандартными средствами METAPOST'а позволяет решать сходные задачи.

Пакет boxes загружается с помощью команды

```
input boxes
```

Ключевая идея, лежащая в основе данного пакета, состоит в том, что рисование состоит из четырех операций:

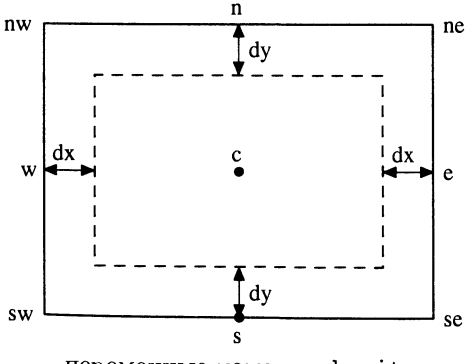
1. Создание именованных боксов (рисунков на языке META).
2. Выражение связей между боксами.
3. Указание системе расположить боксы.
4. Соединение боксов линиями и стрелками.

Это означает, что от вас требуется указывать не точное положение объектов, а только их взаимосвязь с другими объектами.

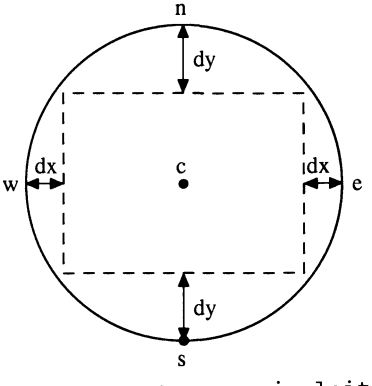
Базовыми командами пакета boxes для создания боксов являются

`boxit.`*suffix*`(`*picture*`)`
`circleit.`*suffix*`(`*picture*`)`

которые рисуют вокруг *picture* прямоугольную или круглую рамку. Дополнительный пакет rboxes содержит еще одну команду `rboxit`, рисующую прямоугольную рамку с закругленными углами. Результату этих команд присваивается имя *suffix* и вам становится доступен набор переменных *suffix*.c, *suffix*.e, *suffix*.n и т. д., которые содержат координаты крайних точек этого объекта:



переменные команды `boxit`



переменные команды `circleit`

Переменные *dx* и *dy* равны зазору между рисунком и окружающим его боксом. Вы можете либо задать их явно, либо позволить METAPOST'у воспользоваться значениями `defaultdx` и `defaultdy`. Переменная `circmargin` (значение которой по умолчанию равно 2 пунктам) определяет в команде `circleit` минимальное расстояние до текста.

Взаимосвязи между боксами определяются как набор выражений, имеющих вид уравнений:

`boxjoin(`*equations*`)`

Этими уравнениями вы описываете взаимосвязи между определенными точками боксов *a* и *b*. Они применяются последовательно к каждому вызову команды `boxit`. Кроме того вы можете явно задать уравнения, описывающие взаимосвязь между любыми боксами.

После того как вы создали серию боксов, вы можете поместить их на страницу с помощью команд:

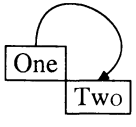
`drawboxed(`box_1`,`box_2`,...`box_n`)` `drawunboxed(`box_1`,`box_2`,...`box_n`)`
`drawboxes(`box_1`,`box_2`,...`box_n`)` `pic(`*box*`)`

которые требуют лишь список объектов, определенных командами `boxit` или `circleit`. Команда `drawboxed` рисует боксы и их содержимое, `drawunboxed` рисует только содержимое, а `drawboxes` — только боксы. Четвертая команда преобразует нарисованный бокс в «рисунок» на языке META, с которым можно обычным образом выполнять операции `draw` или `fill`.

Теперь вы можете соединить боксы типичными для META линиями или кривыми. Два полезных стандартных METAPOST'овских макро рисуют одно- или двусторонние стрелки-указатели:

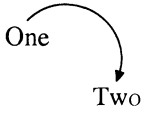
`drawarrow` *path*
`drawdblarrow` *path*

Описания *path* могут содержать специальные координаты, существующие у каждого бокса. Например, `drawarrow one.n--two.s` рисует линию, соединяющую верхнюю сторону бокса «one» и нижнюю сторону бокса «two». Для рисования соединяющих боксы изогнутых линий полезен уточнитель `dir`, что демонстрируется в следующих примерах:




```
boxjoin(a.se=b.nw);
boxit.one("One");
boxit.two("Two");
drawboxed(one,two);
drawarrow one.n{up}..two.n;
```

3-3-1



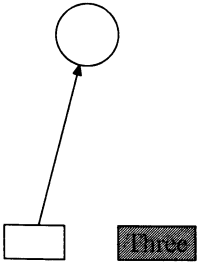

```
boxjoin(a.se=b.nw-(10,-10));
boxit.one("One");
boxit.two("Two");
drawunboxed(one,two);
drawarrow one.n{dir 45}..two.n;
```

3-3-2

Часто возникает желание нарисовать линии, по существу соединяющие центры двух боксов, но обрывающиеся в точках пересечения их границ. Это можно сделать с помощью трех полезных стандартных METAPOST'овских макро:

`bpath` *box name*
`cutafter` *path*
`cutbefore` *path*

Макро `bpath` порождает траекторию в виде прямоугольника, окружающего бокс *box name*. Когда макро `cutafter` и `cutbefore` появляются после команды рисования, они контролируют взаимодействие к заданной траектории *path*. Так, в следующем примере линия, соединяющая центры двух боксов, становится видна в том месте, где она покидает прямоугольник, окружающий первый бокс, и исчезает там, где достигает прямоугольника, окружающего второй бокс.




```
boxjoin(a.n=b.s-(20,60));
boxit.one("One");
circleit.two("Two");
drawboxes(one,two);
boxjoin();
boxit.three("Three");
three.w=one.e+(20,0);
fill bpath three withcolor green;
draw bpath three;
draw pic(three) withcolor red;
drawarrow one.c--two.c
 cutbefore bpath one
 cutafter bpath two;
```

3-3-3

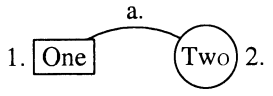
Этот пример также показывает, как указать положение третьего бокса относительно первого абсолютным образом: действующие по умолчанию правила отменяются при помощи пустого уравнения `boxjoin`.

Присвоение боксам меток обычно является легкой задачей. Но какова ситуация с присвоением меток линиям, соединяющим боксы? Для этого можно воспользоваться макро `point`:

`point` *distance* of *path*

который возвращает координату точки, в которую можно попасть, пройдя расстояние *distance* вдоль пути *path*. В сочетании с макро `length`, предна-

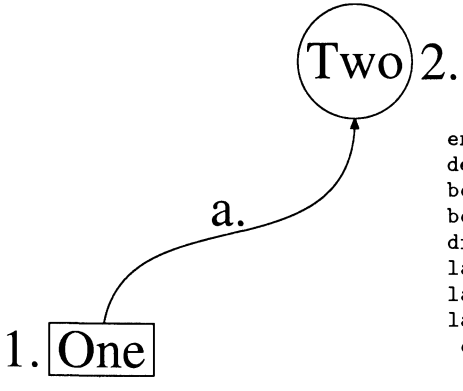
значенным для вычисления длины пути, его можно использовать в качестве параметра команды `label`:




```
boxjoin(a.e=b.w-(30,0));
boxit.one("One");
circleit.two("Two");
drawboxed(one,two);
label.lft("1.",one.w);
label.rt("2.",two.e);
label.top("a.",
 point .5*length (one.c{dir 45}..two.c)
   of (one.c{dir 45}..two.c) );
draw one.c{dir 45}..two.c
 cutbefore bpath one
 cutafter bpath two;
```

3-3-4

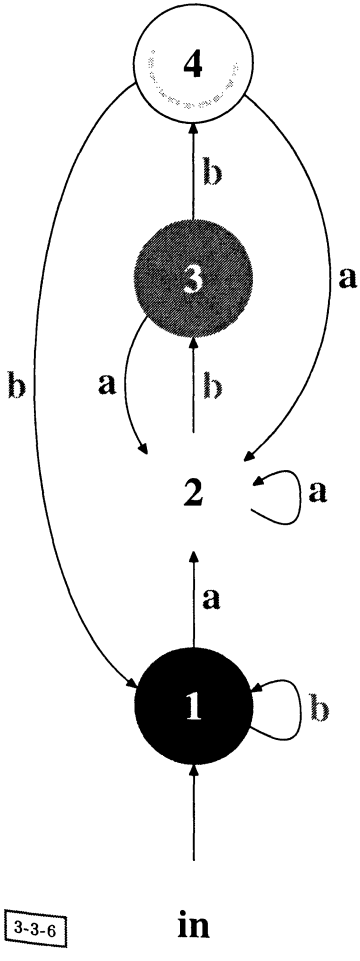
Более элегантная версия этого примера выполняет рисование линии вычисление координат метки с помощью макро `vardef` языка META (взятого из METAPOST'овской документации, посвященной диаграммам). Он содержит два оператора, разделенных ;. Первый рисует соединительную линию, а второй находит координату, возвращаемую в качестве выходного значения всего макро. Рациональное использование команды `tension` и модификации направлений делают внешний вид соединительной кривой немного более интересным:




```
vardef labelarrow
   (suffix BoxA,BoxB) expr Line =
    drawarrow Line
    cutbefore bpath BoxA
    cutafter bpath BoxB;
    point (.5*length Line) of Line
enddef;
defaultscale:=2;
boxjoin(a.e=b.w-(.75in,1.5in));
boxit.one("One"); circleit.two("Two");
drawboxed(one,two);
label.lft("1.",one.w);
label.rt("2.",two.e);
label.top("a.",labelarrow(one,two)
 one.c{dir90}..tension0.8..{dir90}two.s);
```

3-3-5

Этот раздел завершается большим примером, показанным на рис. 3.2 на следующей странице, который демонстрирует, как можно использовать пакет boxes для набора типичных в области информатики диаграмм. Он также приведен раскрашенным (произвольно выбранными цветами) на цветной вклейке I(b), чтобы показать способы использования цвета пакетом boxes. Та же диаграмма, построенная с помощью TEX'овского пакета XY-pic, приведена на с. 220. Тем, кому требуется создавать рисунки такого рода, может быть интересно сравнить тексты программ.

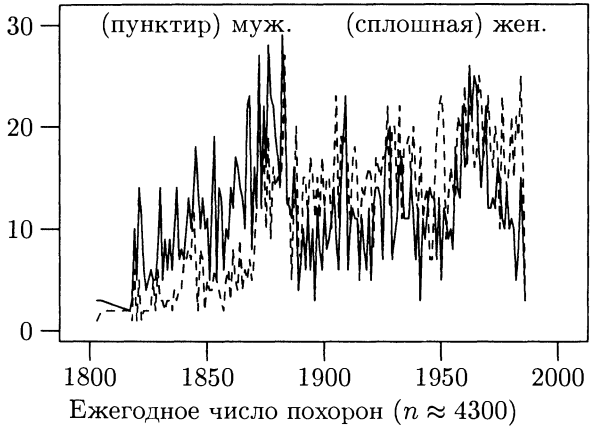



```
defaultfont:="ptmb8r";
vardef labelarrow
 (suffix BoxA,BoxB) expr Line =
  drawarrow Line
  cutbefore bpath BoxA
  cutafter
bpath BoxB;
  point (.5*length Line) of Line
enddef;
color yellow,orange;
yellow:=red+green;  orange:=red+(green/2);
boxjoin(a.n=b.s-(0,.5in));
defaultscale:=1.5; circmargin:=4pt;
circleit.In("in"); circleit.One("1");
circleit.Two("2"); circleit.Three("3");
circleit.Four("4");
boxjoin(); circmargin:=16pt;
circleit.X(""); X.c=Four.c;
drawunboxed(One,Two,Three,Four,In,X);
drawarrow In.n--One.s;
label.rt("a",labelarrow(One,Two) One.c--Two.c)
 withcolor red;
label.rt("b",labelarrow(Two,Three)
 Two.c--Three.c) withcolor green;
label.rt("b",labelarrow(Three,Four)
 Three.c--Four.c) withcolor green;
label.rt("a",labelarrow(Four,Two)
 Four.c{dir335}..{dir205}Two.c)  withcolor red;
label.lft("a",labelarrow(Three,Two)
 Three.c{dir205}..{dir335}Two.c)  withcolor red;
label.lft("b",labelarrow(Four,One)
 Four.c{dir180}..tension2..One.c) withcolor green;
label.rt("b",labelarrow(One,One) One.c{dir45}
 ..One.c+(40,0)..{dir120}One.c) withcolor green;
label.rt("a",labelarrow(Two,Two) Two.c{dir65}
 ..Two.c+(40,0)..{dir100}Two.c)  withcolor red;
fill bpath One withcolor blue;
fill bpath Two withcolor yellow;
fill bpath Three withcolor orange;
draw bpath Four;  draw pic Two;
draw pic One withcolor white;
draw pic Three withcolor white;
pickup pencircle scaled 2pt;
draw bpath X dashed evenly
 withcolor (1 , .75, .8);
```

Рис. 3.2. Диаграмма состояний, нарисованная с помощью программы METAPOST.

3.3.2 METAPOST'овский пакет graph

Автор METAPOST'а Джон Хобби написал на METAPOST'е библиотеку макрокоманд высокого уровня (Hobby, 1993), реализующую сложный пакет для рисования диаграмм, сравнимый с программой grap (см. Bentley and Kernighan 1984). Мы уже видели пример работы программы grap в гл. 1 (рис. 1.22). Мы можем нарисовать похожую диаграмму (на основе данных Протестантского кладбища в Риме, см. Rahtz (1987)) с помощью METAPOST'овского пакета graph (использованные на рисунке штриховые линии являются METAPOST'овским дополнением в язык META):




```
draw
begingraph(2.75in,1.75in);
gdraw "yearm.dat"
    dashed evenly;
gdraw "yearw.dat";
glabel.lft
 (btex(сплошная) жен.etex,
    1999,30);
glabel.rt
 (btex(пунктир) муж.etex,
    1801,30);
glabel.bot
 (btex Ежегодное число
  похорон
  ($n \approx 4300$)%
  etex,OUT);
endgraph;
```

3-3-7

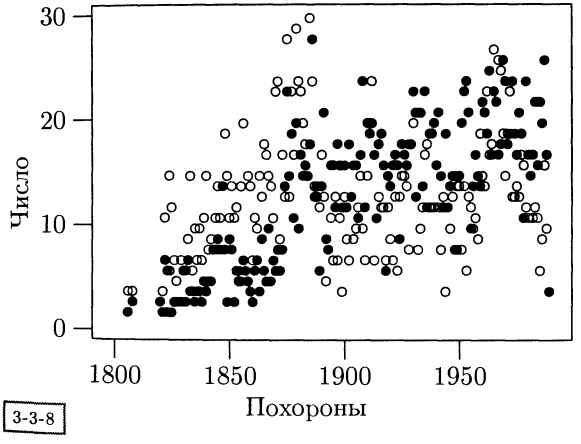
В пакете graph предусмотрены команды для изображения данных, содержащихся во внешних файлах, и добавления меток в том виде, в котором вы хотели бы их увидеть, а также для:

- автоматического изменения масштаба данных;
- автоматического создания и размещения меток рядом с рисками или линиями сетки;
- создания нескольких систем координат на одном рисунке;
- создания линейных и логарифмических шкал;
- изображения информации с помощью произвольных символов;
- обработки нескольких столбцов данных из одного и того же файла процедурами, определенными пользователем.

Вы начинаете с того, что загружаете макропакет graph, после чего каждая диаграмма окружается парой команд `begingraph` ... `endgraph`. В результате вы получаете рисунок, к которому можно применять команду `draw` (и выполнять

другие преобразования наподобие поворота). У команды `begingraph` есть параметры (размеры x и y, разделенные запятой), задающие размер диаграммы. Чтобы соответствовать этим размерам, масштаб данных может быть автоматически изменен с помощью специальных вариантов команд `draw` и `fill`, имеющих префикс «g». Команда `gdraw` обладает таким дополнительным свойством, что когда за ней следует имя файла, команда рисует траекторию, созданную в результате считывания пар координат из файла. За ней могут следовать такие обычные спецификаторы META, как `withpen` или METAPOST'овские `withcolor` и `dashed`. Также существуют «g»-варианты других команд: `glabel`, `gdotlabel`, `gdrawarrow` и `gdrawdblarrow`.

За командой `gdraw` также может следовать команда `plot`, параметром которой является METAPOST'овский «рисунок», который помещается в каждой паре координат. В следующем примере показано, что «рисунок» может быть набран в TEX'е:




```
draw
begingraph(2.5in,1.75in);
gdraw "yearm.dat"
 plot btex $\bullet$ etex;
gdraw "yearw.dat"
 plot btex $\circ$ etex;
glabel.bot
 (btex Похороны etex,OUT);
glabel.lft
 (btex Число etex
     rotated 90,OUT);
endgraph;
```

Обратите внимание на то, что у команды `glabel` параметр позиционирования может принимать специальное значение `OUT`, означающее, что текст должен быть размещен вне диаграммы (обычно оно используется для задания надписей на осях). На рассмотренной диаграмме метка оси y была вдобавок повернута на 90°.

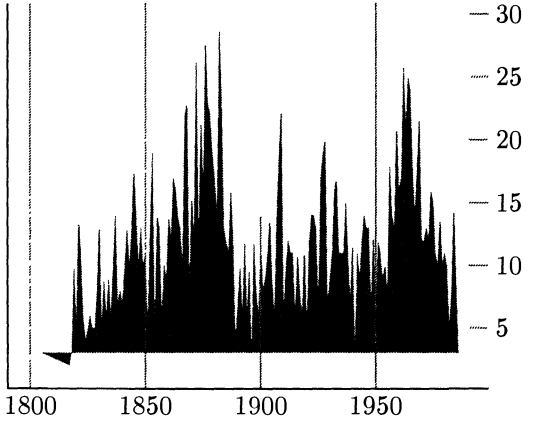
Рамки, шаги, сетки и шкалы

По умолчанию диаграммы окружаются рамкой со всех сторон, изображаются без сетки, а шаг шкалы указывается снизу и слева. Вид рамки задается с помощью команды `frame`, обладающей таким же набором необязательных

суффиксов, что и у команды `label` (см. с. 87). Вид линий сетки и рисок на шкале задается с помощью команды `autogrid`:

`autogrid`(*x specification*,*y specification*)

Спецификаторы «specification» могут принимать значения `grid`, `itick` или `otick`, рисующие линии сетки, риски шкалы внутри и вне диаграммы соответственно. К ним может быть добавлен суффикс `.top` или `.bot`, относящийся к оси x, и `.lft` или `.rt`, относящийся к оси y, как показано в следующем примере:




```
draw begingraph(2.5in,2in);
gfill "yearw.dat"
  withcolor red;
autogrid(grid.bot,itick.rt)
  withcolor .5white;
frame.llft;
endgraph;
```

3-3-9

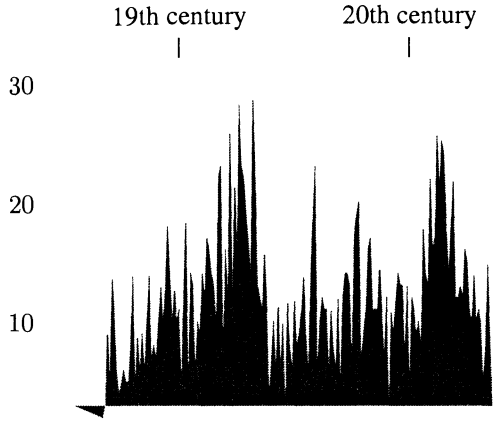
Чтобы изменить настройки пакета `graph`, определяющие, где на шкале располагать риски и как писать метки, вы можете явным образом задать расположение отметок на шкале с помощью команд `itick` или `otick`, а расположение линий сетки — с помощью команды `grid`. Команды могут иметь такие же суффиксы, что и `autogrid`, за которыми следует METAPOST'овская графическая переменная, содержащая метку, или команда `format` и координаты. Команда `format` применяется для задания внешнего вида выводимых чисел:

`format`(*specification*,*number*)

Спецификация *specification* состоит из необязательной начальной строки, знака процента, необязательного числа, указывающего количество цифр после запятой (по умолчанию равного 3), символа преобразования (`e`, `f` или `g`) и необязательной финальной строки. Символ преобразования указывает, используется или нет научная форма записи. Так, в случае буквы `%g` большинство чисел будет представлено в десятичном формате. То,

как будет осуществляться набор для научной формы записи, зависит от METAPOST'овского макро `init_numbers` (см. руководство). Поскольку в нем присутствует конструкция `btex...etex`, вы должны внимательно просмотреть его, если вас интересует, какие именно шрифты будут использоваться.

На следующей диаграмме демонстрируются оба способа явного задания меток. Не забудьте в конце отказаться от применения обычных пометок!




```
draw begingraph(2.5in,2in);
gfill "yearw.dat" withcolor red;
for y=10,20,30:
  itick.lft(format("%g",y),y);
endfor
otick.top("19th century",1850);
otick.top("20th century",1950);
frame.llft;
autogrid(,);
endgraph;
```

3-3-10

Размещением меток также можно управлять с помощью команды `setcoords`:

`setcoords`(*x style,y style*)

Значения параметров для осей x и y могут устанавливаться равными `log`, `-log`, `linear` или `-linear`.

Хотя выбор программой масштаба данных с целью соответствия размерам диаграммы обычно приводит к хорошим результатам, оно может быть изменено с помощью команды `setrange`:

`setrange`(*min,max*)

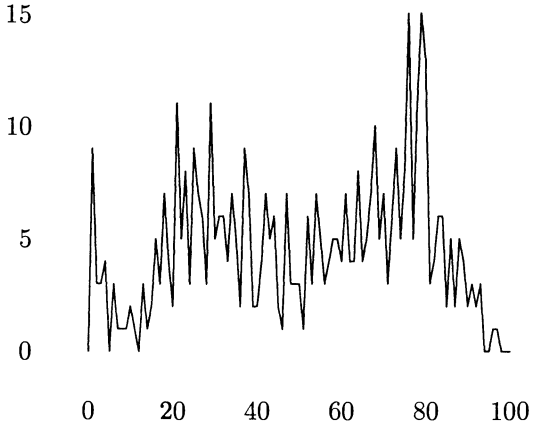
Вы должны указать минимальное и максимальное значения координат. Специальная константа `origin` представляет собой удобную сокращенную форму записи точки (0,0). Чтобы предоставить METAPOST'у самому определить значение любой величины, задайте `whatever`. Если вы не укажете вовсе никакого диапазона, METAPOST найдет его на основе исходных данных и добавит небольшие расстояния по краям.

Считывание файлов с данными

Хотя команд `gdraw` и `gfill` часто бывает достаточно, можно лучше контролировать считывание данных из файла, применяя команду `gdata`:

`gdata`(*filename*, *variable*, *commands*)

Команды *commands* выполняются для каждой строки данных в файле *filename*, причем возможно обращение к значениям данных в каждом столбце, например, *c*1, *c*2 ... *c*n в случае переменной *c*. Параметр *filename* является переменной типа «строка» в языке META, поэтому простые имена следует заключать в кавычки (имена файлов также могут вычисляться на основе переменных языка META). Используя некоторые другие данные Протестантского кладбища, в которых в каждой строке указан возраст умершего, мы можем продемонстрировать распределение смертности в зависимости от возраста путем накопления данных в массиве и последующего использования его для изображения траектории:



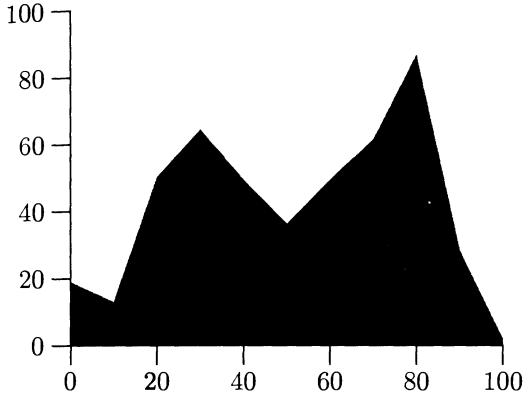

```
draw begingraph(2.5in,2in);
numeric p[]; path r;
for j := 0 upto 100: p[j]:=0;
  endfor
gdata ("ages.dat",y,
  age:=(scantokens y1);
 p[age]:=p[age] + 1;);
r:=(0,0)
 for j := 1 upto 100:
   --(j,p[j]) endfor;
gdraw r;
frame.llft;
endgraph;
```

3-3-11

Единственной сложностью тут являются необходимость инициализации каждого массива и преобразование строки, считываемой из файла, в число с помощью команды `scantokens`.

Когда команда `gdata` считывает файл с данными, она останавливается, достигнув пустой строки или конца файла. Если вы запустите эту же команду снова с тем же самым именем файла, то она продолжит считывание следующей порции данных. Это позволяет вам помещать все данные в один файл, но пользуйтесь этим с осторожностью. Проблема заключается в том, что файлы данных остаются открытыми, если в конце файла находится пустая строка, так как METAPOST считает, что далее есть еще данные. Если у вас много маленьких файлов данных, такая ситуация может вызвать сбой METAPOST'а — следите за тем, чем завершаются ваши файлы.

Внешний вид диаграммы в предыдущем примере выглядит не очень хорошо. Возможно, было бы лучше усреднить взятые из файла данные о смертности за десятилетие. Поскольку эта задача немного сложнее, она решается отдельным METAPOST'овским макро, вызываемым командой `gdata`:



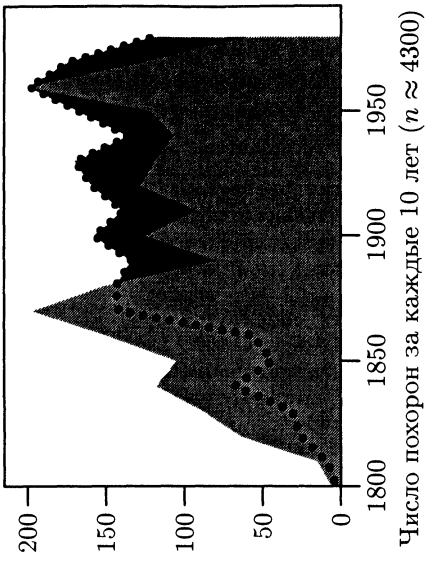

```
draw begingraph(2.25in,1.75in)
setrange(origin,(100,100));
numeric p[]; path r;
for j := 0 step 10 until 100:
  p[j]:=0; endfor
def check(expr age) =
 if age < 100:
   xage:=round(age/10) * 10;
   p[xage]:=p[xage] + 1; fi
enddef;
gdata ("ages.dat",y,
  check((scantokens y1)););
r:=(0,0) for j :=
   0 step 10  until 100:
   --(j,p[j]) endfor --(100,0)
gfill r -- cycle
   withcolor blue;
frame.llft;
endgraph;
```

3-3-12

Часто бывает полезно сначала собрать все точки траектории из каждой строки, считанной из файла данных. Для этой цели предназначен макро `augment`. Получив в виде суффикса имя переменной типа `path` и параметр со значением координаты, этот макро создает эту траекторию, если она не существует, или добавляет к ней данную точку. Он был использован для еще одной демонстрации взятых с надгробных камней дат, на этот раз обработанных таким образом, чтобы получить количество смертей за каждое десятилетие отдельно женщин (столбец 2) и мужчин (столбец 3):

```
1800 3 6
1810 9 15
1820 26 64
1830 31 88
...
```

В каждом десятилетии отслеживается последняя достигнутая точка и ею дорисовываются траектории, соответствующие числу мужских и женских смертей. Затем обе диаграммы раскрашиваются разными цветами (этот рисунок также есть на вклейке I(c)), чтобы продемонстрировать, как мужская и женская смертности изменяются со временем. Нам необходимо знать значение за последнее десятилетие, чтобы осмысленным образом указать угол закрашенной формы. Кривая для женщин изображена точечной линией, наложенной на заштрихованную область для мужчин.



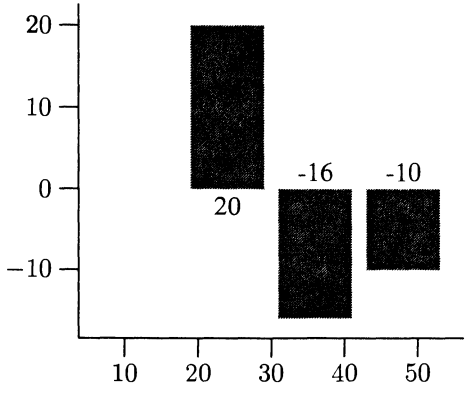

```
path m,w,last;
draw begingraph(2.5in,1.75in);
setrange((1800,0),(whatever,whatever));
gdata ("decade.dat",y,
  last:=((scantokens y1),0);
  augment.w(y1,y2);
  augment.m(y1,y3););
gfill (1800,0)--w--last--cycle
  withcolor red;
gfill (1800,0)--m--last--cycle
  withcolor green;
pickup pencircle scaled 3pt;
gdraw w dashed withdots;
pickup pencircle scaled .75pt;
glabel.bot (btex
  Число похорон за каждые 10 лет
  ($n \approx 4300$)etex,OUT);
endgraph
rotated 90;
```

3-3-13

Этот пример также демонстрирует, что графические макро возвращают рисунок на языке METAFONT, который затем может быть преобразован (в данном случае повернут).

Разные виды диаграмм

Приложив небольшие усилия, можно заставить пакет graph рисовать гистограммы. Для того чтобы это продемонстрировать, мы копируем гистограмму из Goossens et al. (1994), с. 327, которая была построена с помощью LaTeX'овского пакета bar. Наш подход состоит в том, чтобы составить из данных для всех столбцов диаграммы единственную траекторию, а в конце закрасить получившийся результат:

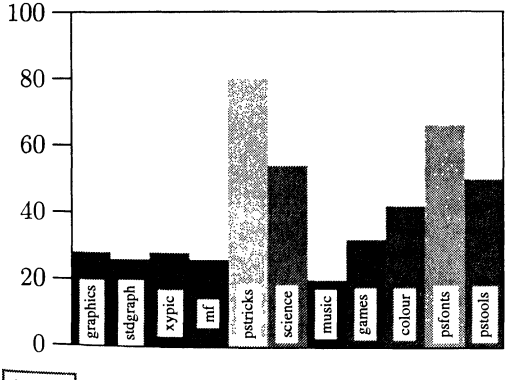



```
path s; numeric x,y;
draw begingraph(2in,1.75in);
gdata ("students.dat",c,
 x:=(scantokens c1) * 12;
 y:=(scantokens c2);
 augment.s((x-5,0)--
 (x-5,y)-- (x+5,y)--
 (x+5,0));
 if y < 0: glabel.top(c2,(x,0)); fi
 if y > 0: glabel.bot(c2,(x,0)); fi
);
gfill s--cycle withcolor .5white;
frame.llft;
endgraph;
```

3-3-14

Мы явным образом вычислили координаты углов каждого столбца и отвели каждому место на оси x путем умножения исходных значений x на 12. Сами столбцы простираются по горизонтали на 5 единиц в каждую сторону от соответствующей данным точки, поэтому между каждыми двумя столбцами остается зазор в 2 единицы.

Сходный подход был использован при создании следующей диаграммы (также приведенной на цветной вклейке I(d)), на которой показано число страниц в каждой главе этой книги на одной из стадий ее подготовки. На этот раз каждый столбец был нарисован отдельно, чтобы их можно было раскрасить в соответствии с полученными значениями. Эта работа поручена макро, который также печатает повернутую копию каждого столбца. Печать надписей на оси x отключена, поскольку названия заданы явным образом.



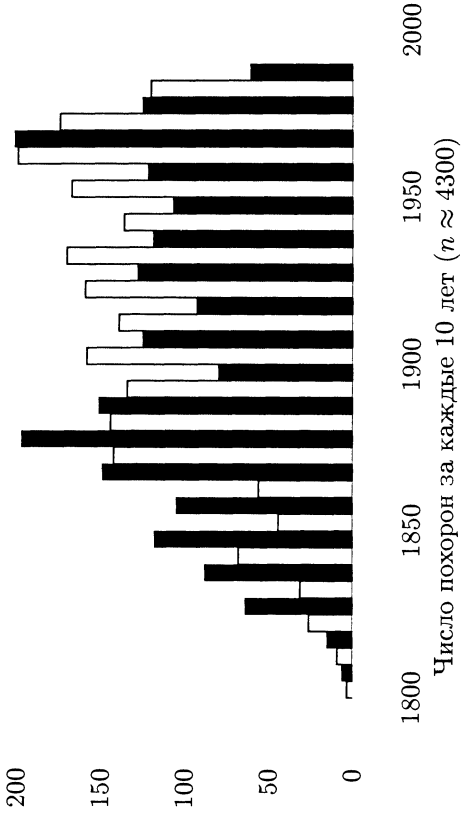
3-3-15

```
path m; numeric n,width;
width:=20; defaultscale:=0.6; n:=0;
def bar(expr name,value) =
 gfill(n,0)--(n,value)--
    (n+width,value)--
    (n+width,0)--cycle
 withcolor
    value/100,value/100,value/100);
 picture p;
 p = name infont defaultfont
  scaled defaultscale rotated 90;
 glabel.rt
  (image(unfill bbox p; draw p),
        (n,10));
 n:=n+width; enddef;
draw begingraph(2.25in,1.75in);
setrange((0,0),(11*width,100));
autogrid(,otick.lft);
gdata("chap.dat",c,bar(c1,
            (scantokens c2)););
endgraph;
```

Считываемое из первого столбца данных значение типа «строка» помещается в графическую переменную METAPOST'а с помощью команды низкого уровня `infont`. Это дает нам возможность воспользоваться командой `bbox` для определения требующегося тексту места, которое заполняется белым цветом с помощью команды `unfill`. Полезным макро, создающим рисунок на основе некоторой последовательности графических команд, является `image`. Этот рисунок используется в качестве метки. Исходные данные для этой диаграммы начинаются следующим образом:

```
graphics 28
stdgraph 26
xypic 28
mf 26
...
```

Мы можем также представить полученные ранее «десятилетние» данные в виде двойной гистограммы, на которой столбцы мужской и женской смертности расположены бок о бок. Чтобы добиться этого, формируются две отдельные траектории, одна из которых закрашивается, а другая остается в виде контура:




```
path m[],w[];
def wcheck(expr decade,value) =
 augment.w1(decade,0);
 augment.w1(decade,value);
 augment.w1(decade+5,value);
 augment.w1(decade+5,0);
enddef;
def mcheck(expr decade,value) =
 augment.m1(decade+5,0);
 augment.m1(decade+5,value);
 augment.m1(decade+10,value);
 augment.m1(decade+10,0);
enddef;
draw begingraph(3.75in,2in);
gdata ("decade.dat",y,
  wcheck((scantokens y1),
         (scantokens y2));
  mcheck((scantokens y1),
         (scantokens y3)););
gfill m1--cycle;
gdraw w1;
glabel.bot (btex
   Число похорон за каждые 10 лет
   ($n \approx 4300$) etex,OUT);
frame.llft;
endgraph rotated 90;
```

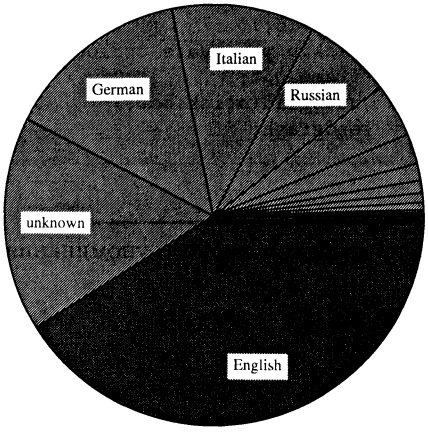
3-3-16

При определенном старании на основе сходных идей можно нарисовать даже круговую диаграмму. В следующем примере данные с надгробных плит Протестантского кладбища считываются в следующей форме:

```
Romanian 1 0.02796420582
Czech 2 0.05592841163
.....
Italian 391 10.93400447
German 508 14.20581655
unknown 599 16.75055928
English 1462 40.8836689
```

Здесь во втором столбце указано число погребенных представителей каждой национальности, а чтобы сделать формирование диаграммы проще, в третьем

столбце в процентах указана доля от общего количества похороненных. Для построения каждого сегмента применяется макро buildcycle, ищущий наименьшую замкнутую фигуру, получающуюся при объединении окружности и двух линий, проведенных из центра под углами, открывающими и закрывающими данный сегмент. Заполняющий сегмент цвет определяется на основе процентной величины.

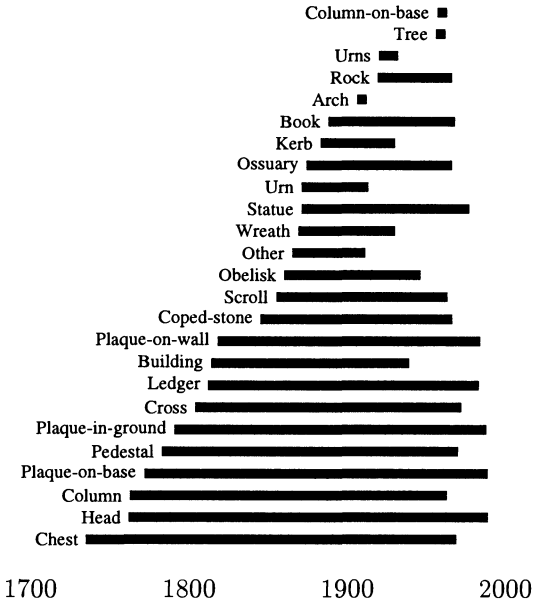


3-3-17

```
numeric r,last; path c,w;
r:=5; c:=fullcircle scaled 2r;
last:=0.0;
def wedge (expr lang,value,perc) =
 numeric current,n,half,xoff,yoff;
 picture p;
 n:=perc*3.6;
 current:=last+n; half:=last+(n/2);
 w:=buildcycle((0,0)--(2r,0) rotated last
  c, (2r,0)--(0,0) rotated current);
 gfill w withcolor (0,0.8-(perc/100),0);
 gdraw w;
 if perc > 5:
 p = lang infont defaultfont
    scaled defaultscale;
 glabel(image(unfill bbox p; draw p),
  3/4r*dir(half));
 fi;
 last:=current;
enddef;
draw begingraph(2.5in,2.5in);
defaultscale:=0.6;
gdata ("langs.dat",c,%
  wedge(c1, (scantokens c2),%
   (scantokens c3)););
autogrid(,); frame withcolor white;
endgraph;
```

Другой способ создания этой диаграммы с помощью пакета PSTricks будет рассмотрен на с. 165. Некоторого внимания заслуживает размещение подписей на этой круговой диаграмме. Они расположены в центре каждого сегмента, на расстоянии трех четвертых радиуса от центра. Альтернативный алгоритм выбора координаты размещения подписи, использующий функции синус и косинус языка META, мог бы иметь вид `((r*3*cosd(half))/4,(r*3*sind(half))/4))`.

На диаграммах еще одного типа шкала вдоль оси x линейна, а ось y используется просто для сравнения данных. На следующей диаграмме данные нашего кладбища иллюстрируют первые и последние случаи сооружения надгробий каждого типа. Текст программы вполне понятен за исключением того, что мы рисуем линии пером (квадратной формы) измененного размера и возвращаемся к тонкой линии, чтобы нарисовать шкалу и рамку (только внизу, так как шкала вдоль оси y не является линейной).




```
draw begingraph(2.5in,3in);
n:=10;
defaultscale:=0.7;
pickup pensquare scaled 3.5pt;
setrange((1700,0),
        (whatever,whatever));
gdata("stones.dat", s,
gdraw ((scantokens s2),n)--
  ((scantokens s3),n);
glabel.lft(s1,
        (scantokens s2)-3,n);
n:=n+16;);
pickup pensquare scaled .5pt;
frame.bot;
autogrid(otick.bot,);
endgraph;
```

3-3-18

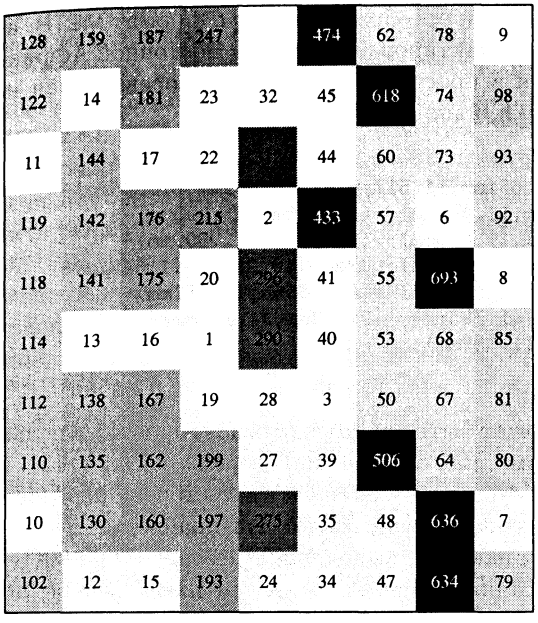
Данные, упорядоченные в соответствии со временем первого упоминания, начинаются так:

```
Chest 1738 1966
Head 1765 1986
Column 1766 1960
Plaque-on-base 1775 1986
Pedestal 1786 1967
Plaque-in-ground 1794 1985
```

Наш последний пример еще более необычен. Мы хотим изобразить данные в виде таблицы и закрасить каждую клетку таблицы в соответствии со стоящей в ней величиной. В файле данных первые два столбца содержат координаты нижнего левого угла клетки таблицы, третий столбец содержит абсолютное значение, а четвертый — эту же величину в процентах:

```
2 1 102 85
2 2 10 98
2 3 110 84
2 4 112 83
2 5 114 83
...
```

Текст печатается белым или черным цветом в зависимости от величины в процентах. Этот же рисунок, построенный с помощью LaTeX'овской таблицы и пакета color, показан на с. 378.



3-3-19

```
def sq(expr x,y,num,perc) =
 gfill(x,y)--(x+10,y)--
  (x+10,y+10)--(x,y+10)--cycle
 withcolor
  (perc/100,perc/100,perc/100)
 glabel(num,(x+5,y+5))
   if perc < 50:
      withcolor white  fi;
enddef;
defaultscale:=0.7;
draw begingraph(70mm,80mm);
setrange((20,10),(110,110));
autogrid(,);
gdata ("pot.dat",c,
 sq((scantokens c1)*10,
 (scantokens c2)*10,
 c3, (scantokens c4)););
endgraph;
```

3.4 Средства LaTeX'а для использования METAFONT

Хотя язык METAFONT позволяет создавать прекрасные рисунки, он далеко не прост с точки зрения того, кто использует его нерегулярно для построения относительно несложных диаграмм. Существует три способа справиться с этой проблемой:

1. воспользоваться библиотеками программ на языке METAFONT высокого уровня. Удивительно, но есть несколько таких библиотек. Среди них METAPOST'овский макропакет graph (см. разд. 3.3.2 на с. 98), по-видимому, является наиболее «укомплектованным»;
2. воспользоваться интерактивным графическим пакетом, умеющим выводить результат в формате METAFONT. Существует программа fig2mf, конвертирующая файлы, созданные программой xfig в операционной системе Unix, но никто не занимается ее обновлением;
3. воспользоваться каким-нибудь LaTeX'овским пакетом, порождающим требуемый текст на языке METAFONT.

Поскольку большинство интерактивных графических пакетов пользуются весьма упрощенным внутренним представлением своих рисунков, результаты их работы, переведенные на язык METAFONT, имеют ограниченное применение. Таким образом, мы могли бы прибегнуть к программе gnuplot или AutoCAD,

сохранить результат в формате HPGL и преобразовать его в формат META, но полученный код вряд ли поддается редактированию. Карта Восточной Африки (рис. 3.5 на с. 121), нарисованная в программе AutoCAD и переведенная на язык METAFONT, порождает строки, наподобие следующих:

```
pickup pencircle scaled 0.1mm;
draw (10.732mm,11.563mm)--(10.651mm,11.517mm)--(10.592mm,11.505mm)--
(10.545mm,11.493mm)--(10.487mm,11.493mm)--(10.440mm,11.505mm)--
(10.405mm,11.540mm)--(10.346mm,11.563mm)--(10.299mm,11.599mm)--
(10.264mm,11.622mm)--(10.217mm,11.645mm); draw (10.217mm,11.645mm)--
(10.159mm,11.680mm)--(10.100mm,11.727mm)--(10.065mm,11.751mm)--
(10.007mm,11.786mm)--(9.960mm,11.798mm)--(9.913mm,11.821mm)--
(9.714mm,11.868mm)--(9.667mm,11.868mm);
```

Программа такого рода вряд ли может быть вручную отредактирована с целью раскрасить страны другими цветами, как это сделано на рис. 3.5 на с. 121. В данном случае единственное преимущество METAFONT'а состоит в переносимости на любые устройства, для которых существуют dvi-драйверы.

Самым сложным LaTeX'овским пакетом, выводящим результаты на METAFONT'е, наверное, является написанный Торстеном Охлем пакет feynmf, подробно обсуждаемый в разд. 6.4 на с. 272. Пакет feynmf был разработан для рисования диаграмм Фейнмана, используемых в физике высоких энергий, и служит хорошим примером высокоспециализированного языка, тщательно продуманного для того, чтобы хорошо решать единственную задачу. Его близнец — пакет feynmp — вместо METAFONT'а использует METAPOST. В следующем разделе мы обсудим пакет более широкого назначения mfpic.

3.4.1 Пакет mfpic

Пакет mfpic был задуман и реализован Томасом Литрамом; в настоящее время его поддержкой и обновлением занимается Джеффри Тобин. Он состоит из пакета на TeX'е (который можно использовать вместе с LaTeX'ом) и служебной библиотеки макро на METAFONT'е. В этом разделе описывается текущая версия, по состоянию на ноябрь 1996 г., но работа над пакетом все еще продолжается и существуют планы появления новых возможностей (например, версии на METAPOST'е).

LaTeX'овские макро из пакета mfpic записывают текст на METAFONT'е во внешний файл и считывают получившийся в результате рисунок как символ из специального шрифта (для этого требуется два запуска LaTeX'а, если изменяются размеры рисунка, и один запуск METAFONT'а, если каким-либо образом меняется содержание рисунка). Каждый рисунок представляет собой один METAFONT'овский символ. Преимущество данной системы состоит в том, что пользователю LaTeX'а доступна вся мощь METAFONT'а как графического пакета, и при этом нет необходимости в изучении нового языка. Набор меток и текста обрабатываются макрокомандами на LaTeX'е, что тем самым решает

основную проблему, связанную с использованием METAFONT'а (по этой причине использовать METAPOST проще).

Один из главных недостатков пакета mfpic заключается в том, что пользователь обязательно должен не забыть обработать METAFONT'ом созданный «шрифт» и поместить результаты там, где TeX мог бы их найти. Вам также нужно следить за ресурсами METAFONT'а, которые легко могут оказаться исчерпанными в случае высокого разрешения. Мы были вынуждены ограничить разрешение рисунков для этой книги величиной 600 dpi для того, чтобы METAFONT не вышел за пределы своих возможностей. Кроме того, вы должны следить за тем, не вносились ли изменения в рисунки и нет ли необходимости новой обработки шрифта после каждого запуска LaTeX'а. Однако все это сравнимо с использованием BibTeX'а или программы makeindex в качестве этапа LaTeX'овского процесса и может быть спрятано в хорошо продуманных batch-файлах.

Используя mfpic, вы должны помнить, что имеете дело со *шрифтом*. Это означает, во-первых, что в одном шрифте у вас не может быть более 255 рисунков, и, во-вторых, что вы должны учесть особенность вашей реализации TeX'а, которая обычно имеет ограничение на число загружаемых шрифтов. Если вам требуется создать несколько тысяч рисунков, то вам придется загрузить несколько шрифтов и внимательно следить за тем, чтобы отведенная под шрифты память не переполнилась.

Обычно от вас не требуется читать или менять METAFONT'овский файл со шрифтом, поскольку об этом заботятся макро из пакета mfpic. Однако, если этого потребуют обстоятельства, программа на METAFONT'е может быть легко прочитана.

Вы должны окружить раздел документа, в котором предполагается использовать пакет mfpic, макрокомандами, открывающими и закрывающими METAFONT'овский файл, содержащий подлежащие включению рисунки:

`\opengraphsfile{`*file*`}` ... `\closegraphsfile`

Как правило, вам будет достаточно одного файла на документ, но, если у вас больше 255 рисунков, вам придется открывать новые файлы, и тогда вы можете предпочесть иметь по одному отдельному файлу с графикой для каждого раздела.

Все рисунки помещаются внутри окружения `mfpic`:

`\begin{mfpic}[`x_{scale}`][`y_{scale}`]{`x_a`}{`x_b`}{`y_a`}{`y_b`}` ... `\end{mfpic}`

Эти параметры задают систему координат и размер рисунка:

x_{scale} Шаг шкалы вдоль оси x в данной системе координат, кратен длине величины `\mfpicunit` (принятое по умолчанию значение которой равно 1 pt).

y_{scale} Шаг шкалы вдоль оси y в данной системе координат. Если параметр y_{scale} не задан, его значение считается равным x_{scale}.

x_a Нижняя граница координаты x.
x_b Верхняя граница координаты x.
y_a Нижняя граница координаты y.
y_b Верхняя граница координаты y.

Четыре обязательных параметра задают размер рисунка, а два необязательных позволяют изменить его масштаб. Как и в случае LaTeX'овского окружения `picture`, пакет не проверяет, попадают ли элементы рисунка в указанные пределы. Они вполне могут накладываться на окружающий текст.

Пакет mfpic поддерживает примерно дюжину примитивных графических команд, набор модификаторов объектов и несколько макро высокого уровня для выполнения функций рисования и осуществления специальных преобразований. По умолчанию каждая из простейших графических команд рисует самостоятельную линию или форму, но их можно комбинировать для получения более сложных образов:

```
\begin{connect}     ...     \end{connect}
```

Внутри окружения `connect` вы можете соединить объекты вместе. Дополнительные отрезки линий будут проведены от последней точки одного объекта к первой точке следующего. Даже если объекты являются замкнутыми (наподобие окружностей), соединительные линии все равно, к сожалению, будут проведены.

По умолчанию объекты изображаются прямыми линиями. Вы можете изменить это соглашение при помощи команды `\setrender`:

`\setrender{`*commands*`}`

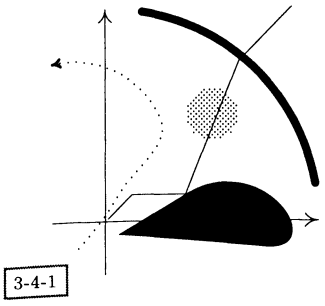
Все объекты внутри LaTeX'овской области действия этой команды будут изображаться при помощи команд *commands*. Тем самым вводится более краткая нотация, например для того, чтобы нарисовать несколько объектов в одном стиле. Единственная команда

```
\setrender{\gfill}
```

приведет к тому, что все последующие объекты будут закрашены.

Все координаты и расстояния обычно представлены в TeX'овских пунктах. Углы выражаются в градусах и измеряются против часовой стрелки. Поведение примитивов для рисования зависит от набора переменных, которые вы можете задать глобально. В большинстве это команды, задающие длины, которые изменяются с помощью команды `\setlength`. Пакет mfpic также располагает набором команд, которые могут быть использованы в любой точке рисунка для изменения последующих команд.

В следующем примере продемонстрировано действие целого ряда простых графических команд пакета mfpic. Последняя строка содержит *составные* модификаторы (`\gfill` и `\bclosed`).



3-4-1

```
\usepackage {mfpic}
\begin{mfpic}[1]{-20}{80}{-20}{80}
\pen{.3pt}\axes
\lines{(1,1),(10,10),(30,10),(50,60),(70,80)}
\pen{3pt}
\arc[p]{(0,0),10,80,80}
\pen{1pt}
\shade\circle{(40,40),10}
\arrow\dotted
 \curve{(-10,-10),(5,10),(20,40),(-20,60)}
\gfill\bclosed\lines{(60,-10),(5,-5),(30,10)}
\end{mfpic}
```

Основные базовые команды для рисования

`\arc[`*c*`]{`$(x_0,y_0),(x_1,y_1),\theta$`}`
Нарисовать дугу с центром в (x,y), начинающуюся в точке (x_1,y_1), охватывающую θ градусов.

`\arc[`*p*`]{`$(x,y),\theta_a,\theta_b,r$`}`
Нарисовать дугу с центром в (x,y) и радиусом r между углами θ_a и θ_b.

`\arc[`*s*`]{`$(x_0,y_0),(x_1,y_1),\theta$`}`
Нарисовать дугу от точки (x_0,y_0) до точки (x_1,y_1), имеющую угол θ градусов.

`\arc[`*t*`]{`$(x_0,y_0),(x_1,y_1),(x_2,y_2)$`}`
Нарисовать дугу, проходящую через три данные точки.

`\axes`
Нарисовать оси координат, проходящие вдоль всего рисунка. На конце каждой оси нарисовать стрелку-указатель длины `\axisheadlen`.

`\circle{`$(x,y),r$`}`
Нарисовать окружность радиуса r с центром в точке (x,y).

`\curve{`$(x_0,y_0),(x_2,y_2),\ldots,(x_n,y_n)$`}`
Нарисовать кривую Безье, проходящую через данные точки.

`\cyclic{`$(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n)$`}`
Нарисовать кривую Безье, проходящую через данные точки, и соединить первую и последнюю точки для создания замкнутого объекта.

`\ellipse[`θ`]{`$(x,y),r_x,r_y$`}`
Нарисовать эллипс с центром в точке (x,y), имеющий «ширину» r_x и «высоту» r_y. Если задан угол θ, то эллипс поворачивается на эту величину.

`\lines{`$(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n)$`}`
Нарисовать отрезки линий между точками. Команда `\polylines` служит псевдонимом команды `\lines`.

`\point{`$(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n)$`}`
Нарисовать небольшие закрашенные окружности с центрами в данных точках. Диаметр окружностей равен величине `\pointsize`.

`\polygon{`$(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n)$`}`
Нарисовать замкнутый многоугольник с вершинами в данных точках.

`\rect{`$(x_0,y_0),(x_1,y_1)$`}`
Нарисовать прямоугольник с вершинами в точках (x_0,y_0) и (x_1,y_1).

`\turtle{`$(x,y),(\theta_0,l_0),(\theta_1,l_1),\ldots$`}`
Нарисовать отрезок линии, начинающийся в точке (x,y), а затем отрезки линий длины l_i, проведенные под углами θ_i по отношению к направлению предыдущего отрезка.

`\sector{`$(x,y),\theta_1,\theta_2,r$`}`
Нарисовать сегмент круга, простирающийся от угла θ_1 до θ_2, с центром в (x,y) и радиусом r.

`\xmarks{`$(x_0),(x_1),\ldots,(x_n)$`}`
`\ymarks{`$(y_0),(y_1),\ldots,(y_n)$`}`
Нарисовать риски на осях x и y в точках, указанных в списке. Длина рисок равна параметру `\hashlen`.

Параметры для рисования

`\dashlen` (по умолчанию 4pt)
Длина штрихов.

`\dashspace` (по умолчанию 4pt)
Расстояние между штрихами.

`\hashlen` (по умолчанию 4pt)
Длина рисок на осях.

`\shadespace` (по умолчанию 1pt)
Расстояние между точками, нарисованными командой `\shade`.

`\hatchspace` (по умолчанию 3pt)
Расстояние между линиями, нарисованными командой `\hatch`.

`\mfpicunit` (по умолчанию 1pt)
Базовая единица длины для рисунков.

`\pointsize` (по умолчанию 2pt)
Диаметр окружности в макро `\point`.

`pointfilled` (по умолчанию истина)
Булева переменная, определяющая, закрашиваются ли нарисованные командой `\point` окружности.

`\headlen` (по умолчанию 3pt)
Длина стрелки-указателя.

`\axisheadlen` (по умолчанию 5pt)
Длина стрелки-указателя, рисуемой командой `\axes`.

Глобальные модифицирующие команды

`\headshape{`*ratio*`}{`*tension*`}{`*filled*`}`
Задать внешний вид стрелок-указателей. Параметр *ratio* равен отношению ширины стрелки к ее длине (по умолчанию 1). Параметр *tension* представляет собой напряжение кривой Безье, используемой для рисования стрелки (по умолчанию 1). Параметр *filled* является булевой переменной, определяющей, следует ли закрашивать стрелку (по умолчанию — да, true).

`\pen{`*pensize*`}`
Задать ширину пера для рисования (по умолчанию 5pt).

`\dashlineset`
Параметры `\dashlen` и `\dashspace` устанавливаются равными 4pt.

`\dotlineset`
Параметр `\dashlen` устанавливается равным 1pt, а `\dashspace` — 2pt.

`\darkershade`
Затенение делается более густым путем умножения параметра `\shadespace` на коэффициент 5/6.

`\lightershade`
Затенение делается более светлым путем умножения параметра `\shadespace` на коэффициент 6/5.

Модифицирующие рисунок команды

Любой объект, нарисованный какой-либо простейшей командой, или составной объект, созданный с помощью окружения `connect`, может быть модифицирован одной или несколькими *префиксными* командами. Они оказывают влияние на указанный следом за ними объект и чаще всего применяются к замкнутым объектам, таким, как окружности, эллипсы и т. д.

`\arrow[l`*length*`][r`θ`][b`*distance*`]`
В последней указанной точке объекта рисуется стрелка-указатель. Три необязательных параметра позволяют вам изменить ее внешний вид: длина стрелки задается параметром *length* (по умолчанию равна `\headlen`), угол — параметром θ, а расстояние от последней точки — параметром *distance* (это дает вам возможность рисовать двойные стрелки). Необязательные параметры могут располагаться в произвольном порядке, но перед ними обязательно должны указываться специальные ключевые символы.

`\bclosed`, `\lclosed`, `\sclosed`, `\cbclosed`
Незамкнутая фигура превращается в замкнутую путем соединения крайних точек. `\lclosed` рисует прямую линию, `\bclosed` — кривую Безье, `\sclosed` — гладкую кривую, а `\cbclosed` — кубический B-сплайн.

`\dotted[`*dash*`][`*gap*`]`
Составляющие объект линии рисуются штрихами или пунктиром. Длина штрихов определяется параметром `\dashlen`, но может быть изменена заданием параметра *dash*. Аналогичным образом, расстояние между штрихами может быть задано с помощью параметра *gap* (по умолчанию оно равно `\dashspace`).

`\draw`
Рисуется контур объекта. Эта команда бывает полезна, если вы хотите нарисовать и затем закрасить какой-то объект.

`\gclear`
Удаляется все, что находится внутри замкнутой области.

`\gfill`
Закрашивается замкнутый объект.

`\hatch[`*distance*`]`
Замкнутая область заполняется крестиками. По умолчанию расстояние между

линиями равно \hatchspace, но может быть изменено заданием необязательного параметра *distance*. Определены и другие команды, выполняющие штрихование.

\reverse
Меняется ориентация объекта. Эта команда полезна для изменения порядка следования точек внутри окружения connect, в результате чего выбирается другая крайняя точка.

\rotatepath{*(x,y),θ*}
Объект поворачивается на θ градусов вокруг точки (x,y).

\shade[*distance*]
Замкнутая область заполняется точками. По умолчанию расстояние между точками равно \shadespace, но может быть изменено заданием необязательного параметра *distance*.

Команды заполнения *не* рисуют контур объекта и должны идти первыми, если указывается более одного модификатора.

Аналитические кривые и функции

При анализе научных данных часто возникает необходимость построения функций и параметрических кривых. METAFONT очень хорошо подходит для задач подобного рода, а в пакет mfpic входят пять общих команд. У каждой из них есть необязательный параметр, принимающий значение s, означающее, что линия рисуется с помощью гладкой кривой Безье, или p, означающее, что она рисуется с помощью прямых линий. По умолчанию его значение равно s для всех команд за исключением \btwnfcn и \plrregion.

У каждой команды есть обязательный параметр для задания минимальных и максимальных значений функции и шага изменения. Остальные параметры являются функциями, передаваемыми METAFONT'у для расчета их значений.

\function[*type*]{*minimum,maximum,step*}{*expr*}
Нарисовать функцию *expr*.

\parafcn[*type*]{$minimum_t$,$maximum_t$,*step*}{$(expr_1,expr_2)$}
Нарисовать параметрическую кривую $(x(t),y(t)) = (expr_1,expr_2)$.

\plrfcn[*type*]{*minimum,maximum,step*}{*expr*}
Нарисовать кривую *expr* в полярных координатах.

\btwnfcn[*type*]{*minimum,maximum,step*}{$expr_1$}{$expr_2$}
Нарисовать область, заключенную между функциями $f(x) = expr_1$ и $g(x) = expr_2$, слева и справа эта область ограничивается вертикальными линиями, проведенными в точках *minimum* и *maximum*. По умолчанию функция изображается набором отрезков прямых линий.

\plrregion[*type*]{*minimum,maximum,step*}{expr}
Нарисовать область в полярных координатах, определенную выражением $r = expr(\theta)$. Значениями θ являются углы (измеренные в *градусах*) между направлениями *minimum* и *maximum*. По умолчанию эта функция изображается набором отрезков прямых линий.

Размещение аннотаций

Две команды, предназначенные для размещения аннотаций, полностью обрабатываются LaTeX'ом и ничего не передают в METAFONT'овский файл:

`\tlabel[`*pos*`](`*x*`,`*y*`){`*text*`}`

Эта команда помечает график меткой *text* в точке (x,y). По умолчанию в точке (x,y) располагается нижний левый угол этого текста, но можно также использовать необязательный параметр позиционирования *pos*. Действие параметра *pos* похоже на действие параметра позиционирования боксов в окружении `picture`; оно указывает относительное расположение метки по отношению к опорной точке: первый символ предназначен для задания вертикального положения (`t` — вверху, `c` — в центре и `b` — внизу), а второй — для задания горизонтального положения (`l` — слева, `c` — в центре и `r` — справа).

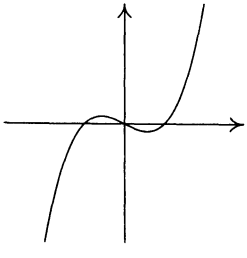
`\tcaption[`*maxwidth*`,`*linewidth*`]{`*text*`}`

Эта команда центрирует подпись под вашим рисунком. С помощью необязательных параметров вы можете указать, на строки какой длины следует разбить текст подписи. Если подпись шире, чем величина *maxwidth* × ширина рисунка, то текст набирается в виде абзаца с длиной строки *linewidth* × ширина рисунка. По умолчанию значения аргументов равны `[`*1.2,1.0*`]`.

Примеры

Мы завершим описание пакета mfpic несколькими примерами. Они устроены так, чтобы продемонстрировать скорее типичные способы применения, чем все доступные возможности этого пакета.

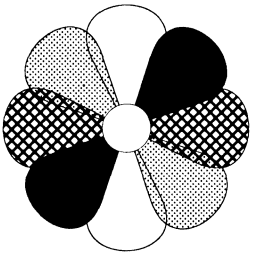
Следующий простой пример построения функции демонстрирует прозрачность инструкций пакета mfpic:



3-4-2

```
\usepackage {mfpic}
\begin{mfpic}[15]{-3}{3}{-3}{3}
\axes
\function{-2,2,0.1}{((x**3)-x)/2}
\end{mfpic}
```

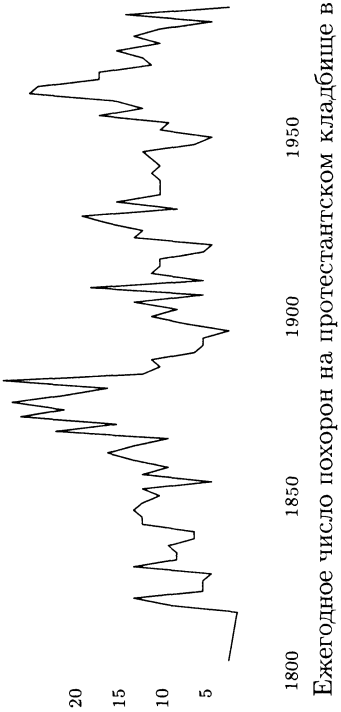
Преимущества, предоставляемые программируемостью LaTeX'а, могут быть использованы при выполнении многократных операций рисования. В следующем примере сначала определяется макро для рисования цветочного лепестка, который затем вызывается несколько раз с различными модификаторами. Чтобы выполнить совокупный поворот, в этом примере используется аффинное преобразование.



```
\usepackage {mfpic}
\begin{mfpic}[.9]{-50}{50}{-50}{50}
\pen{.3pt}
\newcommand{\petal}%
 {\bclosed\lines{(30,10),(0,0),(10,30)}}
\newcommand{\halfflower}%
{\begin{coords}
 \gfill\petal \turn{45}\petal
 \turn{45}\shade\draw\petal
 \turn{45}\hatch\draw\petal\end{coords}}
\halfflower\turn{180}\halfflower
\gclear\draw\circle{(0,0),10}
\end{mfpic}
```

3-4-3

Следующий рисунок, на котором изображена простая диаграмма, демонстрирует использование меток. Снова для того, чтобы параметризовать процесс их добавления, мы прибегли к LaTeX'овскому макро. Поскольку диапазон изменения значений x гораздо шире, чем значений y, удобно воспользоваться необязательным параметром окружения mfpic для изменения масштаба вдоль оси y. Кроме того здесь показано, как можно манипулировать рисунком как единым целым с помощью стандартных макро поворота.

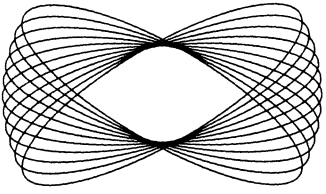


Ежегодное число похорон на протестантском кладбище в Риме.

```
\usepackage {graphics,mfpic}
\usepackage [cp866]{inputenc}
\usepackage [LCY]{fontenc}
\rotatebox{90}{\fbox{%
\begin{mfpic}[1.35][3.25]{-10}{212}{-5}{35}
\newcommand{\XmyLabel}[2]{%
 \tlabel[bc](#1,-5){{\scriptsize#2}}
   \lines{(#1,0),(#1,-2)}}
\newcommand{\YmyLabel}[1]{%
 \tlabel[br](-5,#1){{\scriptsize#1}}
   \lines{(0,#1),(-2,#1)}}
\XmyLabel{0}{1800}\XmyLabel{50}{1850}
\XmyLabel{100}{1900}\XmyLabel{150}{1950}
\YmyLabel{5}\YmyLabel{10}
\YmyLabel{15}\YmyLabel{20}
\arrow\lines{(0,30),(0,0),(212,0)}
\lines{(03,3), (17,2), (19,10), (21,14),
(23,6), (26,6), (28,5), (30,14), (32,9),
(34,9), (36,10), (38,7), (40,7),
(42,13), (44,13), (46,14), (48,13),
(50,11), (52,13), (54,5), (56,13),
(58,10), (60,14), (62,17), (64,14),
(66,10), (68,23), (70,16), (72,27),
(74,22), (76,28), (78,22), (80,17),
(82,29), (84,13), (86,11), (88,12),
(90,7), (92,6), (94,6), (96,3), (98,8),
(100,12), (102,9), (104,14), (106,6),
(108,19), (110,6), (112,12), (114,11),
(116,11), (118,6), (120,5), (122,14),
(124,13), (126,17), (128,20), (130,9),
(132,16), (134,11), (136,11), (138,11),
(140,12), (142,11), (144,12), (146,13),
(148,7), (150,5), (152,11), (154,10),
(156,18), (158,13), (160,16), (162,26),
(164,25), (166,18), (168,18), (170,12),
(172,13), (174,16), (176,11), (178,14),
(180,11), (182,5), (184,15), (186,3)}
\tcaption[.98,.75]{Ежегодное число похорон
  на протестантском кладбище в Риме.}
\end{mfpic}}}
```

3-4-4

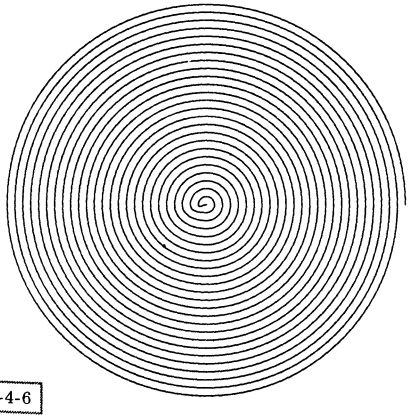
При построении сложных рисунков полезным инструментом оказывается стандартный LaTeX'овский пакет ifthen, позволяющий организовать циклическое выполнение процедур:



3-4-5

```
\usepackage {mfpic,ifthen}
\newcounter{mfcntA}
\begin{mfpic}[1.5]{-30}{40}{-25}{25}
\setcounter{mfcntA}{-30}
  \whiledo{\themfcntA<31}{%
    \ellipse[\themfcntA]{(0,0),40,12}
    \addtocounter{mfcntA}{6}}%
\end{mfpic}
```

Еще одним применением циклов оказывается вычерчивание спирали на основе того же приема, что и на с. 79. Конечно, это нельзя считать «корректным» способом рисования этой фигуры, которую, возможно, было бы лучше нарисовать с помощью команды \parafcn:



3-4-6

```
\usepackage {mfpic,ifthen}
\begin{mfpic}[1.5]{-40}{50}{-48}{50}
\newcounter{mfcntA}
\newcounter{mfcntB}
\setcounter{mfcntA}{50}
\setcounter{mfcntB}{48}
\whiledo{\themfcntA>1}{%
  \expandafter\curve{(\themfcntA,0),
    (0,\themfcntA),
    (-\themfcntA,0),
    (0,-\themfcntB),
    (\themfcntB,0)}
   \addtocounter{mfcntA}{-2}%
   \addtocounter{mfcntB}{-2}%
 }
\end{mfpic}
```

Другие возможности

В пакете mfpic определен набор аффинных преобразований (поворот, отражение, перенос, изменение масштаба и растяжение), подробно описанных в документации. Одно из них (\turn) было использовано выше в примере с «цветком» для поворота текущего графического состояния. Все эти преобразования обладают свойством накопления, но их действие при необходимости может быть ограничено пределами, задаваемыми окружением coords (что продемонстрировано выше на примере «лепестка»).

Пользователи, желающие создать новые инструменты, имеют возможность записывать METAFONT'овские команды непосредственно в выходной файл с целью сохранения и повторного использования METAFONT'овских траекторий и создания сложных узоров для раскрашивания объектов.

3.5 Пакет mftoeps: прямая связь между METAFONT'ом и PostScript'ом

Иногда у вас может возникнуть желание очистить выходной файл от программ на METAFONT или улучшить их с помощью какого-нибудь из замечательных интерактивных графических пакетов. Один из способов сделать это состоит в том, чтобы воспользоваться METAPOST'ом, а результат отредактировать в программе Adobe Illustrator (самая последняя версия которой может работать почти с любым текстом на PostScript'е). Однако более точный результат может быть получен с помощью элегантного пакета под названием mftoeps, написанного Богуславом Яцковским, Петром Пяновским и Мареком Рыцько (Jackowski, 1995). Эта METAFONT'овская программа считывает графические команды и переписывает их в log-файл в виде инкапсулированного PostScript'а. В пакете также содержатся вспомогательные программы, извлекающие полученный результат, который может быть отредактирован программами Corel Draw и Adobe Illustrator.

Эта система по большей части имеет дело с обычным кодом на METAFONT'е. Для того чтобы ее активизировать, вам необходимо загрузить METAFONT'овскую библиотеку `mftoeps.mf` и установить переменную `yesEPS` равной 1. Тогда обычные команды `draw` и `fill` замещаются специальными разновидностями, показанными в следующем примере:

```
input mftoeps;
EPS_mode_setup;
beginchar("A",2cm#,1cm#, 0cm#);
 set_BB 0,-d,w,h;
 write_preamble "test";
 draw_C (0,0)..(1cm,1cm)..(2cm,0);
 write_postamble;
endchar;
end.
```

Доступны следующие дополнительные команды:

`EPS_mode_setup`
: Используйте ее вместо `mode_setup`. Ее значение устанавливается равным 1 для порождения EPS.

`write_preamble` *name*
: Инициализирует процесс записи PostScript'а в log-файл (соответствующий файлу *name*).

`write_postamble`
: Завершает запись PostScript'а в log-файл.

`find_BB`
: На основе следующего за ним списка METAFONT'овских траекторий находит обрамляющий бокс и записывает его в текст программы на PostScript'е.

`set_BB`
: Записывает следующие за командой четыре числа или две разделенные запятой пары чисел в текст программы на PostScript'е в качестве обрамляющего бокса. Перед командой `write_preamble` должна стоять либо эта команда, либо `find_BB`.

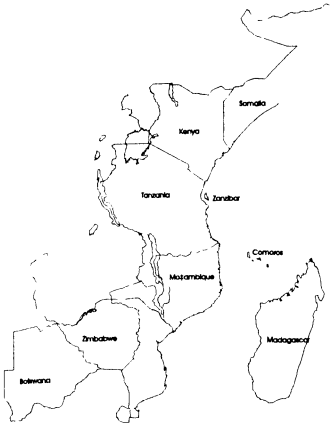


Рис. 3.3. Созданная в AutoCAD'е карта, преобразованная в METAFONT.

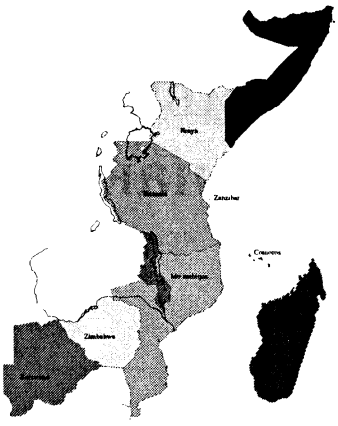


Рис. 3.4. METAFONT'овский рисунок, улучшенный с помощью программы Corel Draw.

`draw_C` *paths*
Записывает траектории *paths* (разделенный запятыми список) в PostScript'овский файл в виде многоугольных объектов из программы Illustrator.

`fill_C` *paths*
Записывает траектории *paths* (разделенный запятыми список) в PostScript'овский файл в виде закрашенных областей из программы Illustrator.

`fix_line_width` *number*
Устанавливает толщину линий (по умолчанию она равна 0.4pt) для объектов, созданных с помощью команды `draw_C`.

`fix_fill_cmyk` *four numbers*
Устанавливает цвет в формате CMYK для закрашивания объектов, созданных с помощью команды `fill_C`.

`fix_drawn_cmyk` *four numbers*
Устанавливает цвет линий в формате CMYK для объектов, созданных с помощью команды `draw_C`.

На рис. 3.5 представлена карта, нарисованная в программе AutoCAD и преобразованная в формат METAFONT. На рис. 3.5 показана та же самая карта, которая была преобразована в формат программы Adobe Illustrator и затем раскрашена с помощью программы Corel Draw. Хотя раскрашивание можно выполнить и силами METAPOST'а, часто оказывается удобнее работать с картой наподобие этой в интерактивной программе.

Встраивание PostScript'а в LaTeX: пакет PSTricks

Как мы видели в гл. 1, одним из способов представления графики в LaTeX'е является вставка создающих изображение примитивов низкого уровня, предназначенных для определенного устройства, в LaTeX'овские макро таким образом, чтобы доступной стала вся информация для набора и можно было бы работать в знакомой среде макропрограммирования. Когда целевым устройством оказывается нечто со столь богатыми возможностями, как язык PostScript, тогда может возникнуть очень мощная система. Хотя доступ к отдельным конструкциям языка PostScript реализован во многих макропакетах, наиболее полно это сделано в пакете PSTricks. В данной главе дан обзор его возможностей и продемонстрированы некоторые достоинства, возникающие в результате комбинирования LaTeX'а и PostScript'а.

Мы не будем пытаться полностью описать каждый макро из пакета PSTricks, так же как и давать примеры всех возможных комбинаций и трюков, так как для этого потребовалась бы самостоятельная большая книга. Однако мы постарались описать и привести примеры всех важных особенностей этого пакета.

Поскольку в пакете PSTricks есть очень много команд и параметров, в конце этой главы (разд. 4.13 на с. 186) мы приведем краткое описание большинства из них.

4.1 Компоненты пакета PSTricks

Пакет PSTricks, написанный Тимоти ван Зандтом, состоит из ядра, в которое входят создающие рисунок примитивы, реализованные с помощью конструкций `\special`, передающих команды PostScript драйверу. Помимо этого в пакет входит набор макро высокого уровня, предназначенных для конкретных целей. С их помощью вы можете:

- рисовать прямые линии, многоугольники, окружности и кривые;
- позиционировать и манипулировать TeX'овским текстом;

Таблица 4.1. Вспомогательные пакеты, входящие в PSTricks.

Пакет	*Функция*
pst-3d	Трехмерная графика
pst-char	Вычерчивание и заполнение траекторий символов
pst-coil	Витые и зигзагообразные объекты
pst-eps	Экспорт объектов непосредственно в eps-файлы
pst-fill	Дополнительный макро `\psboxfill`
pst-grad	Заполнение с плавным изменением цвета
pst-node	Размещение и соединение вершин
pst-plot	Построение диаграмм
pst-text	Набор текста вдоль траектории
pst-tree	Макро, создающие деревья

- на основе данных строить диаграммы со сложным образом помеченными осями;
- рисовать вершины и связи между ними (включая деревья);
- раскрашивать линии и заполнять объекты;
- определять новые графические команды.

Это — исключительно мощный пакет и для того, чтобы осознать его возможности, может потребоваться некоторое время. Пакет описан в van Zandt (1993a), а его реализация — в van Zandt and Girou (1994). В Girou (1994) замечательно проиллюстрированы его возможности и мы выражаем благодарность Дени Жиру за разрешение воспроизвести здесь некоторые из его примеров.

Этот пакет опирается на способность `dvi`-драйвера пропускать через себя тексты программ на PostScript'е и понимать, что им предстоит определенным контролируемым образом взаимодействовать с текстом на TEX'е. Говоря о драйвере, мы будем иметь в виду драйвер dvips, но данный пакет может работать и со многими другими драйверами. В руководстве по установке указывается, какими именно качествами должен обладать драйвер и как его настраивать (см. также разд. 4.12).

В отличие от многих других пакетов, описанных в этой книге, пакет PSTricks предназначен не для того, чтобы хорошо рисовать какой-нибудь один вид диаграмм. Он представляет собой среду программирования, реализующую максимально тесное взаимодействие TEX'а и PostScript'а, насколько это возможно с существующим в настоящее время программным обеспечением. Его преимуществами являются модульность, расширяемость и способность использовать всю мощь PostScript'а.

Бóльшая часть пакета PSTricks загружается в виде одного пакета, но для выполнения некоторых более сложных действий требуется загрузить дополнительный пакет с помощью команды `\usepackage`. Эти пакеты перечислены в табл. 4.1. Как правило, в последующих описаниях явно *не* указывается, какой файл требуется загрузить для реализации определенной функции, так как

обычно это очевидно. Однако в каждом примере текста программы присутствуют необходимые команды \usepackage. Часто для рисунков, в которых используются цвета, вместо pstricks загружается pstcol.

Поскольку пакет PSTricks все еще находится на стадии разработки, в будущем могут появиться новые модули. Весь материал из данной главы был протестирован с использованием версии, являвшейся общедоступной в конце 1996 г.

4.2 Основные базовые понятия пакета PSTricks

Большинство команд в пакете PSTricks рисуют определенный объект в точке с координатами, отсчитываемыми относительно текущей точки в TeX'е. В представлении TeX'а эти объекты не занимают никакого места. Они могут перемежаться с обычными командами или использоваться в графическом окружении для построения целого рисунка. Для рисунков обычно используется специальное LaTeX'овское окружение:

\begin{pspicture}*[*settings*] [*baseline*] $(x_0,y_0)(x_1,y_1)$ ... \end{pspicture}

TeX отводит место для прямоугольника с вершинами в точках (x_0,y_0) и (x_1,y_1). Как и в других PSTricks'овских командах, которые будут обсуждаться дальше, первая пара координат (x_0,y_0) является необязательной и по умолчанию считается равной $(0,0)$. Вариант этого окружения со звездочкой отсекает те части графических объектов, которые выходят за границы отведенной области. По умолчанию базовая линия проходит вдоль нижней границы этого бокса, но необязательный аргумент помещает ее на некотором расстоянии от нижней границы, равном доле *baseline*.

4.2.1 Команды и аргументы

Почти все команды имеют одинаковую (сложную) структуру. Им требуются некоторые или все следующие аргументы, каждый из которых обладает своими собственными ограничениями. Обязательные параметры заключены в фигурные скобки: {arg}. Необязательные параметры заключены в квадратные скобки: [par1=val1, ...]. Координаты заключены в круглые скобки: (x,y).

Поскольку PSTricks'овские макро могут быть очень сложными, со множеством разнообразных параметров, в приведенных здесь кратких описаниях макро (главным образом, в таблицах в конце этой главы) необязательные параметры помещены на сером фоне.

Общий синтаксис большинства команд имеет вид:

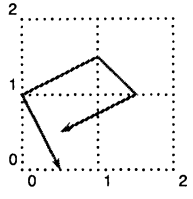
```
\command*[settings]{arrows/parameters}(coordinates)
```

Вариант команды со звездочкой обычно подразумевает, что изображаемый объект должен быть сплошным, а не контурным. Необязательные *настройки* (*settings*) в квадратных скобках состоят из набора пар типа «параметр—значение», которые для текущего объекта переопределяют значения, принятые в PSTricks по умолчанию, как обсуждается ниже.

Многие рисующие линии команды требуют введения аргумента, такого, как угол поворота, или имеют аргумент, указывающий, следует ли на каком-либо конце нарисовать стрелку-указатель и если да, то какой формы. Эти аргументы, помещенные в фигурные скобки, следуют за необязательными параметрами.

Для большинства объектов требуется указать одну или несколько координат. Они состоят из двух чисел, разделенных запятой и помещенных в круглые скобки. В случае нескольких координат каждая из них заключается в свою пару скобок. Ниже приведен простой пример, в котором на рисунок наложена сетка для демонстрации системы координат (команда `\showgrid` объясняется на с. 173):

4-2-1




```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,2)\showgrid
\psline[linewidth=1pt,linecolor=gray]
   {<->}(0.5,0)(0,1)(1,1.5)(1.5,1)(.5,.5)
\end{pspicture}
```

4.2.2 Задание графических параметров

В пакете PSTricks используется нотация, сходная с той, что была введена в разд. 2.2.2 для пакета graphicx, т. е. пары «параметр—значение». Она применяется для задания большого числа графических параметров, относящихся почти ко всем объектам. Эти параметры можно задавать двумя способами:

- для каждого объекта с помощью необязательных параметров типа «параметр—значение» в квадратных скобках, разделенных запятыми. В этом случае их действие является локальным для данного объекта и не требует никакого дополнительного группирования;
- для некоторой области текущего окружения с помощью команды `\psset`. Ее синтаксис имеет вид:

`\psset{`*par1=value1,par2=value2,...*`}`

Дополнительные символы пробелов допускаются только после запятой, разделяющей пары вида *par=value*. Графические параметры, которые могут устанавливаться для объектов, перечислены в табл. 4.5, начинающейся на с. 188. Первая группа параметров может относиться в большей или меньшей степени ко всему, а остальные применимы только к определенным командам.

4.2.3 Координаты и единицы измерения

По умолчанию единица измерения координатных осей x и y равна 1 см, но как мы скоро увидим, это может быть легко изменено. Пары координат можно задавать не только в принятых по умолчанию единицах, но и в любых единицах измерения TeX'а. Это справедливо для всех «количественных» настроек, так что `linewidth=.5` означает 0.5 того, что в данный момент является единицей измерения, но `linewidth=2pt` является абсолютным значением. Вы также можете использовать принятые по умолчанию единицы для задания не-PSTricks'овских размеров с помощью команд

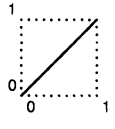
```
\pssetlength{cmd}{dim}
\psaddtolength{cmd}{dim}
```

где аргумент *cmd* — это длина, а *dim* — ее численное значение с необязательной единицей измерения. Для задания стандартных LaTeX'овских параметров длины с помощью принятых в пакете PSTricks по умолчанию единиц воспользуйтесь командами `\pssetlength` и `\psaddtolength` взамен LaTeX'овских команд `\setlength` и `\addtolength`.

Команда `\SpecialCoor` настраивает пакет PSTricks на работу с полярными координатами в форме $(r;a)$, где r — это радиус (размер), а a — угол. Вы по-прежнему можете использовать декартовы координаты.

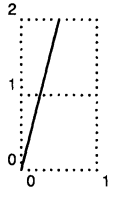
Параметр `unit` на самом деле устанавливает три параметра: `xunit`, `yunit` и `runit`. Они представляют собой принятые по умолчанию единицы измерения для координат x, координат y и всех остальных координат соответственно. Определяя их независимо друг от друга, вы можете изменять масштаб вдоль направлений x и y в декартовых координатах.

Влияние изменения принятых по умолчанию единиц измерения можно увидеть, сравнив следующие два примера (сетка осталась той же самой):




```
\usepackage {pstcol}
\begin{pspicture}(0,0)(1,1)\showgrid
\psline(0,0)(1,1)
\end{pspicture}
```

4-2-2




```
\usepackage {pstcol}
\begin{pspicture}(0,0)(1,2)\showgrid
\psset{xunit=.5cm,yunit=2cm}
\psline(0,0)(1,1)
\end{pspicture}
```

4-2-3

Сходный результат может быть получен с помощью стандартного LaTeX'овского пакета graphics и входящего в него макро \resizebox:

4-2-4

```
\usepackage {pstcol}
\resizebox{.5cm}{2cm}{%
\begin{pspicture}(1,1)
 \psline(0,0)(1,1)
\end{pspicture}
}
```

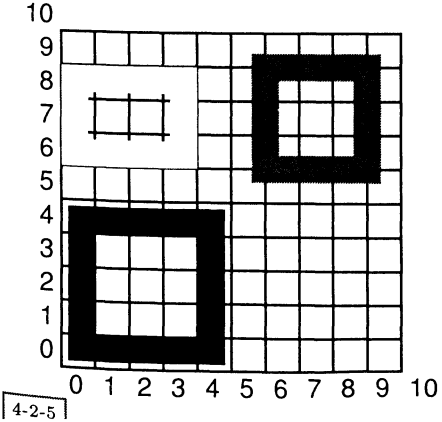
Поскольку в пакете PSTricks некоторые из принятых по умолчанию значений заданы в абсолютных величинах (а не выражаются через единицы измерения), изменение единиц измерения не оказывает на них влияния. В случае необходимости изменить масштаб *целого рисунка* надежнее сделать это с помощью команды \resizebox. Вероятно, этот способ является наилучшим при изменении размера уже готового рисунка для размещения его на странице другого формата.

Углы в полярных координатах и в других случаях должны выражаться числом, по умолчанию выраженным в градусах. Вы можете изменить единицу измерения углов с помощью команды \degrees[*num*], где величина *num* — это число таких единиц в круге. Таким образом, команду

```
\degrees[100]
```

можно использовать при создании круговой диаграммы, если исходные данные выражены в процентах. Команда \radians представляет собой краткую запись выражения \degrees[6.28319].

При рисовании объектов важно понимать, что линии могут быть проведены внутри, снаружи или накладывающимися на прямую, соединяющую две точки. Это определяется параметром dimen, который обычно разумно устанавливать равным «middle». В следующем примере показан результат рисования одного и того же квадрата с помощью трех вариантов параметра dimen:



4-2-5

```
\usepackage {pstcol}
\psset{unit=.45cm}
\begin{pspicture}(10,10)
\psset{subgriddiv=0,linewidth=.8}
\psgrid
\psset{dimen=inner}
\psframe
   (1,1)(4,4)
\psframe[dimen=middle,linecolor=red]
   (6,6)(9,9)
\psframe[dimen=outer,linecolor=yellow]
   (0,6)(4,9)
\end{pspicture}
```

4.2.4 Раскрашивание объектов

Пакет PSTricks содержит свою собственную коллекцию макро, обеспечивающих базовый набор цветов и дающих вам возможность определять имена новых цветов в терминах моделей RGB, CMYK или HSB (см. гл. 9). Однако, мы рекомендуем пользователям LaTeX'а ограничиться пакетом color, описанным в гл. 2. Стандартный пакет для работы с цветом и PSTricks можно использовать совместно, загрузив пакет pstcol, который входит в число универсальных средств для работы с цветом и графикой. Пакет pst-grad добавляет возможность плавного изменения цвета при закрашивании объектов.

Следующие цвета являются предопределенными: *black*, *darkgray*, *gray*, *lightgray*, *white*, *red*, *green*, *blue*, *cyan*, *magenta*, *yellow* (черный, темносерый, серый, светлосерый, белый, красный, зеленый, синий, голубой, пурпурный, желтый).

4.3 Графические объекты

В разд. 4.13.1 на с. 186 перечислены базовые объекты, которые можно рисовать с помощью PSTricks. За исключением команд, помещающих текст в рамку, все эти объекты в TeX'е не занимают места и при создании рисунка должны появляться только внутри окружения pspicture. Почти во всех случаях, где требуется наличие пары координат, первая из них может быть опущена и ей будет приписано значение по умолчанию $(0,0)$. Линии и незамкнутые кривые при желании могут заканчиваться различными символами. Чтобы добиться такого эффекта, используется параметр arrows (см. примеры и табл. 4.3 на с. 132).

Любой графический объект можно использовать в качестве отсекающей траектории. Команда

```
\psclip{graphics} ...\endpsclip
```

отсекает все, расположенное между \psclip и \endpsclip, что не попадает внутрь нарисованного командой *graphics* (не старайтесь быть слишком умными, пользуйтесь только простыми графическими объектами). Это иллюстрирует следующий пример:


```
\usepackage {pstcol}
\begin{pspicture}(-1,0)(2,2)
 \psclip{\psdiamond(1.5,0.2)(2.0,0.5)}
 \Huge Scary Monsters
 \endpsclip
\end{pspicture}
```

4-3-1

В PSTricks входят мощные команды позиционирования (и, при необходимости, поворота) любых объектов, включая обычный LaTeX'овский материал.

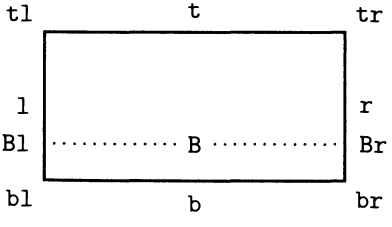
Они аналогичны базовой LaTeX'овской команде \put в графическом режиме. Наиболее распространенной командой является

\rput*[*settings*] [*refpoint*] {*angle*} (x_0, y_0) {*stuff*}

Вариант со звездочкой помещает вокруг *stuff* рамку \psframebox, которая эффективно блокирует все лежащее под ней. Материал *stuff* поворачивается на угол *angle*, если задан соответствующий аргумент. Звездочка *, предшествующая углу *angle*, отменяет все вращения, заданные во внешних обращениях к команде \rput. Это позволяет размещать текстовые метки в нужной ориентации. Поскольку повороты часто задаются с шагом, равным 90°, вы можете воспользоваться сокращенными обозначениями, приведенными в табл. 4.2 ниже.

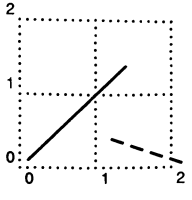
Аргумент *refpoint* описывает точку привязки материала *stuff*, и именно эта точка привязки помещается в точку (x, y). По умолчанию ею служит центр бокса. Это можно изменить, указав значение параметра *refpoint* равным одно- или двухбуквенной аббревиатуре из приведенных на следующей диаграмме. На ней показано положение точки привязки, представленное разными комбинациями (пунктиром указана базовая линия).

По горизонтали		*По вертикали*	
l	Left (слева)	t	Top (сверху)
r	Right (справа)	b	Bottom (снизу)
		B	Baseline (базовая линия)



По большей части те же самые соглашения относительно задания поворота боксов приняты и в пакете graphicx (см. рис. 2.2 на с. 68).

Важно иметь в виду, что внутри аргумента система координат локально изменяется таким образом, что точка (0,0) соответствует координате команды \rput, что показывает следующий пример:



4-3-2

```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2.2,2)\showgrid
\rput(.1,.1){\psline(0,0)(1.3,1.3)}
\psset{linestyle=dashed}
\rput{-20}(1.2,.4){\psline(0,0)(1,0)}
\end{pspicture}
```

Таблица 4.2. Сокращенные обозначения для часто используемых углов в PSTricks.

Буква		*градусы*	*Буква*		*градусы*
U	*Up* (*Вверх*)	0	N	*North* (*Север*)	*0
L	*Left* (*Налево*)	90	W	*West* (*Запад*)	*90
D	*Down* (*Вниз*)	180	S	*South* (*Юг*)	*180
R	*Right* (*Направо*)	270	E	*East* (*Восток*)	*270
r	*right* (*направо*)	0	ur	*up-right* (*вверх-направо*)	45
u	*up* (*вверх*)	90	ul	*up-left* (*вверх-налево*)	135
l	*left* (*налево*)	180	dl	*down-left* (*вниз-налево*)	225
d	*down* (*вниз*)	270	dr	*down-right* (*вниз-направо*)	315

Поскольку очень часто возникает потребность *поместить* метку рядом с каким-нибудь объектом, существует вариант команды \rput, называющийся \uput:

\uput*[*settings*] {*labelsep*} [*refangle*] {*rotation*} (*x*,*y*) {*stuff*}

Эта команда располагает материал *stuff* в направлении *refangle* на расстоянии *labelsep* от точки (x,y). Если аргумент *labelsep* не указан, то используется текущее значение параметра \labelsep (по умолчанию равное 5pt в стандартном LaTeX'e). Так как углы часто указываются с шагом 45°, для наиболее распространенных случаев определены сокращенные буквенные обозначения (см. табл. 4.2). Наконец, существует специальный макро для нанесения обведенных окружностью меток за один шаг:

\cput*[*settings*] {*angle*} (*x*,*y*) {*stuff*}

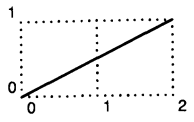
Эта команда приводит к тому же результату, что \rput и \pscirclebox вместе. Точкой привязки всегда является центр бокса, а необязательные настройки *settings* можно использовать для изменения того, как рисуется окружность.

4.4 Примеры основных графических объектов

В следующих примерах демонстрируются некоторые из строительных блоков пакета PSTricks и применение графических параметров. Необходимо помнить, что все эти простые объекты не занимают места. Содержащее их окружение pspicture указывает TeX'у, сколько места нужно выделить, но внутри самого окружения рисование осуществляется на основе координат. Некоторые из этих рисунков воспроизведены на цветной вклейке II.

Начнем с простой линии. Начальной точкой по умолчанию является $(0,0)$, значения всех параметров выбраны по умолчанию.

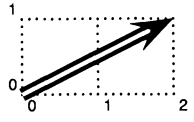
4-4-1




```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,1)\showgrid
  \psline(2,1)
\end{pspicture}
```

У команды \psline есть необязательный параметр, позволяющий завершать линии символами типа стрелок (существующие возможности перечислены в табл. 4.3 ниже), а внешний вид линии можно изменять с помощью параметров linewidth, doubleline и doublesep:

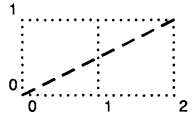
4-4-2




```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,1)\showgrid
  \psline[linewidth=0.6mm,doubleline=true,
          doublesep=0.5mm]{->}(2,1)
\end{pspicture}
```

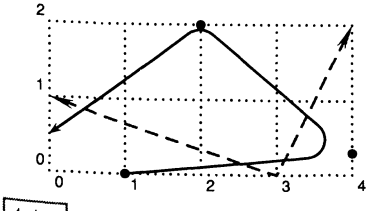
Помимо этого внешний вид линии можно изменять с помощью параметра linestyle (допустимые значения этого параметра см. в табл. 4.5 на с. 188):

4-4-3




```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,1)\showgrid
  \psline[linestyle=dashed](2,1)
\end{pspicture}
```

Как видно из следующего примера, команде \psline можно передать в виде аргументов сколько угодно координат. На рисунке нарисованы две линии. У одной линии мы изменили цвет, длину стрелки-указателя, а стиль самой линии изменили на пунктирный. У другой для изменения внешнего вида поворотов линии около точек мы использовали параметр linearc. Параметр showpoints указывает пакету PSTricks печатать заданные точки, хотя сама линия не обязательно проходит через них.



4-4-4

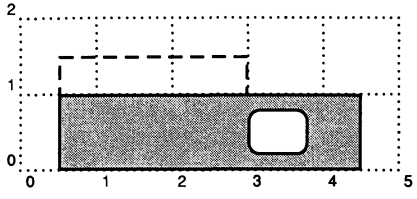
```
\usepackage {pstcol}
\begin{pspicture}(0,0)(4,2)\showgrid
 \psline[linearc=0.25,showpoints=true]%
    {->}(1,0)(4,0.3)(2,2)(0,0.5)
  \psline[linestyle=dashed,
    linecolor=blue,
    arrowlength=3]{<->}(0,1)(3,0)(4,2)
\end{pspicture}
```

Таблица 4.3. Символы, используемые в PSTricks для завершения линий.

Значения и примеры						
-						Ничего
<->		->		<-		Стрелки-указатели
>-<		-<		>-		Обратные стрелки-указатели
<<->>		->>		<<-		Двойные стрелки-указатели
>>-<<		-<<		>>-		Двойные обратные стрелки-указатели
|-|		-|		|-		Т-образные концы, выровненные по краю линии
|*-|*		-|*		|*-		Т-образные концы, отцентрированные относительно краев линии
[-]		-]		[-		Квадратные скобки
(-)		-)		(-		Круглые скобки
o-o		-o		o-		Окружности, отцентрированные относительно краев линии
-		-*		*-		Диски, отцентрированные относительно краев линии
oo-oo		-oo		oo-		Окружности, выровненные по краю линии
-		-**		**-		Диски, выровненные по краю линии
c-c		-c		c-		Удлиненные с округлыми концами
cc-cc		-cc		cc-		Выровненные по краю с округлыми концами
C-C		-C		C-		Удлиненные с квадратными концами
|<->|		->|		|<-		Т-образные концы и стрелки-указатели
|<*->|*		->|*		|<*-		Т-образные концы и стрелки-указатели, выровненные по краю

Простейшим способом нарисовать бокс является использование команды \psframe. В следующем примере показано, как проделать «дыру» в сплошной фигуре путем наложения сплошной белой фигуры меньшего размера:

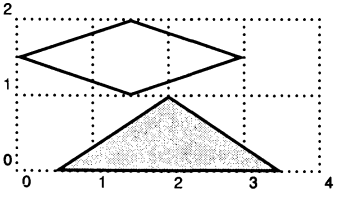
4-4-5




```
\usepackage {pstcol}
\definecolor{pink}{rgb}{1, .75, .8}
\begin{pspicture}(0,0)(4.5,2)
  \showgrid
  \psframe[linestyle=dashed]
    (0.5,0.5)(3,1.5)
  \psframe[fillstyle=solid,
     fillcolor=pink]
    (.5,0)(4.5,1)
  \psframe
    [fillstyle=solid,
     fillcolor=white,framearc=0.5]
    (3,0.2)(3.8,0.8)
\end{pspicture}
```

PSTricks содержит универсальный макро \pspolygon, но некоторые наиболее распространенные формы предопределены (см. разд. 4.13.1 на с. 186):

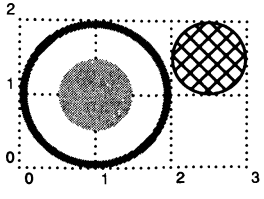
4-4-6




```
\usepackage {pstcol}
\begin{pspicture}(0,0)(4,2)\showgrid
  \psdiamond(1.5,1.5)(1.5,0.5)
  \pstriangle[fillstyle=solid,
     fillcolor=yellow](2,0)(3,1)
\end{pspicture}
```

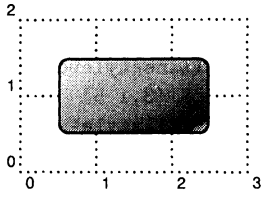
Объекты можно заполнять сплошным цветом или каким-нибудь узором:

4-4-7




```
\usepackage {pstcol}
\definecolor{lightblue}{rgb}
                {.68, .85, .9}
\begin{pspicture}(0,0)(3,2)\showgrid
  \pscircle[linewidth=1mm,
     linecolor=green](1,1){1}
  \pscircle[linestyle=none,
     fillstyle=solid,
     fillcolor=lightblue](1,1){0.5}
  \pscircle[fillstyle=crosshatch]
     (2.5,1.5){0.5}
\end{pspicture}
```

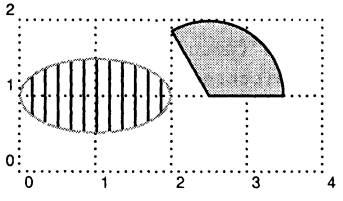
Загрузка необязательного пакета pst-grad расширяет возможности заполнения объектов цветом. Если заданы имена двух цветов, то цвет заполнения плавно меняется внутри объекта, переходя из одного цвета в другой:




```
\usepackage {pstcol,pst-grad}
\begin{pspicture}(0,0)(3,2)\showgrid
 \psframe[framearc=0.25,
   fillstyle=gradient,
   gradangle=30,
   gradbegin=yellow,
   gradend=red]
   (0.5,0.5)(2.5,1.5)
\end{pspicture}
```

4-4-8

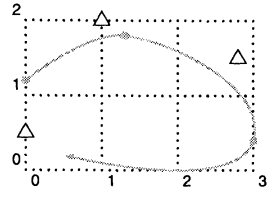
В следующем примере продемонстрировано большее разнообразие стилей закрашивания, включающее изменение угла и цвета линий для штрихования:




```
\usepackage {pstcol}
\begin{pspicture}(0,0)(4,2)\showgrid
  \psellipse[linecolor=green,
             fillstyle=vlines,
             hatchangle=0]
             (1,1)(1,0.5)
  \pswedge[fillstyle=solid,
           fillcolor=lightgray]
          (2.5,1){1}{0}{120}
\end{pspicture}
```

4-4-9

Существует множество команд для объединения списков координат. Следующий пример иллюстрирует две из них: команда \psdots просто помещает в каждую точку указанный символ (широкий выбор стилей изображения точек показан в табл. 4.4), а команда \pscurve проводит кривую через указанные в ней точки (здесь снова использована команда showpoints, чтобы были видны координаты).



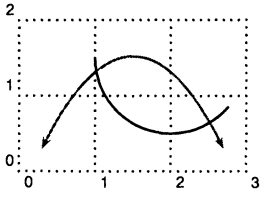

```
\usepackage {pstcol}
\begin{pspicture}(0,0)(3,2)\showgrid
  \psdots[linecolor=blue,dotstyle=triangle,
          dotscale=2](0,0.5)(1,2)(2.8,1.5)
  \pscurve[linecolor=cyan,showpoints=true]
          {->}%
          (0,1.2)(1.3,1.8)(3,0.4)(0.5,0.2)
\end{pspicture}
```

4-4-10

Таблица 4.4. Стили изображения точек в пакете PSTricks.

Стиль	*Результат*	*Стиль*	*Результат*
*	● ● ● ● ● ● ●	oplus	⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕
+	+ + + + + + +	otimes	⊗ ⊗ ⊗ ⊗ ⊗ ⊗ ⊗
|	| | | | | | |	pentagon*	⬟ ⬟ ⬟ ⬟ ⬟ ⬟ ⬟
o	○ ○ ○ ○ ○ ○ ○	pentagon	⬠ ⬠ ⬠ ⬠ ⬠ ⬠ ⬠
x	× × × × × × ×	square*	■ ■ ■ ■ ■ ■ ■
asterisk	* * * * * * *	square	□ □ □ □ □ □ □
diamond*	♦ ♦ ♦ ♦ ♦ ♦ ♦	triangle*	▲ ▲ ▲ ▲ ▲ ▲ ▲
diamond	◊ ◊ ◊ ◊ ◊ ◊ ◊	triangle	△ △ △ △ △ △ △

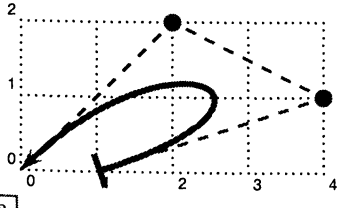
Команда \parabola рисует параболу, начинающуюся в точке, описанной одной парой координат, максимум которой расположен в точке, описанной другой парой координат, тогда как команда \psarc не требует задания координат, а рисует часть окружности, для которой указаны центр, радиус и начальный и конечный углы:




```
\usepackage {pstcol}
\begin{pspicture}(0,0)(3,2)\showgrid
  \parabola[linecolor=red]{<->}%
           (0.3,0.3)(1.5,1.5)
  \psarc(2,1.5){1}{180}{320}
\end{pspicture}
```

4-4-11

Тем, кто предпочитает тонкую настройку формы вычерчиваемых кривых, макро \psbezier предоставляет в распоряжение кривую Безье, реализованную на PostScript'е. Мы просто задаем четыре контрольные точки (отображаемые в следующем примере с помощью ключа showpoints), и PostScript строит по ним гладкую кривую:



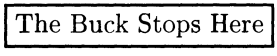

```
\usepackage {pstcol}
\begin{pspicture}(0,0)(4,2)\showgrid
  \psbezier[linewidth=0.8mm,linecolor=red,
            showpoints=true]{|->}%
           (1,0)(4,1)(2,2)(0,0)
\end{pspicture}
```

4-4-12

4.5 Соединение текста и графики

Ситуация совершенно меняется при наличии текста. Теперь размер объектов определяется размером содержащегося в них текста. Теперь TeX начинает заботиться об отведенном пространстве, поэтому последовательно идущие объекты располагаются так, как если бы они были буквами, а координаты больше не используются. Здесь также некоторые из рисунков воспроизведены на цветной вклейке II.

Широко применяемой командой является \psframebox, которая окружает текст рамкой:




```
\usepackage {pstricks}
\psframebox{The Buck Stops Here}
```

4-5-1

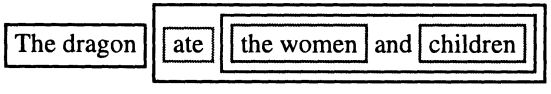
Когда белый текст на черном или цветном фоне помещается в рамку, обычно необходимо использовать полужирный шрифт:


```
\usepackage {pstricks}
\psframebox[fillstyle=solid,
   fillcolor=black]
   {\bfseries\color{white}\LARGE
            Beware of The Dog}
```

4-5-2

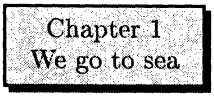
Как обычно, возможно вложение команд \psframebox:

```
\usepackage {pstcol}
  \psframebox{The dragon}
  \psframebox{\psframebox[linecolor=green]{ate}
     \psframebox[linecolor=blue]
       {\psframebox[linecolor=red]{the women}  and
        \psframebox[linecolor=red]{children}%
       }}
```



4-5-3

Несколько вариантов команды \psframebox реализуют часто используемые эффекты. Одним из них является \psshadowbox:




```
\usepackage {pstcol}
  \psshadowbox[fillstyle=solid,
               fillcolor=yellow]
     {\color{red}
     \begin{tabular}{c}
       Chapter 1\\We go to sea
     \end{tabular}}
```

4-5-4

Одни из этих эффектов реализуются с помощью специальных команд, другие — с помощью специальных параметров, передаваемых командам типа \psframebox. Охватывающая текст двойная рамка в форме окружности получается с помощью параметра doubleline:

4-5-5


```
\usepackage {pstricks}
  \pscirclebox[doubleline=true]
      {\bfseries STOP!}
```

но для создания двойной прямоугольной рамки есть специальная команда \psdblframebox:

4-5-6


```
\usepackage {pstcol}
  \psdblframebox[linecolor=green,
    linestyle=dotted,linewidth=3pt,
    dotstyle=*]
   {\color{red}All Hail Caesar!}
```

Другими доступными формами рамок являются ромбовидная (\psdiabox), треугольная (\pstribox) и овальная (\psovalbox):

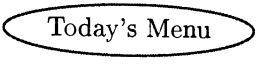
4-5-7


```
\usepackage {pstricks}
  \psdiabox{\sffamily No Parking}
```

4-5-8


```
\usepackage {pstcol,pst-grad}
 \pstribox[shadow=true,fillstyle=gradient,
   gradbegin=yellow,gradend=red]
  {\color{white}\Huge
   $\frac{\alpha}{\Omega}$}
```

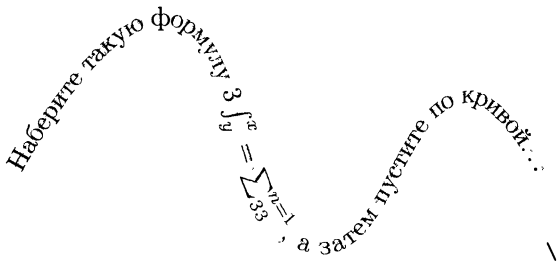
4-5-9




```
\usepackage {pstcol,pst-grad}
  \psovalbox[linecolor=red]
    {\color{blue}Today's Menu}
```

Приверженцы PostScript'а прекрасно знают, что отношение к тексту в PostScript'е точно такое же, как и к любому другому графическому объекту (это было одной из революционных идей, воплощенных в данном языке), и что текстом можно манипулировать. Следующие примеры демонстрируют работу

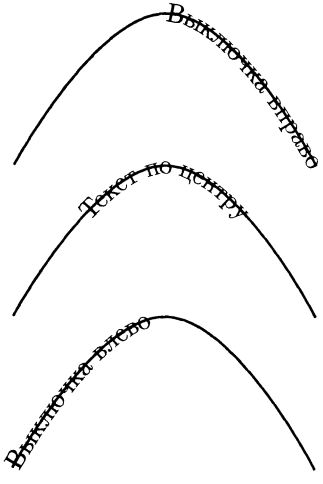
макро \pstextpath, имеющего два параметра: графический объект и некоторый текст. Этот текст размещается в соответствии с графическим объектом (в данном случае задаваемым командой \pscurve). Допускается лишь простой текст, но он может содержать математику[1]




```
\usepackage {pstcol,pst-text}
\usepackage [cp866]{inputenc}
\usepackage [LCY]{fontenc}
\begin{pspicture}(-4,-3.2)(3,0.2)
 \psset{linecolor=lightgray}
 \pstextpath
  {\pscurve(-4,-2)(-2,0)(0,-3)
           (2,-1)(3,-2)}
  {\color{blue}
  Наберите такую формулу
  $3 \int^x_y = \sum^{n=1}_{33}$,
  а затем пустите
  по кривой\ldots}
\end{pspicture}
```

4-5-10

Используя необязательный аргумент команды \pstextpath, текст может быть смещен относительно концов траектории:



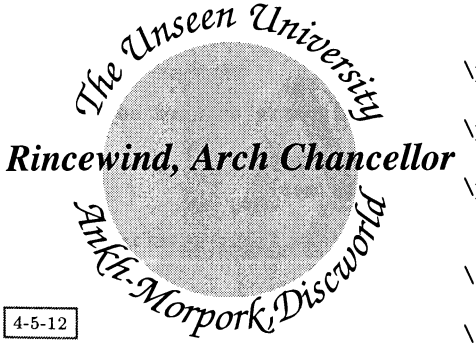

```
\usepackage {pstricks,pst-text}
\usepackage [cp866]{inputenc}
\usepackage [LCY]{fontenc}
\begin{pspicture}(0,0)(4.2,6.2)
\pstextpath[l]
 {\pscurve(0,0)(2,2)(4,0)}
 {Выключка влево}
\pstextpath[c]
 {\pscurve(0,2)(2,4)(4,2)}
 {Текст по центру}
\pstextpath[r]
 {\pscurve(0,4)(2,6)(4,4)}
 {Выключка вправо}
\end{pspicture}
```

4-5-11

В приведенных выше примерах мы задали, чтобы направляющая кривая была явно показана. В приведенных ниже более сложных примерах она сделана невидимой. В следующем примере текст помещен таким образом, чтобы

[1] За исключением таких структур, как квадратные корни, которые используют TeX'овские линейки для создания верхней линии. Такие структуры не смогут быть правильно обработаны.

соответствовать верхней и нижней частям окружности (сами полуокружности определены с помощью команд \psarc и \psarcn). В центре помещена закрашенная окружность без границы (поскольку для отключения рисования линий в последующих объектах была использована команда \psset), а для набора заголовка применена команда \rput. На самом деле на момент написания книги г-н Rincewind еще не был назначен на должность Arch Chancellor, но кто знает?



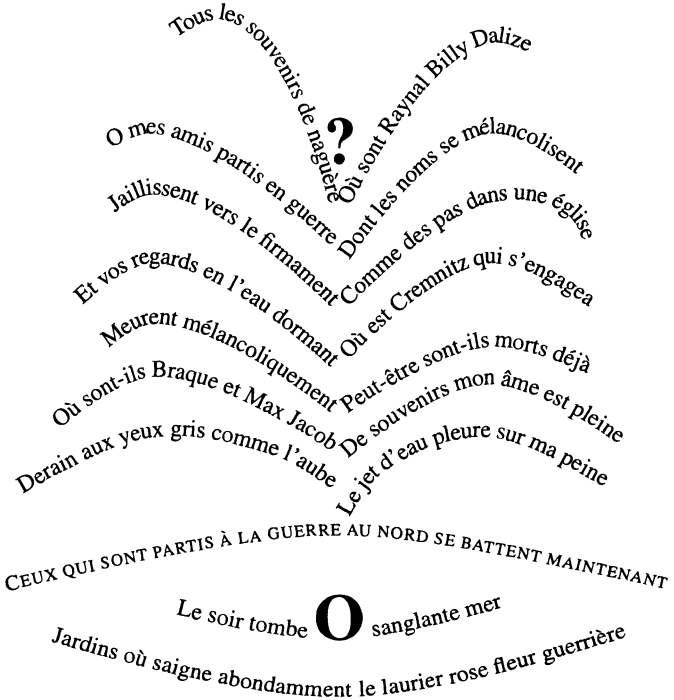
4-5-12

```
\usepackage {pstcol,pst-text,pst-grad}
\definecolor{pink}{rgb}{1, .75, .8}
\begin{pspicture}(-3,-2.2)(3,2.2)
\psset{linestyle=none}
\newcommand{\curly}[1]{%
 {\fontfamily{pzc}%
 \fontsize{17}{17}\itshape#1}}
\pstextpath[c]{\psarcn(0,0){2}{180}{0}}
 {\curly{The Unseen University}}
\pstextpath[c]{\psarc(0,0){2}{180}{0}}
 {\curly{Ankh-Morpork,Discworld}}
\pscircle[fillstyle=gradient,
 gradangle=45,gradbegin=pink,
 gradend=yellow](0,0){1.7}
\rput[B](0,0){{\Large\itshape\bfseries
 Rincewind, Arch Chancellor}}
\end{pspicture}
```

В качестве более сложного примера текста, набранного вдоль замысловатых траекторий, приведем отрывок стихотворения Гийома Аполлинера *La colombe poignardée et le jet d'eau*, набранный Дени Жиру в соответствии с внешним видом оригинала. Координаты каждой кривой были получены путем нанесения текста на миллиметровую бумагу и выбора некоторых точек, расположенных на каждой кривой. Автоматизация и параметризация этого процесса для произвольного текста была бы нетривиальной (хотя и осуществимой) задачей.

```
\usepackage[latin1]{inputenc}
\usepackage{pstricks,pst-text}
\begin{pspicture}(9,13.5)
  \small\psset{linestyle=none}
  \pstextpath[r]{\pscurve(1.5,8)(3,9)(4.45,6.5)}
       {Tous les souvenirs de naguère}
  \pstextpath[r]{\pscurve(1,7)(2,7.5)(4.45,6)}
       {O mes amis partis en guerre}
    \pstextpath[r]{\pscurve(1,6)(2,6.7)(4.45,5.2)}
       {Jaillissent vers le firmament}
    \pstextpath[r]{\pscurve(0.8,5)(2,5.8)(4.45,4.4)}
       {Et vos regards en l'eau dormant}
    \pstextpath[r]{\pscurve(1,4.3)(2,5)(4.45,3.8)}
       {Meurent mélancoliquement}
```

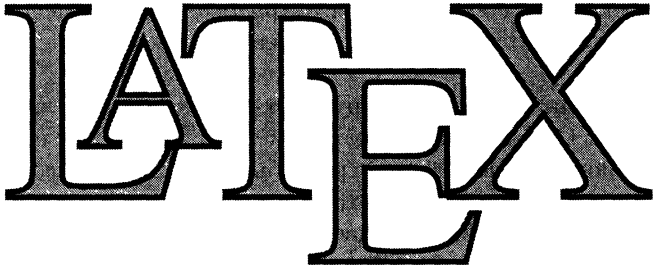
```
    \pstextpath[r]{\pscurve(0.3,3.2)(2,4.3)(4.45,3.3)}
       {Où sont-ils Braque et Max Jacob}
    \pstextpath[r]{\pscurve(0.2,2.7)(2,3.5)(3.8,3.2)(4.45,2.8)}
       {Derain aux yeux gris comme l'aube}
    \rput(4.5,7.3){\bfseries\Huge ?}
    \pstextpath[l]{\pscurve(4.55,6.5)(6.5,8.7)(7.8,8)}
       {Où sont Raynal Billy Dalize}
    \pstextpath[l]{\pscurve(4.55,5.8)(6.5,7.5)(8.3,6.4)}
       {Dont les noms se mélancolisent}
    \pstextpath[l]{\pscurve(4.55,5.2)(6.5,6.6)(7.6,6.4)(8.2,5.2)(8.5,5.4)}
       {Comme des pas dans une église}
    \pstextpath[l]{\pscurve(4.55,4.5)(6.5,5.8)(8.5,4.7)}
       {Où est Cremnitz qui s'engagea}
    \pstextpath[l]{\pscurve(4.55,3.8)(6.5,4.7)(8.5,4)}
       {Peut-être sont-ils morts déjà}
    \pstextpath[l]{\pscurve(4.55,3.2)(6.5,4.2)(9,2.8)}
       {De souvenirs mon âme est pleine}
    \pstextpath[l]{\pscurve(4.55,2.4)(5,3)(6.5,3.5)(8.8,2.3)}
       {Le jet d'eau pleure sur ma peine}
    \pstextpath[c]{\pscurve(-0.1,1.4)(4.5,2.2)(9,1.4)}
{{\footnotesize\scshape Ceux qui sont partis à la guerre au nord se battent
       maintenant}}
    \pstextpath[c]{\pscurve(2.1,1.2)(4.5,0.8)(6.9,1.3)}
       {Le soir tombe {\bfseries\Huge O} sanglante mer}
    \pstextpath[c]{\pscurve(0,1)(4.5,0)(9,1)}
       {Jardins où saigne abondamment le laurier rose fleur guerrière}
  \end{pspicture}
```



4-5-13

Кроме того существует возможность набирать текст так, как если бы он был графикой, указывая заполнение, цвет и характеристики линий. Для этой цели используется макро \pscharpath. Ему присущи некоторые ограничения (например, его нельзя использовать совместно с \pstextpath), но все же он оказывается в некоторых случаях полезным инструментом. Следующий пример также показан на цветной вклейке IIIb:

```
\usepackage {pstcol,pst-char}
\definecolor{lightblue}{rgb}{0, .93, .93}
\begin{pspicture}(0,-1)(8.7,2.5)
\fontfamily{ptm}\selectfont
\pscharpath[fillstyle=solid,linecolor=red,
      fillcolor=lightblue,linewidth=2pt]
{\fontsize{1.5in}{1.5in}\selectfont \LaTeX}
\end{pspicture}
```



4-5-14

Любители четкого стиля публикаций Dorling Kindersley могут захотеть применить следующий эффект (также изображенный на цветной вклейке IIIc). Для быстрой загрузки букв большого размера, не заботясь об изменении настроек базовой линии и других, использована команда \DeclareFixedFont. Семейство шрифтов zmn — это Adobe Multiple Master Minion, основной шрифт, использованный для набора текста в данной книге![1]

```
\usepackage {pstcol,pst-char}
\begin{pspicture}(-5,-3)(5,1.2)
 \psset{fillstyle=solid,shadow=true,shadowangle=0}
 \DeclareFixedFont{\babyfont}{T1}{ptm}{m}{n}{3cm}
 \DeclareFixedFont{\wordfont}{T1}{ptm}{m}{n}{2cm}
 \newcommand{\Cc}[2]{\pscharpath[fillcolor=#1]{#2}}
 \bfseries
 \rput(0,0){\babyfont
 \Cc{red}B\Cc{green}A\Cc{yellow}B%
 \Cc{red}{Y'}\Cc{blue}S}
 \rput(0,-2){{\wordfont
```

[1] Речь идет об английском оригинале. — *Прим. ред.*

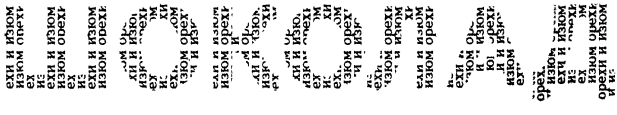
```
 \Cc{blue}{WORLD}}}
\end{pspicture}
```

4-5-15

Установка собственного цвета для каждой буквы обладает тем неприятным побочным эффектом, что кернинг и лигатуры оказываются недоступными, но вы вряд ли будете использовать этот эффект за исключением очень редких случаев.

Третьей из числа команд, оперирующих с текстом как с графикой, является \pscharclip. Она получает некоторый текст в качестве параметра и заканчивается командой \endpscharclip. Любые объекты, нарисованные внутри этой группы, ограничиваются так, чтобы иметь форму заданных букв (полученный рисунок также показан на цветной вклейке IIId).

```
\usepackage {pstcol,pst-char,ifthen}
\usepackage [cp866]{inputenc}
\usepackage [russian]{babel}
\usepackage {pscyr}
\DeclareFixedFont{\bigsf}{T2A}{ftx}{b}{n}{1.6cm}
\DeclareFixedFont{\tinyrm}{T2A}{fha}{m}{n}{2mm}
\newcounter{myN}
\setcounter{myN}{300}
\begin{pspicture}(0,0)(11,1.4)
\pscharclip[linecolor=yellow,fillstyle=solid,fillcolor=yellow]
          {\rput[bl](0,0){\bigsf ШОКОЛАД}}
\rput[t]{90}(0,0)
  {\begin{minipage}{4cm}\offinterlineskip\raggedright\tinyrm\color{black}%
     \whiledo{\value{myN}>0}{\addtocounter{myN}{-1}орехи и изюм }%
   \end{minipage}}%
\endpscharclip
\end{pspicture}
```



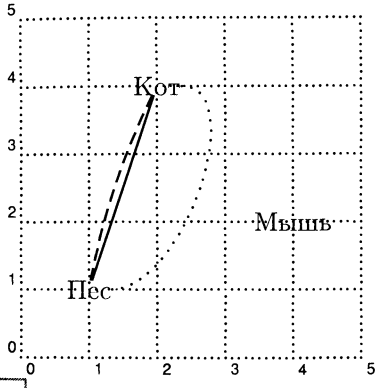
4-5-16

4.6 Вершины, их соединения и деревья

В пакете PSTricks есть сложные макро для задания именованных вершин и соединения их сложным образом с добавлением меток. Поддержка этих возможностей осуществляется при загрузке дополнительного пакета pst-node. Вершины можно создавать тремя способами:

- размещая их в точках с произвольными координатами;
- размещая их на регулярной сетке или внутри матрицы;
- используя макро высокого уровня для создания деревьев.

Некоторое представление о том, что мы собираемся создавать, дает следующий пример:



4-6-1

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(0,0)(5,5)\showgrid
\rput(1,1){\rnode{A}{Пес}}
\rput(2,4){\rnode{B}{Кот}}
\rput(4,2){Мышь}
\ncline{A}{B}
\nccurve[linestyle=dotted]{A}{B}
\ncarc[linestyle=dashed]{A}{B}
\end{pspicture}
```

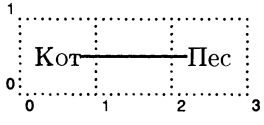
Каждой вершине присваивается символическое имя, которое затем используется для соединения вершин (прямыми линиями, кривыми и т. д.) и присвоения меток. Тот факт, что вершины могут встречаться *где угодно*, даже внутри текста, дает возможность использовать их для достижения неожиданных эффектов, таких, как слияние двух слов. Рассматривая соединительные элементы, необходимо различать положение вершины, т. е. точку привязки вершины, на которую они указывают, и реальную протяженность этого элемента. Большинство средств создания вершин помещает вокруг них невидимый бокс, определяющий границы окончания соединительных линий. В разд. 4.13.2 на с. 191 перечислены все команды для создания вершин и их соединения (вместе с их синтаксисом), а в разд. 4.13.3 на с. 193 — все команды для присвоения меток вершинам и соединительным элементам. Поскольку присвоение меток соединяющим вершины элементам является часто встречающейся задачей, у всех этих команд есть сокращенные обозначения наподобие `\naput`. Если параметр `shortput` установлен равным `nab`, то вместо `\naput` может быть использован символ `^`, а вместо `\nbput` — символ `_`. Если он установлен равным `tab`, то символ `^` заменяет команду `\taput`, символ `_` — команду `\tbput`, символ `<` — команду `\tlput`, а символ `>` — команду `\trput`. В табл. 4.6 на с. 194 перечислены дополнительные графические параметры, которые можно задавать для соединяющих вершины элементов.

Есть несколько важных соображений, которые необходимо иметь в виду, глядя на мириады команд, определяющих вершины и соединительные элементы:

- соединяя две вершины чем-либо, напоминающим кривую (в противоположность прямой линии), соединяющие элементы по умолчанию помещаются справа от объекта (под углом 0 градусов). Если два бокса расположены бок о бок, то необходимо задать параметр `angleB` равным 180, чтобы соединяющий элемент входил во второй бокс слева. Сначала это может показаться несколько обременительным, но это сделано ради гибкости системы;
- все команды, снабжающие метками соединительные элементы, помещают их в некоторой точке, определяемой долей протяженности этих элементов, но они различают расстояние между вершинами и длину самой линии (которая зависит от размера соединяемых вершин). Первая величина бывает полезна в таких конструкциях как матрицы, где позиции меток должны быть постоянными вне зависимости от размера вершин;
- соединяющие вершины элементы рисуются не LaTeX'ом, а на уровне PostScript'а. Это значит, что LaTeX не всегда точно знает, сколько места занимает определенный объект. В частности, искривленные соединительные элементы могут выходить за пределы области, отведенной для них LaTeX'ом — откорректируйте вручную.

Мы продолжим обзор, рассмотрев простые примеры использования большинства команд, создающих вершины и соединительные элементы, а потом изучим более сложный пример. В следующем разделе, где описываются пакеты для работы с матрицами и деревьями, есть более «жизненные» примеры и, в частности, рассматривается присвоение меток соединительным элементам.

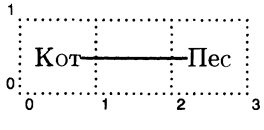
Если в качестве вершины вам нужен текст, то проще всего использовать команду `\rnode`. Простейшей формой соединительной линии является `\ncline`:




```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\rnode{A}{Кот}}
\rput(2.5,.5){\rnode{B}{Пес}}
\ncline{A}{B}
\end{pspicture}
```

4-6-2

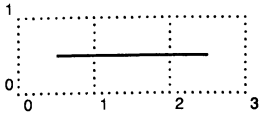
Команда `\Rnode` похожа на предыдущую, но соединительные элементы соединяют не окружающие текст боксы, а базовые линии самого текста:




```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\Rnode{A}{Кот}}
\rput(2.5,.5){\Rnode{B}{Пес}}
\ncline{A}{B}
\end{pspicture}
```

4-6-3

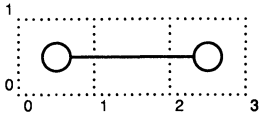
Команда \pnode просто создает точку для привязки соединительного элемента:



4-6-4

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\pnode(.5,.5){A}
\pnode(2.5,.5){B}
\ncline{A}{B}
\end{pspicture}
```

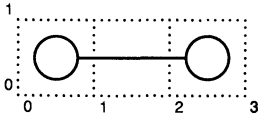
Простые вершины в форме окружности можно создавать с помощью команды \cnode, которой в качестве параметра требуется указать радиус:



4-6-5

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\cnode(.5,.5){.2}{A}
\cnode(2.5,.5){.2}{B}
\ncline{A}{B}
\end{pspicture}
```

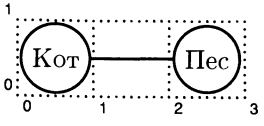
Команде \Cnode указывать радиус в виде параметра не нужно, вместо этого она использует значение глобального параметра radius:



4-6-6

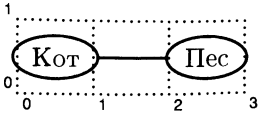
```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\psset{radius=.3}
\Cnode(.5,.5){A}
\Cnode(2.5,.5){B}
\ncline{A}{B}
\end{pspicture}
```

Используемые в \rput команды \circlenode, \ovalnode, \dianode и \trinode позволяют при описании вершины объединять текст и рамки определенного вида:



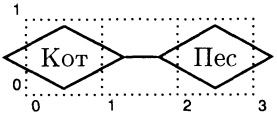
4-6-7

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\circlenode{A}{Кот}}
\rput(2.5,.5){\circlenode{B}{Пес}}
\ncline{A}{B}
\end{pspicture}
```



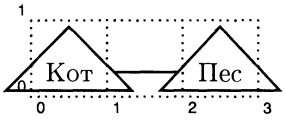
4-6-8

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\ovalnode{A}{Кот}}
\rput(2.5,.5){\ovalnode{B}{Пес}}
\ncline{A}{B}
\end{pspicture}
```




```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\dianode{A}{Кот}}
\rput(2.5,.5){\dianode{B}{Пес}}
\ncline{A}{B}
\end{pspicture}
```

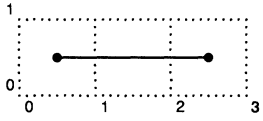
4-6-9




```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\trinode{A}{Кот}}
\rput(2.5,.5){\trinode{B}{Пес}}
\ncline{A}{B}
\end{pspicture}
```

4-6-10

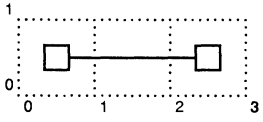
Команда \dotnode похожа на \pnode, но задает закрашенные окружности небольшого радиуса:




```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\dotnode(.5,.5){A}
\dotnode(2.5,.5){B}
\ncline{A}{B}
\end{pspicture}
```

4-6-11

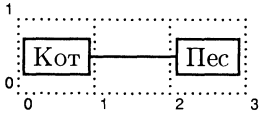
Как вы уже, возможно, начали догадываться, многие введенные ранее в этой главе примитивные команды обладают эквивалентами, создающими вершины. Таким аналогом для \psframebox является \fnode:




```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\fnode(.5,.5){A}
\fnode(2.5,.5){B}
\ncline{A}{B}
\end{pspicture}
```

4-6-12

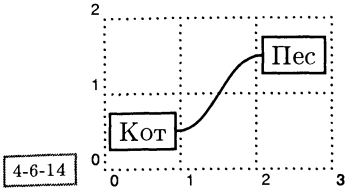
но вы также можете использовать непосредственно \psframebox внутри \rnode:




```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Кот}}}
\rput(2.5,.5){\rnode{B}{\psframebox{Пес}}}
\ncline{A}{B}
\end{pspicture}
```

4-6-13

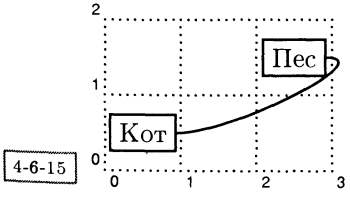
Теперь давайте посмотрим на некоторые элементы, соединяющие вершины. Применявшиеся до сих пор прямые линии вели себя очень просто, соединяя две вершины кратчайшей траекторией. При использовании кривых необходимо думать об угле, под которым они подходят к вершине, как мы уже отмечали в начале этого раздела. Таким образом, в следующем примере нужно установить ключ `angleB` равным 180, чтобы кривая (`\nccurve`) подошла к вершине «Пес» не справа, а слева:



4-6-14

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.2)(3,2)\showgrid
\rput(.5,.5)
     {\rnode{A}{\psframebox{Кот}}}
\rput(2.5,1.5)
     {\rnode{B}{\psframebox{Пес}}}
\nccurve[angleB=180]{A}{B}
\end{pspicture}
```

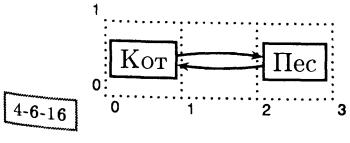
Если опустить этот параметр, то результат будет, пожалуй, не тем, который ожидался:



4-6-15

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.2)(3,2)\showgrid
\rput(.5,.5)
     {\rnode{A}{\psframebox{Кот}}}
\rput(2.5,1.5)
     {\rnode{B}{\psframebox{Пес}}}
\nccurve{A}{B}
\end{pspicture}
```

Элемент, порождаемый командой `\ncarc`, не нуждается в подобной проверке, так как команда сама заботится об этом:

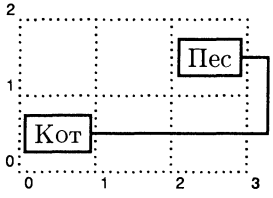


4-6-16

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.2,-.5)(3,1)\showgrid
\rput(.5,.5)
     {\rnode{A}{\psframebox{Кот}}}
\rput(2.5,.5)
     {\rnode{B}{\psframebox{Пес}}}
\ncarc{->}{A}{B}
\ncarc{->}{B}{A}
\end{pspicture}
```

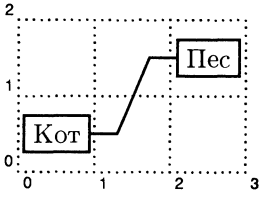
Существует шесть составных соединительных элементов `\ncbar`, `\ncdiag`, `\ncdiagg`, `\ncangle`, `\ncangles` и `\ncloop`, которые *сами* заботятся о значени-

ях параметров angleA и angleB и соединяют вершины последовательностью отрезков:



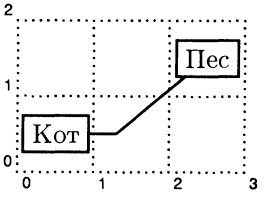

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.5,0)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Кот}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Пес}}}
\ncbar{A}{B}
\end{pspicture}
```

4-6-17



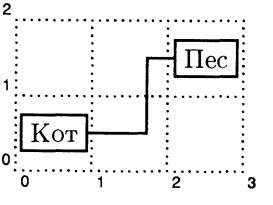

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.5,0)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Кот}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Пес}}}
\ncdiag[angleB=180]{A}{B}
\end{pspicture}
```

4-6-18



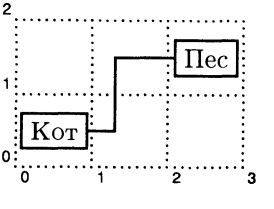

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.5,0)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Кот}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Пес}}}
\ncdiagg[angleB=180]{A}{B}
\end{pspicture}
```

4-6-19



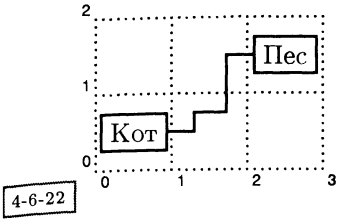

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.5,0)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Кот}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Пес}}}
\ncangle[angleB=180]{A}{B}
\end{pspicture}
```

4-6-20




```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.5,0)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Кот}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Пес}}}
\ncangles[angleB=180]{A}{B}
\end{pspicture}
```

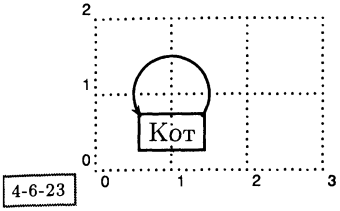
4-6-21



4-6-22

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.5,0)(3,2)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Кот}}}
\rput(2.5,1.5){\rnode{B}{\psframebox{Пес}}}
\ncloop[loopsize=.25,angleB=180]{A}{B}
\end{pspicture}
```

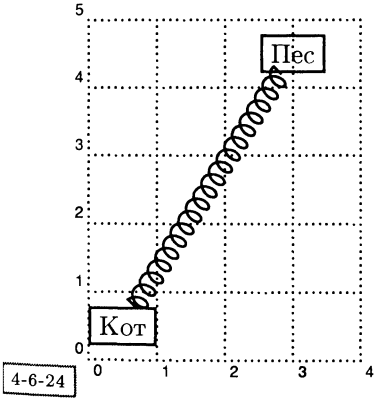
Существует специальная команда \nccircle, соединяющая вершину с самой собой:



4-6-23

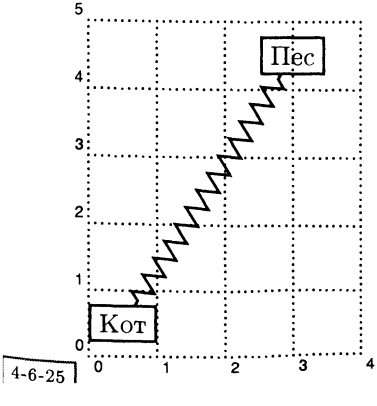
```
\usepackage {pstricks,pst-node}
\begin{pspicture}(-.5,-.1)(3,2)\showgrid
\rput(1,.5){\rnode{A}{\psframebox{Кот}}}
\nccircle{->}{A}{.5}
\end{pspicture}
```

PSTricks предлагает две специальные команды более высокого уровня, соединяющие вершины витыми (\nccoil) и зигзагообразными (\nczigzag) линиями, для чего требуется загрузить пакет pst-coil:



4-6-24

```
\usepackage {pstricks,pst-node,pst-coil}
\begin{pspicture}(-.5,-.1)(3.5,5)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Кот}}}
\rput(3,4.5){\rnode{B}{\psframebox{Пес}}}
\psset{coilarm=.01,coilwidth=.3}
\nccoil{A}{B}
\end{pspicture}
```

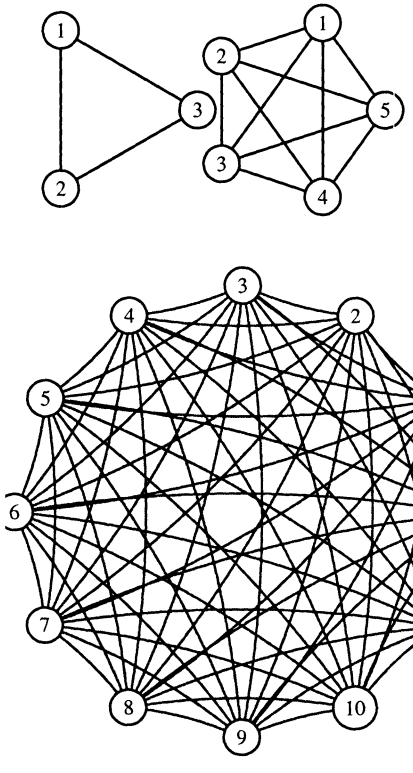


4-6-25

```
\usepackage {pstricks,pst-node,pst-coil}
\begin{pspicture}(-.5,-.1)(3.5,5)\showgrid
\rput(.5,.5){\rnode{A}{\psframebox{Кот}}}
\rput(3,4.5){\rnode{B}{\psframebox{Пес}}}
\psset{coilarm=.01,coilwidth=.3}
\nczigzag{A}{B}
\end{pspicture}
```

Если всего этого недостаточно, то соединительные элементы можно также использовать в качестве обычных инструментов для рисования с помощью команд, перечисленных в разд. 4.13.4 на с. 196, где каждая «pc»-версия команды соответствует аналогичному «nc»-элементу, соединяющему вершины.

Полученный с помощью соединительных элементов результат продемонстрирован в следующем примере, где для размещения объектов на окружности на одинаковых расстояниях друг от друга и соединения их применен макро `\multido` (см. разд. 4.9 на с. 170). Почему в определении объекта `\Wheel` мы явно указали черный цвет, а потом изменили его на белый перед тем, как несколько раз воспользоваться? Это прием, который иногда оказывается полезным для того, чтобы быть уверенным, что LaTeX отводит в выходном dvi-файле столько места, сколько необходимо. Для того чтобы в LaTeX'овском представлении объект имел правильный размер, мы использовали LaTeX'овскую команду `\fbox`, но установили цвет рамки белым и поэтому сама рамка не видна. Дальнейшее обсуждение этого приема см. в разд. 11.6 на с. 531.




```
\usepackage {pstcol,pst-node,multido}
\newcounter{CtA}
\newcounter{CtB}
\newcommand{\Wheel}[3]{{%
\color{black}
\psset{unit=#2}
\pspicture(-1,-1.2)(1,1.2)
\SpecialCoor
\degrees[#1]
\multido{\ia=1+1}{#1}{%
  \setcounter{CtA}{\ia}%
  \stepcounter{CtA}%
  \setcounter{CtB}{#1}%
  \addtocounter{CtB}{-\ia}%
  \multido{\ib=\value{CtA}+1}
     {\value{CtB}}{#3(1;\ia)(1;\ib)}}
\multido{\i=1+1}{#1}{%
  \rput(1;\i){%
   \pscirclebox[fillstyle=solid,
       fillcolor=white]%
    {\footnotesize\i}}}
\endpspicture}}% end of newcommand

{\color{white}
\fbox{\Wheel{3}{1.2}{\psline}
\Wheel{5}{1.2}{\psline}}

\psset{arcangle=10}
\fbox{\Wheel{12}{3}{%
     \pcarc[linecolor=blue]}}
}
```

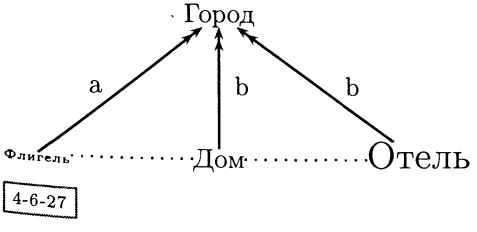
4-6-26

Очевидно, можно построить диаграммы и деревья произвольного вида, просто указав координаты каждой вершины. Однако на практике проще использовать два предназначенных для рисования матриц и деревьев окружения высокого уровня, которые обсуждаются в следующих двух разделах.

4.6.1 Матрицы: вершины, привязанные к сетке

Для размещения вершин могут быть использованы LaTeX'овское окружение `tabular` или $\mathcal{AMS}$-LaTeX'овское окружение `matrix`, но в пакете PSTricks есть свое собственное окружение `psmatrix`. Оно представляет собой упрощенный вариант таблицы, в котором не нужно указывать число столбцов. Столбцы разделяются символом `&`, строки — `\\` (как обычно), а PSTricks делает каждую ячейку вершиной и присваивает ей имя *rownumber*`,`*columnumber*. Таким образом первая вершина в первой строке называется `1,1`, а третья вершина в четвертой строке — `4,3`. Эти имена применяются при создании соединительных элементов.

У окружения `psmatrix` есть необязательный параметр, с помощью которого могут быть заданы величины `rowsep` и `colsep`, определяющие расстояние между вершинами. Внутри окружения `psmatrix` параметр `shortput` (см. с. 143) по умолчанию устанавливается равным `tab`, поэтому можно использовать весьма сжатую форму записи: команды, описывающие положение меток у соединительных элементов, следуют сразу же за определяющими их командами:



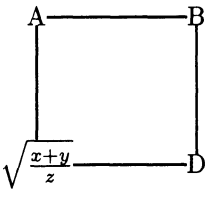
4-6-27

```
\usepackage {pstricks,pst-node}
\begin{pspicture}(0,-.85)(2.5,4)
\begin{psmatrix}[rowsep=1.5cm]
                & Город  &\\
 {\tiny Флигель} & Дом
                & {\Large Отель}
 \psset{arrows=<<-}
 \ncline{1,2}{2,1}<{a}
 \ncline{1,2}{2,2}>{b}
 \ncline{1,2}{2,3}>{b}
 \psset{arrows=-,linestyle=dotted}
 \ncline{2,1}{2,2}
 \ncline{2,2}{2,3}
\end{psmatrix}
\end{pspicture}
```

Обратите внимание, что символ «>» представляет собой сокращенную запись команды `\trput` (размещающей метки в соответствии с расстоянием между центрами вершин, а не длиной соединительного элемента), и применяется здесь, чтобы не учитывать разницу в размерах вершин, расположенных во втором ряду. С помощью команды `\psspan`, помещенной в конце ячейки, можно

сделать так, чтобы вершина охватывала несколько столбцов; параметром этой команды является число охватываемых столбцов.

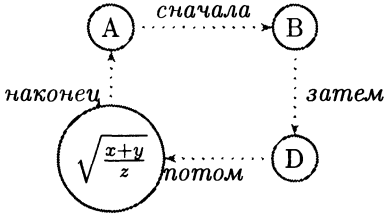
Рассмотрим простую квадратную матрицу:




```
\usepackage {pstricks,pst-node}
\begin{psmatrix}
 A                          & B \\
 $\sqrt{\frac{x + y}{z}}$ & D
 \ncline{1,1}{1,2}
 \ncline{1,2}{2,2}
 \ncline{2,2}{2,1}
 \ncline{2,1}{1,1}
\end{psmatrix}
```

4-6-28

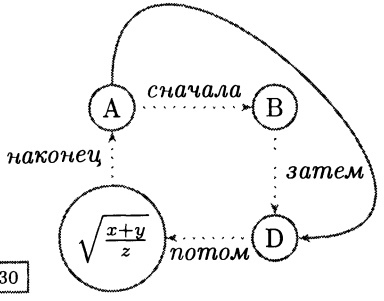
Изменив несколько установочных параметров, можно получить более «замысловатый» результат с обведенными окружностями вершинами, соединительными элементами со стрелками-указателями и более красивым расположением. Присвоение меток соединительным элементам осуществляется с помощью сокращенных вариантов команд, позиционирующих метки относительно центров вершин.




```
\usepackage {pstcol,pst-node}
\psset{arrows=->,labelsep=3pt,
linecolor=gray,mnode=circle}
\begin{psmatrix}
 [rowsep=20pt,colsep=40pt]
 A                          & B \\
 $\sqrt{\frac{x + y}{z}}$ & D
 \psset{linestyle=dotted}
 \ncline{1,1}{1,2}
   ^{\emph{сначала}}
 \ncline{1,2}{2,2}
   >{\emph{затем}}
 \ncline{2,2}{2,1}
   _{\emph{потом}}
 \ncline{2,1}{1,1}
   <{\emph{наконец}}
\end{psmatrix}
```

4-6-29

В показанном ниже втором варианте рисунка метки определяются другими вариантами команд, которые опираются на длину соединительных элементов, что в данном случае приводит к более красивому результату (сравните положение по вертикали слова *наконец* в левой части диаграммы). Помещенный снаружи соединительный элемент в форме петли является примером конструкции, протяженность которой в пространстве TeX, возможно, угадает неправильно.



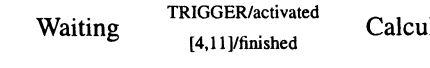
4-6-30

```
\usepackage {pstcol,pst-node}
\psset{arrows=->,labelsep=3pt,
linecolor=gray,mnode=circle}
\begin{psmatrix}
 [rowsep=20pt,colsep=32pt]
 A                          & B \\
 $\sqrt{\frac{x + y}{z}}$ & D
 \psset{linestyle=dotted}
 \ncline{1,1}{1,2}
   \naput{\emph{сначала}}
 \ncline{1,2}{2,2}
   \naput{\emph{затем}}
 \ncline{2,2}{2,1}
   \naput{\emph{потом}}
 \ncline{2,1}{1,1}
   \naput{\emph{наконец}}
 \nccurve[ncurv=2,
   linestyle=solid,angleA=90]{1,1}{2,2
\end{psmatrix}
```

Матрицы можно вкладывать друг в друга, а вершины двух разных матриц могут соединяться друг с другом, если им присвоены имена явным образом. Каждая новая команда `psmatrix` удаляет текущий набор имен вида *строка,столбец*.

В следующем примере показано использование пустой вершины для изображения идущего со стороны соединительного элемента, а также метод, с помощью которого две вершины соединяются двумя элементами, немного смещенными друг относительно друга:

```
\usepackage {pstricks,pst-node}
\psset{framearc=.4}
\begin{psmatrix}[colsep=3]
 \pnode{A}&\psframebox{Waiting} & \psframebox{Calculating}
 \ncline[arrows=->]{1,1}{1,2}
 \psset{offset=-.08}
 \ncline[arrows=->]{1,2}{1,3}\naput{{\scriptsize TRIGGER/activated}}
 \psset{offset=.08}
 \ncline[arrows=<-]{1,2}{1,3}\nbput{{\scriptsize [4,11]/finished}}
\end{psmatrix}
```

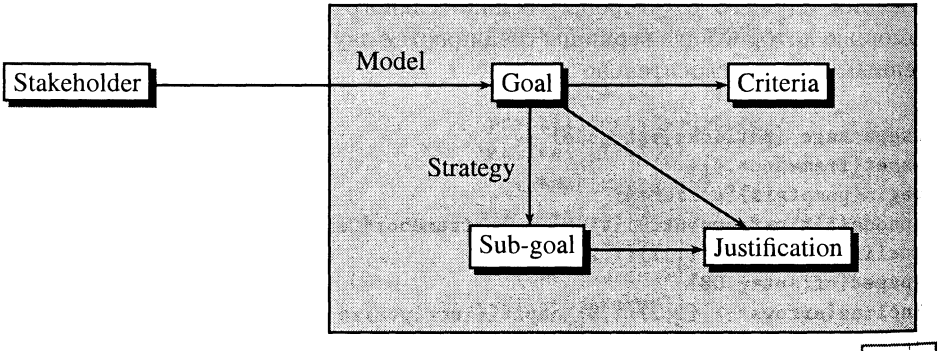


4-6-31

Обычную команду низкого уровня из пакета PSTricks, такую, как `\psframebox`, можно применить ко всей матрице. В следующем примере пришлось приложить некоторые усилия, чтобы соединительная линия была гори-

зонтальной; для этого одиночная вершина, стоящая слева, была помещена в свою собственную матрицу:

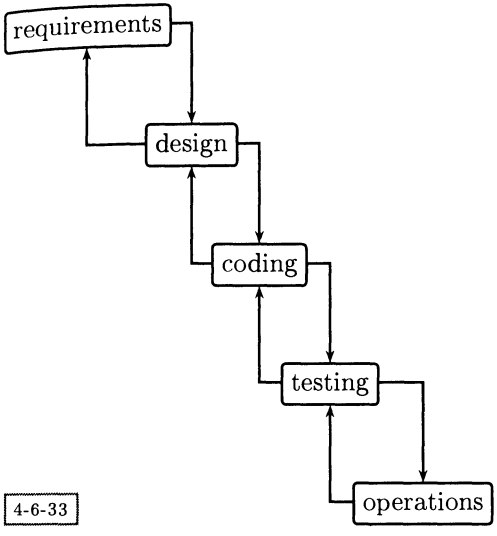
```
\usepackage {pstcol,pst-node}
\psset{arrows=->,fillcolor=white,fillstyle=solid}
\newcommand{\Show}[1]{\psshadowbox{#1}}
\definecolor{pink}{rgb}{1, .75, .8}
\begin{psmatrix}[mnode=r,ref=t]
  \psframebox[linestyle=none,framesep=.75]{%
    \begin{psmatrix}
      [name=A,ref=c]\Show{Stakeholder}
    \end{psmatrix}} &
  \psframebox[fillstyle=solid,fillcolor=pink,framesep=.75]{%
    \rule{1cm}{0pt}
    \begin{psmatrix}[ref=c]
      [name=B]\Show{Goal} & \Show{Criteria}\\
              \Show{Sub-goal} & \Show{Justification}
      \ncline{1,1}{1,2}
      \ncline{1,1}{2,2}
      \ncline{1,1}{2,1}\tlput{Strategy}
      \ncline{2,1}{2,2}
    \end{psmatrix}}
\end{psmatrix}
\ncline[angleB=180]{A}{B}\naput[npos=.7]{Model}
```



4-6-32

Весьма неожиданных результатов можно достичь с помощью *отрицательного значения* расстояния между столбцами, позволяющего последовательно идущим столбцам накладываться друг на друга. В приведенном ниже примере для соединения боксов используется трехзвенный элемент `\ncangle`, но благодаря установке значения параметра `armA` равным 0 первый элемент не имеет протяженности. Значения параметров `angleA` и `angleB` установлены таким образом, чтобы первый соединительный элемент выходил из первой вершины

вправо (angleA=0) и входил во вторую сверху (angleB=90), тогда как второй выходил бы слева (angleA=180) и входил снизу (angleB=-90):



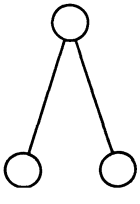
4-6-33

```
\usepackage {pstricks,pst-node}
\psset{framearc=.2}
\begin{psmatrix}
 [rowsep=1cm,colsep=-10pt]
 [name=A]\psframebox{requirements}\\
 &[name=B]\psframebox{design}\\
 &&[name=C]\psframebox{coding}\\
 &&&[name=D]\psframebox{testing}\\
 &&&&[name=E]\psframebox{operations}
 \psset{arrows=->,armA=0,angleB=90}
 \ncangle{A}{B}        \ncangle{B}{C}
 \ncangle{C}{D}        \ncangle{D}{E}
 \psset{angleB=-90,angleA=180}
 \ncangle{B}{A}        \ncangle{C}{B}
 \ncangle{D}{C}        \ncangle{E}{D}
\end{psmatrix}
```

4.6.2 Древовидные диаграммы

Пакет PSTricks обладает исключительно богатыми возможностями для построения древовидных диаграмм, имеющих сложную структуру и внешний вид. Поддержка этих возможностей осуществляется при загрузке дополнительного пакета pst-tree. Доступные команды перечислены в разд. 4.13.5 на с. 196, а графические параметры (относящиеся только к этим командам) перечислены в табл. 4.7 на с. 197. Как и следовало ожидать, большинство других команд и параметров тоже может быть использовано, будь то универсальные средства рисования или средства создания соединительных элементов и присвоения меток. Каждая из разновидностей вершин, описанных выше, превращается в узел дерева, ей присваивается имя с префиксом «T» и удаляется суффикс «node».

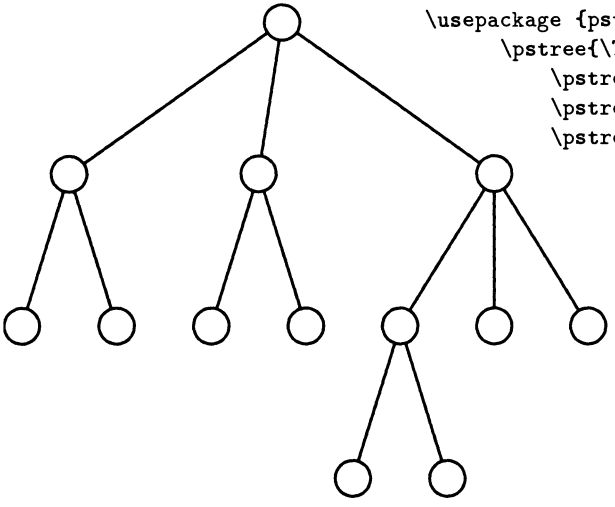
Фундаментальным понятием для построения деревьев в пакете PSTricks является *вложение* деревьев. Дерево (\pstree) состоит из корня и одного или нескольких узлов:



4-6-34

```
\usepackage {pstricks,pst-node,pst-tree}
\pstree{\TC}{
 \TC
 \TC
}
```

Каждый узел может сам быть деревом:

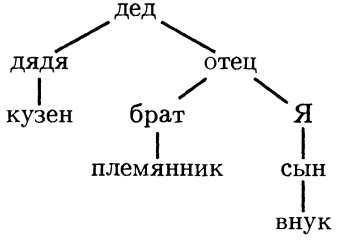


```
\usepackage {pstricks,pst-node,pst-tree}
    \pstree{\TC}{
        \pstree{\TC}{\TC \TC}
        \pstree{\TC}{\TC \TC}
        \pstree{\TC}{\pstree{\TC}
                              {\TC \TC}
                     \TC \TC
                     }
                }
```

4-6-35

Такое строение позволяет создавать структуры произвольной сложности.

Следующее дерево является аналогом напечатанного в Goossens et al. (1994), разд. 10.5.2, которое было нарисовано с помощью пакета ecltree. По простоте записи эти пакеты близки друг к другу. Как часто бывает в LaTeX'e, удобочитаемость сильно зависит от расположения текста программы. По сравнению с принятыми по умолчанию значениями в следующем примере было уменьшено расстояние по вертикали между деревьями (`levelsep`) и увеличено пространство вокруг узлов (`nodesep`).

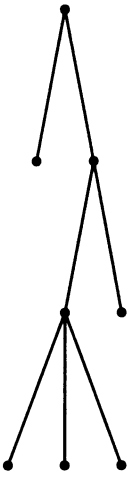



```
\usepackage {pstricks,pst-node,pst-tree}
\pstree[nodesep=2pt,
  levelsep=20pt]{\TR{дед}}
{\pstree{\TR{дядя}}{\TR{кузен}}
 \pstree{\TR{отец}}
   {\pstree{\TR{брат}}{\TR{племянник}}
   \pstree{\TR{Я}}
     {\pstree{\TR{сын}}{\TR{внук}}
     }
   }
}
```

4-6-36

Немного труднее воспроизвести пример, созданный другим пакетом для рисования деревьев, описанным в Goossens et al. (1994)), разд. 10.2.3, а именно пакетом **trees** Питера Ванруза.

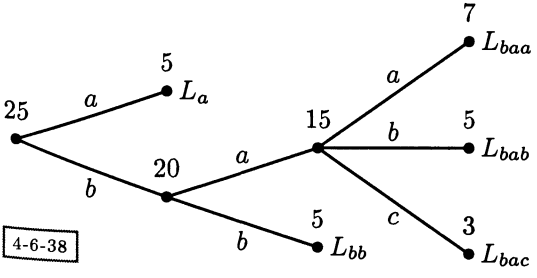
Каркас этого примера очень прост:



4-6-37

```
\usepackage {pstricks,pst-node,pst-tree}
\pstree{\Tdot}
{
\Tdot
\pstree{\Tdot}
 {
 \pstree{\Tdot}
  {\Tdot \Tdot \Tdot}
  \Tdot
 }
}
```

но после присвоения всем узлам и соединительным элементам меток и коррекции параметров с целью достижения более симпатичного результата внешний вид программы становится довольно сложным, хотя и похожим по объему на то, что получилось у Ванруза:



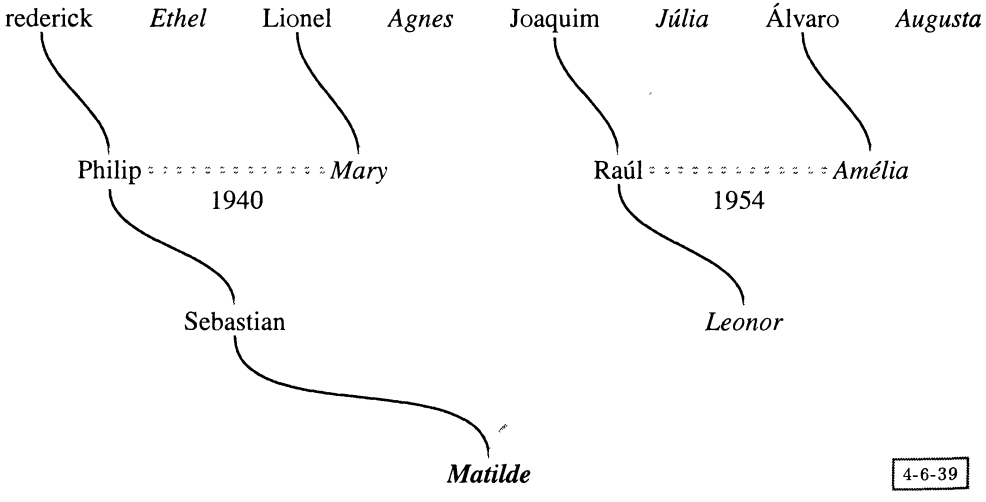
4-6-38

```
\usepackage {pstricks,pst-node,pst-tree}
\psset{labelsep=2pt,tnpos=a,radius=2pt}
          \pstree[treemode=R]{\TC*~{25}}
          {
          \TC*~{5}~[tnpos=r]{$L_a$}
          \taput{$a$}
          \pstree{\TC*~{20}\tbput{$b$}}
           {
           \pstree{\TC*~{15}\taput{$a$}}
            {
             \TC*~{7}~[tnpos=r]{$L_{baa}$}
               \taput{$a$}
             \TC*~{5}~[tnpos=r]{$L_{bab}$}
               \taput{$b$}
             \TC*~{3}~[tnpos=r]{$L_{bac}$}
               \tbput{$c$}
            }
            \TC*~{5}~[tnpos=r]{$L_{bb}$}
            \tbput{$b$}
           }
          }
```

Соединительные элементы в дереве создаются путем вызова макро \psedge, к которому указываются два узла. Действие команды \psedge может быть явно изменено либо с помощью ее переопределения, либо указанием параметра edge. В следующем примере мы переопределим команду \psedge так,

чтобы задать подправленными углами (учитывая, что дерево растет вверх) и достигнуть радующего глаз результата. Обратите внимание на явное задание связей между поименованными узлами, а также на горизонтальные соединительные элементы.

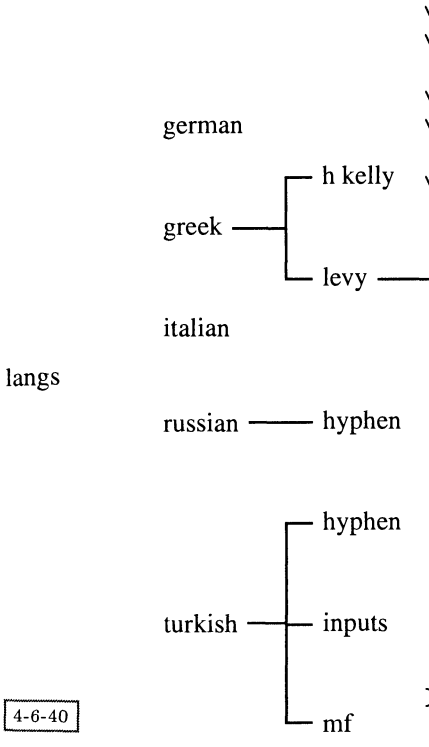
```
\usepackage {pstcol,pst-node,pst-tree}
\definecolor{pink}{rgb}{1, .75, .8}
\renewcommand\psedge{\nccurve}
\newcommand{\Female}[2][]{{\psset{linecolor=pink}\TR[#1]{\emph{#2}}}}
\newcommand{\Male}[2][]{{\psset{linecolor=blue}\TR[#1]{#2}}}
\psset{nodesep=2pt,angleA=90,angleB=-90}
\pstree[treemode=U]{\Female{{\bfseries Matilde}}}{
 \pstree{\Male{Sebastian}}{
  \pstree{\Male[name=P]{Philip}}{\Male{Frederick}\Female{Ethel}}
  \pstree{\Female[name=W]{Mary}}{\Male{Lionel}\Female{Agnes}}}
 \pstree{\Female{Leonor}}{
  \pstree{\Male[name=R]{Ra\'ul}}{\Male{Joaquim}\Female{J\'ulia}}
  \pstree{\Female[name=A]{Am\'elia}}{\Male{\'Alvaro}\Female{Augusta}}}
}
\psset{linecolor=green,doubleline=true,linestyle=dotted}
\ncline{P}{W}\nbput{1940}
\ncline{R}{A}\nbput{1954}
```



4-6-39

Переопределение внешнего вида соединительных элементов оказывается также необходимым, если мы хотим получить вместо прямых линий линии, идущие под прямым углом. При этом нужно следить за параметрами `angleA` и `angleB`, а также убедиться в том, что все узлы в этом дереве, растущем слева направо, выровнены по левому краю (путем задания параметра `ref`).

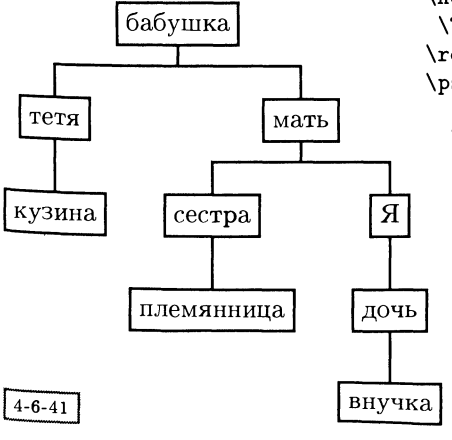
Если этого не сделать, полученные с помощью команды \ncangle края будут выглядеть достаточно странно.




```
\usepackage {pstricks,pst-node,pst-tree}
\newcommand{\Item}[1]{\Tr[ref=l]{%
 \psframebox[linestyle=none]{#1}}}
\renewcommand{\psedge}{\ncangle}
\psset{xbbd=1.5cm,treemode=R,
    angleB=-180,angleA=0,levelsep=60pt}
\pstree{\Item{langs}}{%
          \Item{german}
          \pstree{\Item{greek}}{%
            \Item{h  kelly}
            \pstree{\Item{levy}}{%
              \Item{doc}
              \Item{src}
               }
             }
          \Item{italian}
          \pstree{\Item{russian}}{
            \Item{hyphen}
           }
          \pstree{\Item{turkish}}{%
            \Item{hyphen}
            \Item{inputs}
            \Item{mf}
           }
}
```

4-6-40

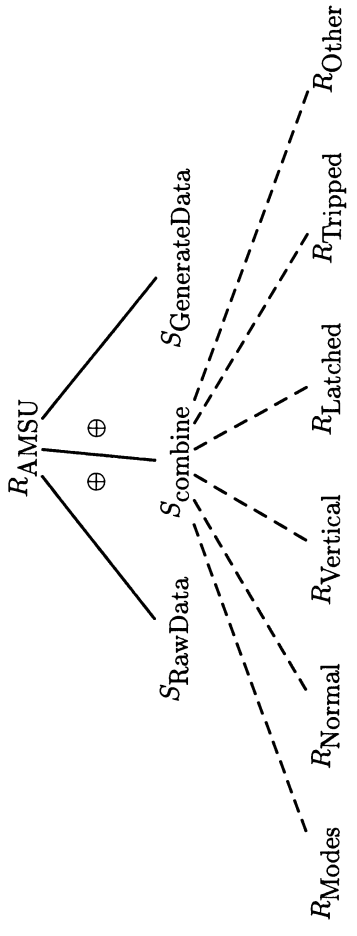
Этот прием позволяет по-новому изобразить небольшое семейное дерево. На этот раз соединительные элементы выглядят гораздо симпатичнее:




```
\usepackage {pstricks,pst-node,pst-tree}
\newcommand{\XX}[1]{%
 \Tr{\psframebox{\rule[-2pt]{0pt}{9pt}#1}}}
\renewcommand{\psedge}{\ncangle}
\psset{angleB=90,angleA=-90,
   levelsep=36pt,armB=12pt}
  \pstree{\XX{бабушка}}
    {\pstree{\XX{тетя}}{\XX{кузина}}
     \pstree{\XX{мать}}
      {\pstree{\XX{сестра}}
       {\XX{племянница}}
       \pstree{\XX{Я}}
        {\pstree{\XX{дочь}}
         {\XX{внучка}}
        }
       }
    }
```

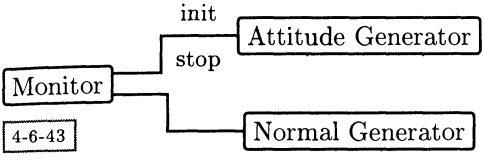
4-6-41

Соединяющие узлы элементы, конечно, являются стандартными объектами пакета PSTricks, поэтому для них можно задавать любые обычные параметры. В следующем примере в нижней части дерева использованы штриховые линии, а содержимое каждого узла набрано в математическом режиме. Чтобы продемонстрировать, что PSTricks'овские конструкции нормально взаимодействуют с другими пакетами, поворот всего дерева на 90 градусов осуществлен с помощью команды `\rotatebox` стандартного LaTeX'овского графического пакета.




```
\usepackage {pstcol,pst-node,pst-tree,graphics}
\rotatebox{90}{%
\psset{nodesep=2pt}
\newcommand{\XX}[2]{%
          \TR{\ensuremath{#1_{\mbox{#2}}}}%
         }
\pstree[xbbr=1.5cm]{\XX{R}{AMSU}}
          {
           \XX{S}{RawData}
           \pstree{\XX{S}{combine}
                   \trput{\ensuremath{\oplus}}
                   \tlput{\ensuremath{\oplus}}}
            {
            \psset{linestyle=dashed}
            \XX{R}{Modes}
            \XX{R}{Normal}
            \XX{R}{Vertical}
            \XX{R}{Latched}
            \XX{R}{Tripped}
            \XX{R}{Other}
            }
           \XX{S}{GenerateData}
          }
}
```

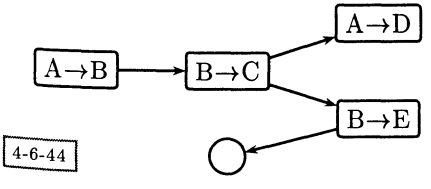
Если требуется построить деревья ясной структуры, исходящие из одного узла, то соединительный элемент \ncangle, к которому добавлено небольшое смещение, позволяет получить желаемый результат:



4-6-43

```
\usepackage {pstcol,pst-node,pst-tree}
\psset{framearc=.2,levelsep=4cm,
  armB=1cm,angleB=-180}
\renewcommand{\psedge}{\ncangle}
\newcommand{\TreeBox}[1]
                {\Tr{\psframebox{#1}}}
\pstree[treemode=R]{\TreeBox{Monitor}}
{\psset{offsetA=4pt}
 \TreeBox{Attitude Generator}
 \naput[npos=2.5]{{\small init}}
 \nbput[npos=2.5]{{\small stop}}
 \psset{offsetA=-4pt}
 \TreeBox{Normal Generator}
}
```

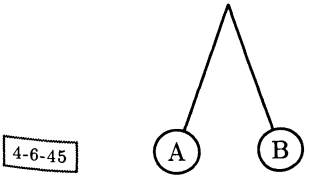
В случае простых соединительных элементов нам не нужно заботиться о смещениях или углах. В следующем примере показано, как можно изменять направление во вложенных деревьях:



4-6-44

```
\usepackage {pstcol,pst-node,pst-tree}
\psset{arrows=->,framearc=.2}
\newcommand{\Treebox}[1]{%
 \Tr{\psframebox{#1}}}
\pstree[treemode=R]
{\Treebox{A$\rightarrow$B}}{
\pstree{
   \Treebox{B$\rightarrow$C}
          }{
 \Treebox{A$\rightarrow$D}
\pstree[treemode=L]
 {\Treebox{B$\rightarrow$E}}
 {\Tn\TC[arrows=<-]}
      }
}
```

Наконец, не забудем и о простом пустом узле:

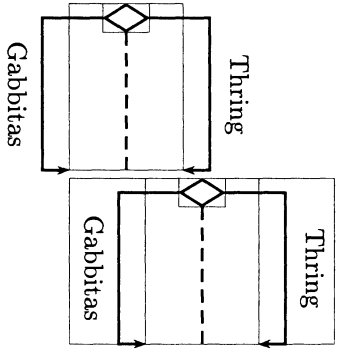


4-6-45

```
\usepackage {pstcol,pst-node,pst-tree}
\pstree{\Tp}{
 \Tcircle{A}
 \Tcircle{B}
}
```

4.6.3 Задание обрамляющих боксов деревьев

Как мы уже говорили выше, пакету PSTricks не всегда удается корректно определить занимаемую объектами площадь, что иллюстрируется соединительными элементами в следующем примере. Благодаря активации параметра showbbox можно увидеть, что без указания параметров xbbl и xbbr во втором случае содержащий дерево бокс был бы определен неверно:




```
\usepackage {pstcol,pst-node,pst-tree}
\psset{angleB=-90,arrows=->,nrot=:U}
\newcommand{\molesworth}[1]{%
 \pstree[#1]{\Tdia{ }}
 {
  \Tp[arrows=->,
      edge={\ncbar[angleA=180]}]
  \nbput{Gabbitas}
  {\psset{linestyle=dashed,arrows=-}
   \Tp }
  \Tp[arrows=->,edge={\ncbar}]
  \naput{Thring}
 }
}
\psset{showbbox=true}
\begin{tabular}{l}
\molesworth{}\\[1cm]
\molesworth{xbbl=1cm,xbbr=1cm}
\end{tabular}
```

4-6-46

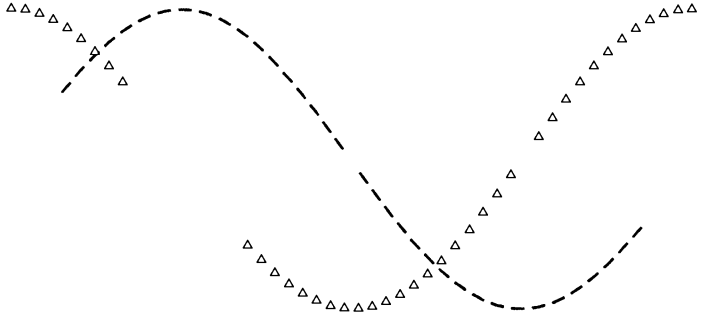
4.7 Графическое представление данных

Пакет PSTricks содержит набор средств высокого уровня для обычных задач графического представления информации в разнообразных форматах на основе считанных из внешних файлов данных. Эти команды, входящие в дополнительный пакет pst-plot, приведены в разд. 4.13.6 на с. 198, а дополнительные параметры перечислены в табл. 4.8 на с. 199. Простейшим форматом данных является перечень чисел, разделенных запятыми или пробелами, но числа также могут быть заключены в фигурные ({}) или круглые (()) скобки. Если все данные заключены внутри одной пары *квадратных* ([]) скобок и открывающая скобка [расположена в начале строки, они считываются гораздо быстрее. Однако в этом случае TeX быстрее начинает жаловаться на нехватку памяти. Будучи единожды считанными, данные могут

использоваться повторно с помощью команд `\readdata` и `\savedata`. Между командами `\fileplot` или `\dataplot`, выполняющими синтаксический анализ и проверку данных в TEX'е, и командой `\listplot`, просто передающей данные PostScript'у, есть существенное различие. Последний вариант не предполагает проверки переполнения памяти PostScript'а, но обладает тем преимуществом, что в выходной файл для создания или преобразования данных могут быть включены любые PostScript'овские инструкции. В сложных случаях применение макро `\PSTtoEPS` вместе с командами для графического представления данных позволяет использовать массивы данных еще бóльшего объема. Пример этого будет приведен в разд. 4.10.2 на с. 178.

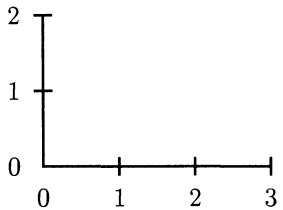
Пользователи должны осознать, что проверка занимаемого данными места и выбор масштаба оказываются их обязанностью. В отличие от многих других пакетов, PSTricks не подгоняет данные в заранее отведенное для них пространство. Обычно разумный выбор значений параметров `xunit` и `yunit` позволяет быстро добиться прекрасных результатов. Оси создаются отдельно, при этом не предпринимается никаких действий для размещения каких-либо специальных меток в точках, обозначающих шаг шкалы. С другой стороны, становится очень легко накладывать несколько графиков один поверх другого и использовать язык PostScript для вычисления функций, как сделано в следующем примере, где друг на друга наложены функции $\sin(x)$, $\sin(x)\cos(x)$ и $\cos(x)$.

```
\usepackage {pstcol,pst-plot}
\begin{pspicture}(-1,-2.5)(9,2.5)
\psset{xunit=.25mm,yunit=2cm}
\psset{plotpoints=50}
\psplot[linestyle=dashed,linecolor=blue]{0}{360}{x sin}
\psplot[plotstyle=dots,dotstyle=triangle]{0}{360}{x cos}
\psset{plotpoints=200}
\psplot[linecolor=red]{0}{360}{x dup sin exch cos mul}
\end{pspicture}
```

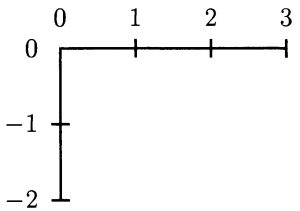


4-7-1

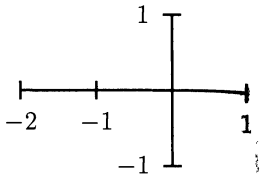
Оси или рамки для графиков могут создаваться многими способами. Присвоение меток по умолчанию является вполне разумным, что видно из приведенных ниже типичных случаев:



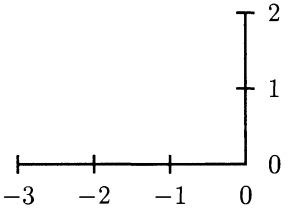
psaxes(3,2)



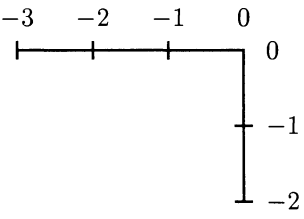
psaxes(3,-2)



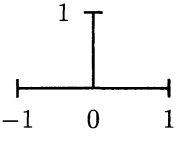
psaxes(2,1)(0,0)(3,2)



psaxes(-3,2)



psaxes(-3,-2)

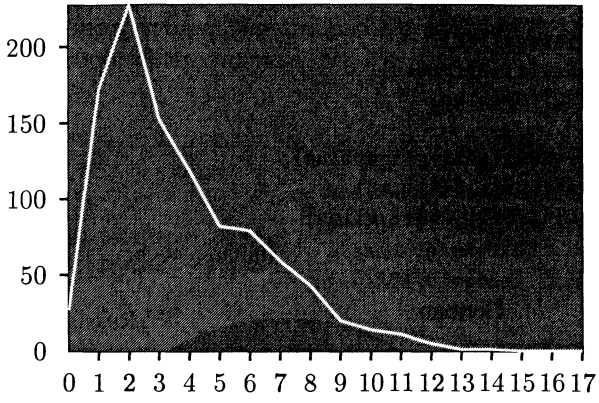


psaxes(0,0)(-1,0)(1,1)

4-7-2

Команда \psaxes также может использоваться для создания рамок у графиков. В этом случае в ней можно указывать графические параметры заполнения и задания цвета. В следующем примере (иллюстрирующем частоту, с которой встречаются слова разной длины в некотором отрывке из Диккенса) также показано, как можно оперировать метками на осях.

```
\usepackage {pstcol,pst-node,pst-tree}
\readdata{\foo}{words.dat}
\psset{yunit=.2mm,xunit=4mm}
\begin{pspicture}(-2,-50)(20,250)
\psaxes[axesstyle=frame,dy=50\psyunit,Dy=50,tickstyle=bottom,
        fillcolor=gray,fillstyle=solid](1,1)(18,229)
\dataplot[plotstyle=line,linecolor=white]{\foo}
\end{pspicture}
```

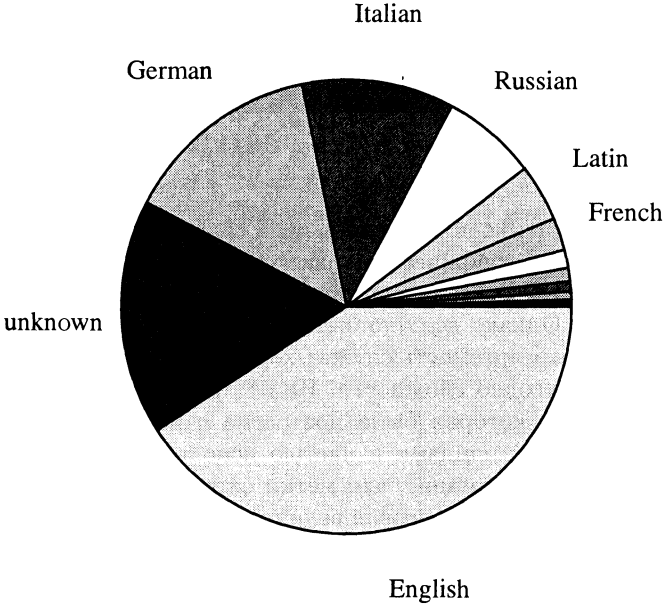


4-7-3

При создании PSTricks не ставилась цель сравниться по возможностям со зрелым коммерческим графическим пакетом. Функции графического представления данных в PSTricks предназначались только для построения простых научных графиков. Однако должно быть понятно, что PSTricks'овские макро и TEX'овская программируемость служат хорошим основанием для получения изображений какой угодно сложности. Ниже представлен еще один вариант круговой диаграммы, которая была построена с помощью METAPOST'а на с. 106, наряду с макрокомандами и частью использованных для ее построения данных. Обратите внимание, что метки есть только у сегментов, размер которых больше некоторой пороговой величины (присвоение осмысленных меток всем сегментам потребовало бы гораздо больших усилий). Дени Жиру написал для PSTricks универсальный пакет для создания круговых диаграмм и гистограмм, способный выдавать очень профессиональные результаты (он использует Unix'овские программы sh и awk, называется pie-chart.sh и может быть найден в дистрибутиве PSTricks).

```
\usepackage{calc,pstcol}
\newcounter{lastval}
\newcounter{thishalf}
\newcounter{thisval}
\definecolor{lightblue}{rgb}{.68, .85, .9}
\newcommand{\lang}[4]{% name, value, proportion of 10000, color
 \setcounter{thisval}{\value{lastval}}
 \addtocounter{thisval}{#3}
 \pswedge[fillcolor=#4]{1}{\thelastval}{\thethisval}%
 \setcounter{thishalf}
           {(((\value{thisval}-\value{lastval})/2)+\value{lastval}}
 \ifnum#3>200\rput(1.3;\thethishalf){#1}\fi
 \setcounter{lastval}{\value{thisval}}
}
\psset{fillstyle=solid}
\degrees[10000]
```

```
\SpecialCoor
\setcounter{lastval}{0}
\lang{Romanian}{1}{3}{green}
\lang{Czech}{2}{6}{blue}
...
\lang{German}{508}{1421}{lightblue}
\lang{unknown}{599}{1676}{red}
\lang{English}{1462}{4085}{yellow}
```



4-7-4

4.8 Работа с третьим измерением

Более поздние версии пакета PSTricks включают некоторые экспериментальные средства просмотра трехмерных объектов. Можно построить проекции двумерных объектов в трехмерную систему координат и смотреть на них под разными углами. В настоящее время PSTricks не умеет изображать реальные трехмерные объекты, строить их с учетом перспективы, удалять закрытые объектом линии или передавать игру света, так что полезность этой части пакета ограничена. Однако, набравшись терпения, можно добиться неплохих результатов (многочисленные примеры которых, как обычно, можно найти в Girou (1994)). Работа с третьим измерением осуществляется путем загрузки дополнительного пакета pst-3d. В разд. 4.13.7 на с. 200 перечислены новые команды этого пакета, а в табл. 4.9 на с. 200 описаны используемые в них дополнительные графические параметры.

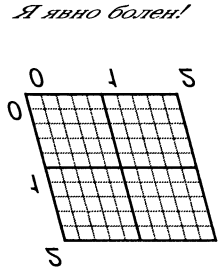
Есть две «трехмерные» команды высокого уровня и один универсальный инструмент. Первая команда — это `\psshadow`, создающая отбрасываемую

некоторым текстом тень, угол падения и размер которой (в виде доли размера основного текста) могут быть заданы:

4-8-1

```
\usepackage {pstcol,pst-3d}
\psshadow[Tshadowangle=45,
  Tshadowsize=2.5]
 {\LARGE\bfseries Слова бросают тень}
```

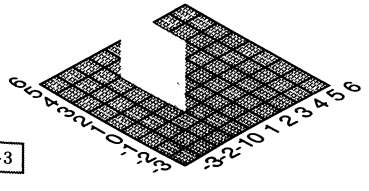
Команда `\pstilt` наклоняет объект в третьем измерении:



4-8-2

```
\usepackage {pstcol,pst-3d}

 \pstilt{45}{Я явно болен!}

\begin{pspicture}(0,-2)(2,1)
  \pstilt{-75}{\psgrid(0,0)(2,2)}
\end{pspicture}
```

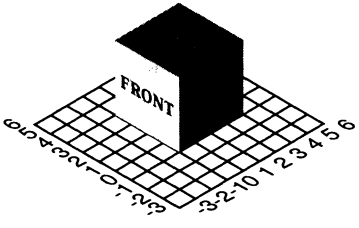
Универсальный макро `\ThreeDput` помещает любой PSTricks'овский объект в некоторую точку трехмерного пространства. В нем почти всегда указывается параметр `normal`, задающий вектор нормали к плоскости объекта. В качестве простого примера предположим, что мы хотим нарисовать куб так, чтобы точка наблюдения располагалась с левой стороны от передней грани. В таком случае передняя грань куба могла бы быть нарисована следующим образом:



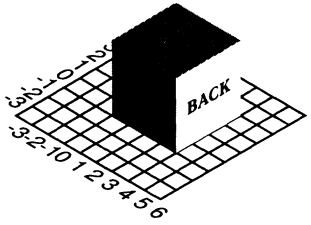
4-8-3

```
\usepackage {pstcol,pst-3d}
\begin{pspicture}(-5,-3)(4,3)
 \psset{unit=.3cm}
 \psset{viewpoint=-1 -1 1}
 \ThreeDput[normal=0 0 1]
           {\psgrid(-3,-3)(6,6)}
 \ThreeDput[normal=-1 0 0](0,4,0){
   \psframe*[linecolor=yellow](4,4)}
\end{pspicture}
```

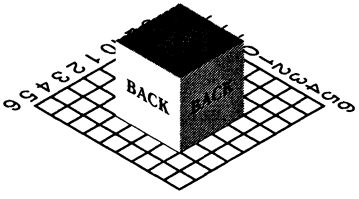
Направление наблюдения `viewpoint` описывается как «сверху» (−1), «сзади» (−1) и «слева» (1). Далее весь куб целиком показан с четырех «верхних» углов:



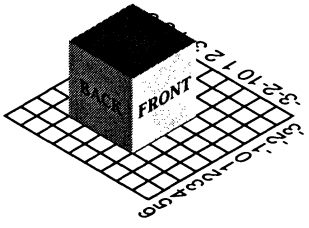
Направление наблюдения: -1 -1 1



Направление наблюдения: 1 -1 1



Направление наблюдения: 1 1 1

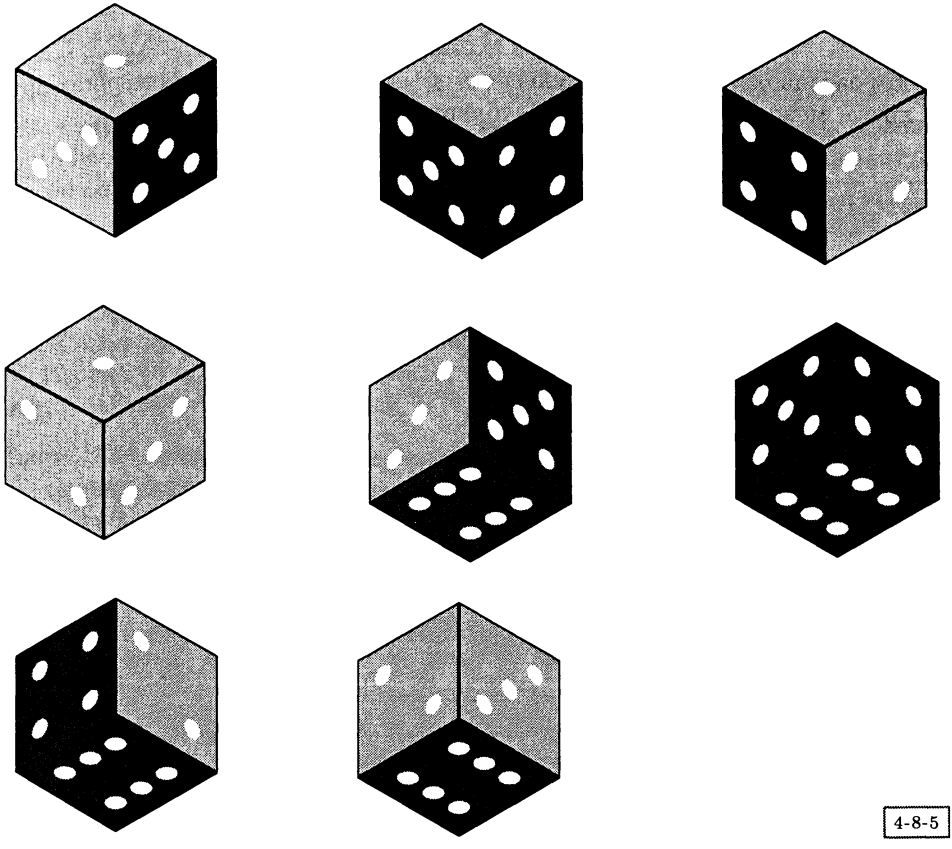


Направление наблюдения: -1 1 1

4-8-4

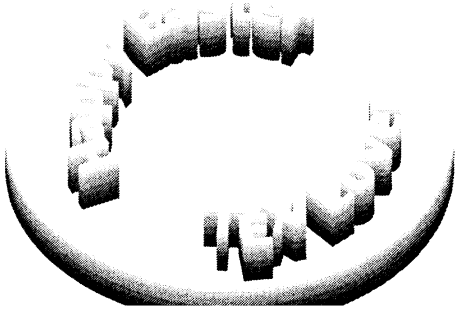
Важно понимать, что PSTricks не проверяет, какая сторона объекта какую загораживает. Последняя нарисованная сторона закрывает все, что было нарисовано ранее даже в том случае, если в трехмерном пространстве она располагается сзади. Поэтому при рисовании под разными ракурсами необходимо рисовать стороны в разном порядке. На следующем рисунке иллюстрируется вышесказанное. Показывая «нижнюю сторону», необходимо в правильном порядке по отношению к направлению взгляда изобразить все шесть граней. Грубое «освещение» игральной кости достигается просто за счет окрашивания трех граней более светлым цветом (изображение хода лучей от источника света существенно превосходит возможности этого пакета!). Ниже приведены значения параметра `normal` для каждой из граней.

Сторона	`normal`			*положение XYZ*
1	0	0	1	0,0,4
2	0	1	0	4,4,0
3	-1	0	0	0,4,0
4	1	0	0	4,0,0
5	0	-1	0	0,0,0
6	0	0	-1	0,4,0



4-8-5

Следуя примеру еще одного рисунка, созданного Дени Жиру, можно попробовать воспользоваться макро `\ThreeDput`, чтобы создать эффект выпуклого текста (этот рисунок также показан на цветной вклейке IVb):

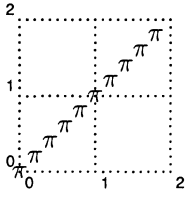


4.9 Команды итерации

Чтобы расположить последовательность объектов на равных расстояниях друг от друга, можно воспользоваться дублирующим макро:

\multirput* [*refpoint*] {*angle*} (x_0,y_0) (x_1,y_1){*int*}{*stuff*}

Он действует аналогично команде \rput, но создает *int* копий, помещая первую в точку (x_0,y_0) и отступая каждый раз на (x_1,y_1). Этот макро можно использовать только в декартовой системе координат. Например:



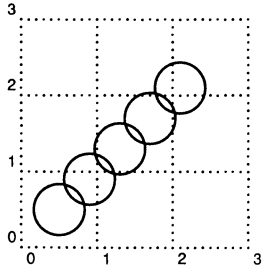

```
\usepackage {pstricks}
\begin{pspicture}(0,0)(2,2)\showgrid
\multirput(0,0)(.2,.2){10}{$\pi$}
\end{pspicture}
```

4-9-1

Для многократного повторения обычных графических объектов существует другой макро:

\multips{*angle*} (x_0,y_0) (x_1,y_1) {*int*} {*graphics*}

синтаксис которого похож на синтаксис команды \multirput. Используйте только такие графические объекты, которые не занимают места.




```
\usepackage {pstricks}
\begin{pspicture}(0,0)(3,3)\showgrid
\multips(.5,.5)(.4,.4){5}
        {\pscircle{.35}}
\end{pspicture}
```

4-9-2

Обратите внимание, что для каждой копии объекта точка $(0,0)$ смещается на величину \multips.

Более общая система многократного повторения реализуется пакетом multido, который неявно запускается при использовании команд графического представления данных (этот пакет в действительности не зависит от PSTricks и может использоваться самостоятельно). Синтаксис команды \multido имеет вид:

`\multido{`*variables*`}{`*repetitions*`}{`*actions*`}`

где аргумент *actions* описывает какие-то действия, подлежащие повторению, и может являться любой TEX'овской командой. Аргумент *repetitions* определяет число повторений. Первый аргумент представляет собой разделенный запятой список описаний переменных, каждое из которых состоит из

переменная = *начальное значение* + *прирост*

Переменная устанавливается равной *начальному значению*, а потом при каждом повторении увеличивается на величину *прироста*.

Первая буква имени *переменной* определяет тип переменной. Поддерживаются четыре типа переменных:

Длина (dimension) (d или D) Начальное значение и прирост должны быть представлены в TEX'овских единицах измерения. Текущее значение выражено в единицах измерения `sp` (TEX'овские масштабные пункты). Например, `\dx=4cm+5pt`.

Число (number) (n или N) Начальное значение и прирост должны быть целыми числами или числами с одинаковым количеством знаков после запятой (до восьми). При добавлении прироста допускается операция сложения с фиксированной запятой. Например, `\n=3+7.05`, `\Nx=5.30+-1.25`.

Целое число (integer) (i или I) Начальное значение и прирост должны быть целыми числами. Результат аналогичен предыдущему пункту, но вычисляется быстрее. Например, `\I=2+-1`.

Действительное число (real) (r или R) Начальное значение и прирост должны либо быть целыми, либо иметь не более четырех цифр по каждую сторону от десятичной точки. Текст подстановки является числом, но допускается операция сложения с плавающей запятой и случайные небольшие ошибки. Этот вариант дает менее удовлетворительные результаты, чем использование типа «число», но работает быстрее. Например, `\ry=4.2+1.05`.

Документация к макро находится прямо в его исходных текстах и описывает еще некоторые другие возможности, но основные способы применения показаны в примерах, встречающихся на протяжении этой главы. В сложном примере на рис. 4.1 (предоставленном Дени Жиру) активно используются многократно повторяемые регулярные операции.

4.10 Настройка и программирование PSTricks

Хотя любой графический материал, написанный на TEX'е, может использовать возможности макропрограммирования самого TEX'а, в PSTricks предоставляются дополнительные средства описания рисунков более элегантным способом, чем в виде простого перечня команд. Мы считаем такую возможность важной

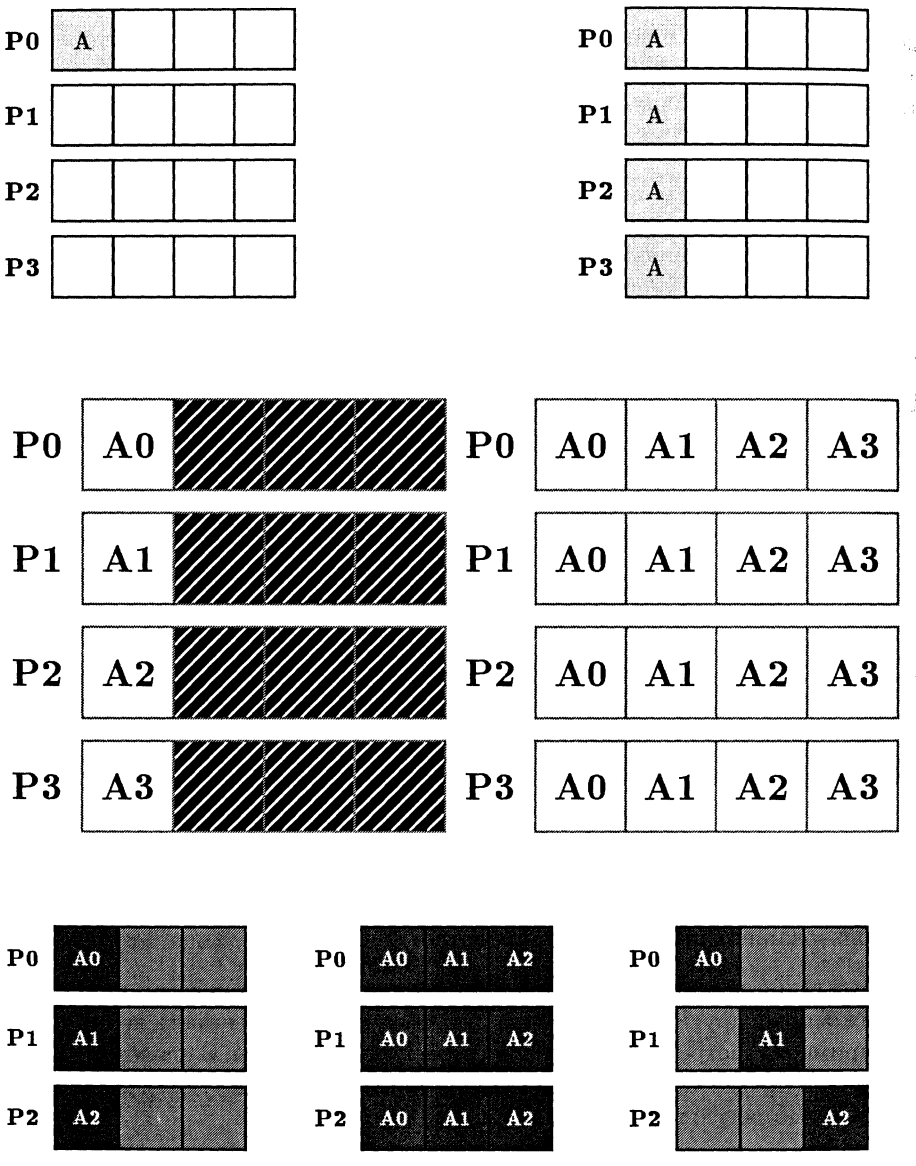


Рис. 4.1. Сетка процессора, запрограммированная Дени Жиру.

частью подобного способа создания рисунков и одной из главных причин выбора рисования в TeX'овской среде, а не с помощью внешнего графического пакета. В данном разделе рассматриваются некоторые свойства, достойные применения, и демонстрируются стили программирования.

4.10.1 Новые объекты и стили в PSTricks

Иногда бывает полезно путем создания особых версий графических объектов со своими собственными параметрами, принимаемыми по умолчанию, обойтись без длинных списков параметров и неуклюжих вызовов команды `\psset`. Это достигается с помощью команды `\newpsobject`:

`\newpsobject{`*name*`}{`*object*`}{`*par1=value1,...*`}`

которая на основе старого объекта *object* создает новый объект под именем *name* с иначе инициализированными параметрами. Примером может служить команда:

```
\newpsobject{trafficsign}{pscirclebox}{linecolor=red,doubleline=true}
```

Параметры при обращении к команде `\trafficsign` можно задать и явным образом:

```
\trafficsign[linecolor=green, doublesep=20pt]{School}
```

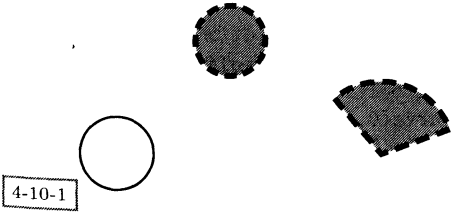
Макрос `\showgrid`, который уже встречался нам раньше во многих примерах в этой главе, был определен следующей командой:

```
\newpsobject{showgrid}{psgrid}{subgriddiv=1,griddots=10,gridlabels=6pt}
```

Действуя иначе, можно ассоциировать новый стиль с набором графических параметров, который вы хотите использовать многократно:

`\newpsstyle{`*name*`}{`*par1=value1,...*`}`

После этого можно применять имя *name* в любых создаваемых объектах, указывая его в качестве значения параметра `style`:




```
\usepackage {pstcol}
\begin{pspicture}(0,0)(5,3)
\newpsstyle{bomb}{fillcolor=gray,
     fillstyle=solid,linestyle=dashed,
     linewidth=2pt}
\pscircle[style=bomb](2,2){.5}
\pscircle(.5,.5){.5}
\pswedge[style=bomb](4,.5){1}{20}{130}
\end{pspicture}
```

Предоставляющий гораздо большие возможности, но и более сложный прием заключается в конструировании совершенно новых составных объектов с помощью команды `\pscustom`:

`\pscustom*[`*settings*`]{`*commands*`}`

Полное описание этой системы см. в van Zandt (1993a). Здесь дан лишь краткий обзор.

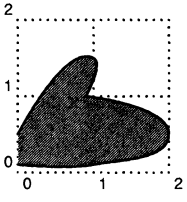
Итогом исполнения команды \pscustom является создание нового объекта, к которому можно применять команды закрашивания и который является замкнутой в терминах PostScript'а траекторией. Хотя в команде \pscustom может присутствовать большинство PSTricks'овских объектов, рекомендуется пользоваться только самыми простыми из них (такими, как \psline и \psarc), просто рисующими траектории. Дополнительный графический параметр liftpen указывает, должен ли рисуемый объект соединяться с предыдущей текущей точкой прямой линией и должна ли текущая точка считаться начальной точкой кривой. Параметр liftpen может принимать следующие значения:

0 Линия проводится из текущей точки до начальной точки следующего объекта. Если им оказывается кривая или прямая линия, то текущая точка считается ее первой координатой.

1 Прямая линия проводится из текущей точки до начальной точки следующего объекта.

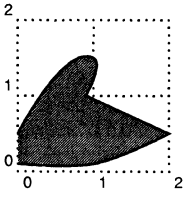
2 Текущая точка не оказывает никакого влияния на кривую. Это означает, что две последовательные кривые не соединяются и поэтому пространство между ними не закрашивается.

Эти три типа поведения видны из следующего примера:



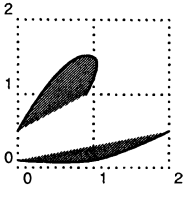

```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,2)\showgrid
\pscustom[fillstyle=solid,
    fillcolor=gray]{%
 \pscurve(0,.5)(1,1.5)(.9,1)
 \pscurve[liftpen=0](2,.5)(1,.1)(0,.1)
}
\end{pspicture}
```

4-10-2




```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,2)\showgrid
\pscustom[fillstyle=solid,
    fillcolor=gray]{%
 \pscurve(0,.5)(1,1.5)(.9,1)
 \pscurve[liftpen=1](2,.5)(1,.1)(0,.1)
}
\end{pspicture}
```

4-10-3




```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,2)\showgrid
\pscustom[fillstyle=solid,
   fillcolor=gray]{%
 \pscurve(0,.5)(1,1.5)(.9,1)
 \pscurve[liftpen=2](2,.5)(1,.1)(0,.1)
}
\end{pspicture}
```

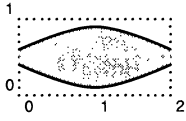
4-10-4

Вдобавок к графическим объектам пакета PSTricks оказывается доступным еще один набор команд, действие которых аналогично действию PostScript'овских операторов с соответствующими именами. Ими являются команды установки `\newpath`, `\gsave`, `\grestore`, `\msave` и `\mrestore`, команды позиционирования `\moveto`*(coordinates)*, команды для рисования `\stroke` и `\fill` и команды преобразования `\translate`*(coordinates)*, `\scale{`*num1 num2*`}` и `\rotate{`*angle*`}`. Команды `\msave` and `\mrestore` позволяют сохранить и восстановить *только* систему координат, тогда как команды `\gsave` и `\grestore` являются более сильнодействующими. На рис. 4.2 показано, как с помощью введенных им команд Дени Жиру запрограммировал случайное блуждание (генератор случайных чисел создан Дональдом Арсено).

Эти дополнительные средства позволяют закрашивать пространство между двумя кривыми, не проводя вспомогательных линий и не соединяя их:

```
\usepackage {pstcol}
\begin{pspicture}(0,0)(2,1)\showgrid
\pscustom{%
 \pscurve(0,.6)(1,.9)(2,.6)
 \gsave
   \pscurve[liftpen=1](2,.4)(1,.1)(0,.4)
   \fill[fillstyle=solid,
         fillcolor=lightgray]
 \grestore
}
  \pscurve[liftpen=2](2,.4)(1,.1)(0,.4)
\end{pspicture}
```

4-10-6



Этот трюк работает потому, что команды `\gsave` и `\grestore` в действительности препятствуют рисованию линии второй командой `\pscurve`, устраняя результаты ее работы после ее завершения. Однако команда `\fill` продолжает работать, поскольку она обращается к существующей при ее вызове траектории. После этого повторно выполняется команда `\pscurve`, наконец-то вычерчивающая нижнюю линию.

Существуют и другие PSTricks'овские версии аналогичных PostScript'овским команд:

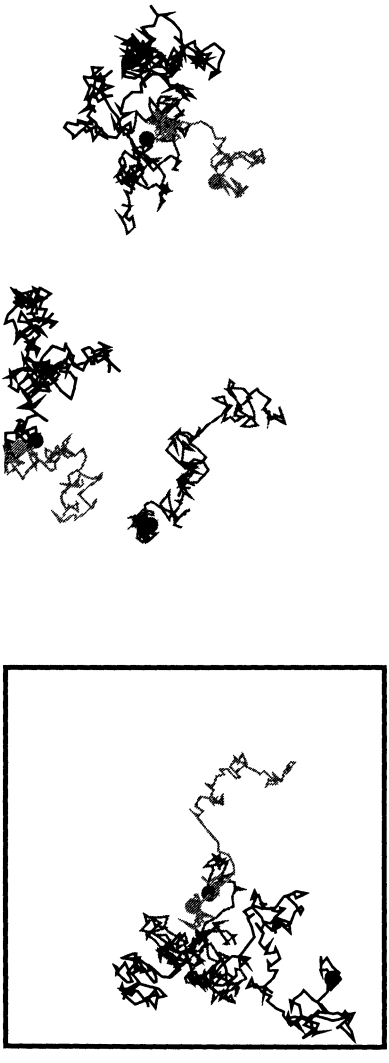
`\lineto`(*coordinates*)

`\rlineto`(*coordinates*)

`\curveto`$(x_1,y_1)(x_2,y_2)(x_3,y_3)$

`\rcurveto`$(x_1,y_1)(x_2,y_2)(x_3,y_3)$

но применение их требует внимательности, поскольку необходимо существование текущей точки. Хорошо владеющие PostScript'ом пользователи могут



```
\usepackage {pstcol,multido}
\input random.tex
\newdimen\X \newdimen\Y\newdimen\Coor
% #1 is number of steps
\newcommand{\RandomWalk}[1]{%
\setrandimm\X{0pt}{1pt}
\setrandimm\Y{0pt}{1pt}
\psdots[dotstyle=*,dotsize=6pt]
  (\pointless\X,\pointless\Y)
\pscustom{%
 \moveto(\pointless\X,\pointless\Y)
 \multido{\i=1+1}{#1}{%
  \MoveCoordinate{\X}
  \MoveCoordinate{\Y}
  \lineto(\pointless\X,\pointless\Y)}}
% Computation of new horizontal
% or vertical position of point
\newcommand{\MoveCoordinate}[1]{%
 \setrandimm\Coor{-0.03pt}{0.03pt}
 \advance #1 by \Coor
 \ifdim#1>1pt #1=1pt \fi
 \ifdim#1<0pt #1=0pt \fi
}
\newcommand{\randomdemo}[2]{{%
 \psframe[linewidth=0.6mm](1,1)
 \rput(#1,#2){\RandomWalk{300}}
 \psset{linecolor=red}
 \rput(#1,#2){\RandomWalk{200}}
 \psset{linecolor=green}
 \rput(#1,#2){\RandomWalk{100}}
}}
\psset{unit=5,dimen=middle}
\framebox{\begin{pspicture}(1,3)
 \randomdemo{0}{0}
 \randomdemo{0}{1}
 \randomdemo{0}{2}
\end{pspicture}}
```

4-10-5

Рис. 4.2. Случайное блуждание, запрограммированное с помощью PSTricks Дени Жиру.

```
\usepackage {pstcol,pst-poly}
\PstRegularPolygon
\PstRegularPolygon[RPolyNbSides=11]
\PstRegularPolygon[RPolyOffset=2]
\PstRegularPolygon[linecolor=red,RPolyIntermediatePoint=0.2]
\PstRegularPolygon[linestyle=dashed,RPolyCurves,RPolyOffset=2,
    RPolyIntermediatePoint=0.75]
\bigskip

\PstRegularPolygon[unit=2,RPolyNbSides=9,RPolyOffset=2]
\PstRegularPolygon[unit=2,RPolyNbSides=27,RPolyOffset=7,RPolyCurves,
  RPolyIntermediatePoint=0.1]
```

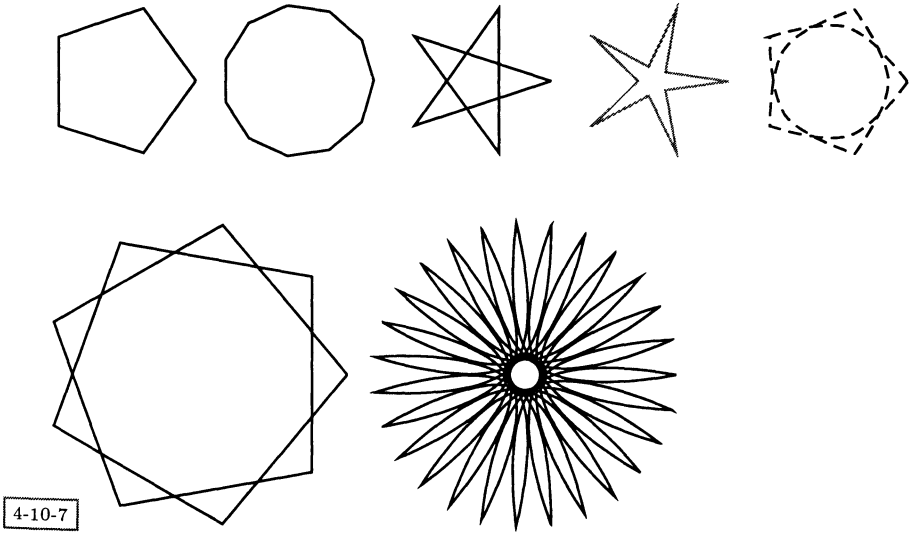


Рис. 4.3. Язык многоугольников, созданный Дени Жиру на основе PSTricks.

также с помощью некоторых средств, предоставленных для реализации заложенных в PSTricks идей, вставлять фрагменты программ непосредственно на PostScript'e. Эти возможности описаны в van Zandt (1993a).

Хотя многие рисунки могут быть запрограммированы с помощью более безопасных средств из основного пакета PSTricks, возможность строить на базе PSTricks новые объекты расширяет диапазон доступных инструментов. На рис. 4.3 показано применение языка многоугольников, написанного Дени Жиру на основе PSTricks (пакет pst-poly).

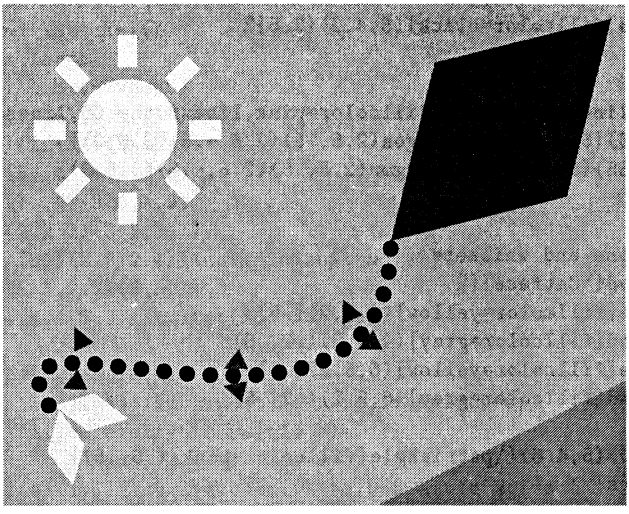
4.10.2 Примеры программирования на PSTricks

В этом разделе мы покажем некоторые из возможностей PSTricks, продемонстрируем преимущества использования языка программирования для рисования и исследуем разнообразные способы упрощения и разбиения текста программы на модули, облегчающие ее чтение. Мы выражаем особую благодарность Дени Жиру за его вклад в написание данного раздела как в плане личного общения, так и в виде опубликованных примеров. Чтобы облегчить обсуждение, строки программ, рисующих эти картинки, пронумерованы.

Обратите внимание, что на первом рисунке, изображающем воздушный змей из детской книжки, хвост нарисован с помощью кривой, соединяющей две вершины, а бантики добавлены в качестве меток этого соединения. Использование свойств вершин подразумевает, что вычисление самой траектории линии и точек вдоль нее целиком возложено на PSTricks. Полезно также вспомнить базовое положение в PostScript'е: объекты являются непрозрачными, если не указано противное. Это означает, что весь рисунок можно разместить на голубом фоне, а потом сверху располагать объекты соответствующей формы и цвета (см. этот рисунок на цветной вклейке Va). Некоторые элементы рисунка являются регулярными и повторяющимися, и поэтому мы воспользовались макро `\multido` для изображения солнечных лучей. Обратите внимание на применение TeX'овского группирования для локализации влияния команды `\psset`, меняющей цвет и стиль.

```
\usepackage {pstcol,multido,pst-node}
\definecolor{lightblue}{rgb}{0, 1, 1}
\psset{unit=.825cm}\begin{pspicture}(10,8)
\psset{fillstyle=solid,linestyle=none,linewidth=0}
\psframe[fillcolor=lightblue](10,8)
\pscircle[fillcolor=yellow](2,6){.8} % Sun
{% Rays
 \psset{linecolor=yellow,linestyle=solid,linewidth=.3}
 \degrees[8]
 \multido{\i=1+1}{8}{\rput{\i}(2,6){\psline(1,0)(1.5,0)}}
}%
\pspolygon[fillcolor=green](6,0)(10,2)(10,0)% Grass
\psdiamond[fillcolor=red,gangle=-45](8,6)(1.5,2.5)% Kite
\rput{45}(8,6){\pnode(-2.5,0){Kitetail}}
\rput{-10}(.8,1.5){\psdiamond[fillcolor=yellow](.6,.1)(.6,.3)}
\rput{-80}(.8,1.5){\psdiamond[fillcolor=yellow](.6,.1)(.6,.3)}
\pnode(.8,1.5){Tailend}
\nccurve[fillstyle=none,angleA=270,angleB=125,ncurvB=.9,ncurvA=1.4,
 linestyle=dotted,dotstyle=square,linewidth=.25]{Kitetail}{Tailend}
\newcommand{\bunting}{\pstriangle(.35,.35)}
\psset{fillcolor=red,labelsep=.01}
\naput[nrot=115,npos=.15]{\bunting}
\nbput[nrot=25,npos=.15]{\bunting}
\naput[nrot=75,npos=.4]{\bunting}
\nbput[nrot=115,npos=.4]{\bunting}
```

```
\naput[nrot=115,npos=.7]{\bunting}
\nbput[nrot=25,npos=.7]{\bunting}
\end{pspicture}
```



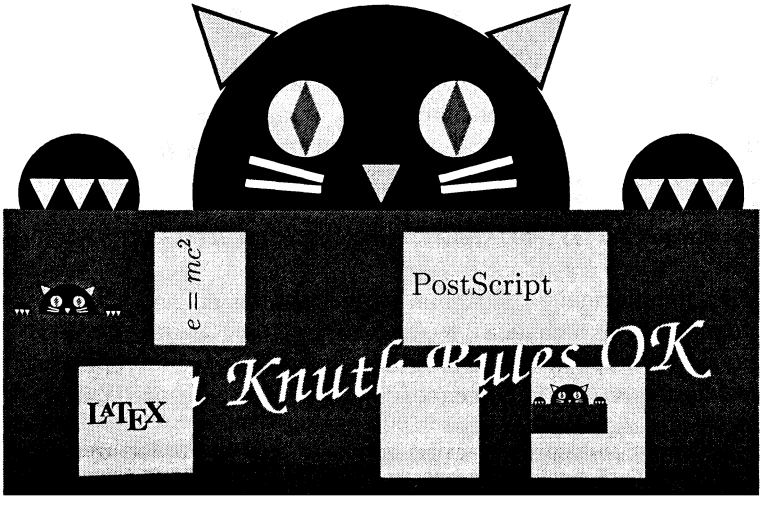
4-10-8

На следующем детском рисунке вновь используется непрозрачная природа PostScript'овских сплошных цветных блоков для изображения кошачьей головы в виде окружности и последующего наложения стены (а поверх нее — кирпичей) так, что нет никакой необходимости беспокоиться о создании подходящего сегмента, немного выходящего за пределы полукруга. Команда `\rput` активно используется для размещения объектов с наклоном. При создании надписей на кирпичах демонстрируется важность слежения за точками привязки размещаемых объектов. Поскольку кирпичи и надписи на них наносятся *после* надписи на стене, они частично закрывают ее. Объекты со сходными характеристиками группируются вместе, а для задания PSTricks'овских параметров для всех элементов внутри этой группы используется TEX'овское группирование. Помимо этого рисунок разбивается на отдельные элементы, описываемые собственными макро и затем группируемые в объекты высокого уровня. Этот прием позволяет создать библиотеку объектов и способствует тому, что итоговое описание рисунка гораздо легче читается. Размер сидящего на стене кота задается некоторым параметром, что дает возможность воспроизвести его несколько раз в разных масштабах. Обратите внимание, что команда `\rput` задает новую систему координат, поэтому кот изображается относительно позиции, заданной в этой команде.

```
\usepackage {pstcol}
\DeclareFixedFont{\curly}{T1}{pzc}{m}{it}{30}
\definecolor{wheat}{rgb}{.96, .87, .7}
\definecolor{brown}{rgb}{.65, .16, .16}
```

```
\definecolor{lightblue}{rgb}{.85, .95, 1}
%
% The cat is designed to appear on a 10 x 10 grid
% cat head
\newcommand{\Cathead}{%
 \pscircle[fillcolor=black](5,4.2){2.5}%
% ears
 {%
  \psset{linecolor=black,fillcolor=pink,linewidth=.05,linestyle=solid}
  \rput{45}(5,4.2){\pspolygon(2.5,.5)(2.5,-.5)(3.5,0)}
  \rput{135}(5,4.2){\pspolygon(2.5,.5)(2.5,-.5)(3.5,0)}
 }%
}
% eyes, nose and whiskers
\newcommand{\Catface}{%
 \pscircle[fillcolor=yellow](4,5.2){.5}
 \psdiamond[fillcolor=gray](4,5.2)(.2,.5)
 \pscircle[fillcolor=yellow](6,5.2){.5}
 \psdiamond[fillcolor=gray](6,5.2)(.2,.5)
% nose
 \rput{180}(5,4.6){\pstriangle[fillcolor=pink](.5,.5)}
 % whiskers
 {%
  \psset{linecolor=white,linestyle=solid,linewidth=.1}
  \rput{5}(5,4.2){\psline(.8,0)(1.8,0)}
  \rput{15}(5,4.2){\psline(.8,0)(1.8,0)}
  \rput{165}(5,4.2){\psline(.8,0)(1.8,0)}
  \rput{175}(5,4.2){\psline(.8,0)(1.8,0)}
 }%
}
% paws with claws
\newcommand{\Catpaws}{%
 \pscircle[fillcolor=black](1,4.2){.8}
 \pscircle[fillcolor=black](9,4.2){.8}
 {%
  \psset{fillcolor=yellow}
  \newcommand{\clawsize}{.4,.4}
  \rput{180}(1,4.4){\pstriangle(\clawsize)}
  \rput{180}(1,4.4){\pstriangle(-.45,0)(\clawsize)}
  \rput{180}(1,4.4){\pstriangle(.45,0)(\clawsize)}
  \rput{180}(9,4.4){\pstriangle(\clawsize)}
  \rput{180}(9,4.4){\pstriangle(-.45,0)(\clawsize)}
  \rput{180}(9,4.4){\pstriangle(.45,0)(\clawsize)}
 }%
}
% wall
\newcommand{\Wall}{%
 \psframe[fillcolor=brown](0,0)(10,4)
}
% The whole cat on its wall
```

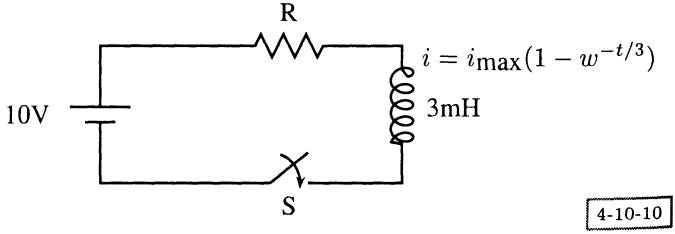
```
\newcommand{\Cat}[1]{%
 {\psset{unit=#1}
  \Cathead\Catface\Catpaws\Wall}%
}
% bricks
\newcommand{\Bricks}{%
 \bfseries\large
 \psset{fillcolor=wheat}
 \psframe(1,.4)(2.5,1.9)
 \rput[bl](1.1,1){\LaTeX}
 \psframe(5,.4)(6.3,1.9)
 \psframe(7,.4)(8.5,1.9)
 \psframe(2,2.2)(3.2,3.7)
 \rput[bl]{90}(2.6,2.4){\normalsize$e=mc^2$}
 \psframe(5.3,2.2)(8,3.7)
 \rput[bl](5.4,2.8){\textsc{PostScript}}
}
\begin{pspicture}(10,8)
 \psset{fillstyle=solid,linestyle=none,linewidth=0}
 \psframe[fillcolor=lightblue](10,8)
 \Cat{1}
 \rput[bl]{5}(1,1){\curly\color{white}Don Knuth Rules OK}
 \Bricks
 \rput(7,1){\Cat{.1}}
 \rput(.1,2){\Cat{.15}}
 \end{pspicture}
```



4-10-9

Третий рисунок (электрическая схема) более сложен. Он нарисован с помощью специально созданного небольшого языка программирования, реализующего необходимые элементы \Battery, \Resistor, \Switch и \Inductor и команды движения \Up, \Down \Left и \Right в стиле языка pic. Поэтому последняя часть программы имеет очень простой вид, не считая команд соединения вершин и размещения меток, используемых привычным образом (некоторая внимательность требуется для учета углов, под которыми соединительные линии входят и выходят из каждой вершины).

```
\Up{1}
\Battery{A}
\Up{1}
\Right{2}
\Resistor{B}
\Right{1}
\Down{.3}
\Inductor{C}
\Down{.5}
\Left{1}
\Switch{D}
\ncangle[angleA=90,angleB=180,armB=0]{A}{B}
\ncangle[angleA=0,angleB=90,armB=0]{B}{C}
\ncput[ref=l]{~~~$i=i_{\mbox{max}}(1-w^{-t/3})$}
\ncangle[angleA=-90,armB=0,angleB=0]{C}{D}
\ncangle[angleA=180,armB=0,angleB=-90]{D}{A}
\nput{180}{A}{10V}
\nput{90}{B}{R}
\nput{0}{C}{3mH}
\nput{270}{D}{S}
```



4-10-10

Эти новые команды определены в терминах текущих координат x и y, которые меняются командами движения. Таким образом, команда \Down (в упрощенном виде) определяется следующей программой:

```
\newcommand{\Cirdirection}{1}
\newcommand{\Down}[1]{%
  \setcounter{CurY}}{-#1}\renewcommand\Cirdirection{3}%
  }
```

Текущее направление движения записывается после каждого перемещения, поскольку от него зависит то, как изображать новые объекты. Для проверки текущего направления в них есть переключатель (представленный в следующей программе в упрощенном виде[1]). Для выбора одного из четырех

[1] Поскольку может возникнуть желание описывать размеры действительными числами, параметры \CurX и \CurY в действительности являются TeX'овскими единицами измерения, которые перед применением снова преобразуются в формат «number».

возможных направлений используется определенная в plain TEX'е команда \ifcase. Таким образом, резистор задается следующим образом:

```
\newcommand{\Resistor}[1]{%
 \ifcase\Cirdirection % right
   \rput[l](\CurX,\CurY){\rnode{#1}{%
   \pszigzag[coilarm=.01,coilwidth=.3](0,.15)(1,.15)%
   \MyBox{1}{.3}}}%
   \addtocounter{CurX}{1}%
 \or % left
   \addtocounter{CurX}{-1}%
   \rput[l](\CurX,\CurY){\rnode{#1}{%
   \pszigzag[coilarm=.01,coilwidth=.3](0,.15)(1,.15)%
   \MyBox{1}{.3}}}%
 \or % up
   \rput[b](\CurX,\CurY){\rnode{#1}{%
   \pszigzag[coilarm=.01,coilwidth=.3](.15,0)(.15,1)%
   \MyBox{.3}{1}}}%
   \addtocounter{CurY}{1}%
 \or % down
   \addtocounter{CurY}{-1}%
   \rput[b](\CurX,\CurY){\rnode{#1}{%
   \pszigzag[coilarm=.01,coilwidth=.3](.15,0)(.15,1)%
   \MyBox{.3}{1}}}%
 \fi
}
```

В этом определении очень важным является макро \MyBox. Поскольку сам по себе результат команды \pszigzag не занимает места, создаваемая этой командой вершина не имеет ни ширины, ни высоты, а соединительные элементы достигают самого центра. Поэтому для создания вокруг зигзагообразной линии невидимого бокса с помощью команды \rule вставляются LATEX'овские подпорки. Команда \pssetlength переводит PSTricks'овские единицы измерения в обычные TEX'овские.

```
\newlength{\Cirtemp}
\newcommand{\MyBox}[2]{% width,height
 \pssetlength{\Cirtemp}{#1}%
 \rule{\Cirtemp}{0pt}%
 \pssetlength{\Cirtemp}{#2}%
 \rule{0pt}{\Cirtemp}%
}
```

В нашем последнем примере PSTricks используется в новой предметной области — картографии. Приведенная карта получена на основе того же созданного программой AutoCAD DXF-файла, что и карта на рис. 1.18. На этот раз 36 многоугольников, состоящих из 9619 отдельных отрезков линий, были преобразованы (с помощью простой программы, написанной *специально* для этой цели) в отдельные файлы, по одному на каждый многоугольник, содер-

жащие данные о координатах. Описание всей карты состоит всего из 36 строк следующего вида:

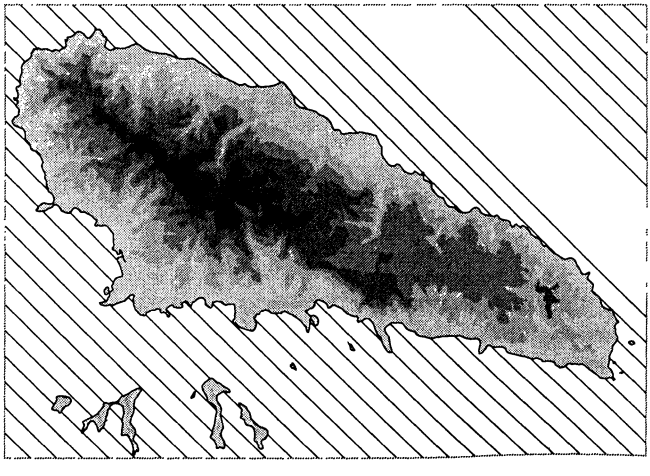
```
\ProcessVector{moh174}
\ProcessVector{moh170}
```

Поскольку число координат так велико, многим инструментам из пакета PSTricks, которые считывают данные из этих файлов (таким, как `\fileplot`), не хватает TeX'овской памяти. Однако для таких чисто графических объектов, как многоугольники, есть более эффективный и менее требовательный к ресурсам способ, состоящий в параллельном создании инкапсулированных PostScript'овских файлов, поэтому содержащие команду `\ProcessVector` строки можно обработать следующим способом:

```
\newcommand{\ProcessVector}[1]{%
 \PSTtoEPS{#1.eps}{\fileplot{#1.dat}}
 \rput(0,0){\includegraphics{#1.eps}}%
}
```

Команда `\PSTtoEPS` имеет два аргумента: имя файла и любые PSTricks'овские команды (т. е. не текст). Вместо вставки в TeX'овский текст необходимых PostScript'овских инструкций в виде команд `\special` PostScript'овский файл создается непосредственно.

Применение очень простой команды `\ProcessVector` высокого уровня означает, что основной файл легко поддается ручному редактированию. Поскольку в файле, созданном программой AutoCAD, многоугольники соответствуют определенному «слою» (карта представляет собой набор контурных линий), цвет заполнения может быть определен для каждого слоя отдельно, и, таким образом, может быть получена карта в более традиционном виде (также представленная на цветной вклейке VIa).



311 4-10-11

4.11 Другие инструменты пакета PSTricks

Мы рассмотрели не все трюки, на которые способен этот пакет. Для полноты картины ниже перечислены остальные свойства:

- Макро, задающие цвета. Мы предпочли использовать стандартный LaTeX'овский пакет color, обеспечивая его взаимодействие с PSTricks при помощи пакета pstcol.
- Макро, выполняющие повороты и изменение масштаба. И здесь мы предпочли пользоваться стандартным LaTeX'овским пакетом graphics. Пользователи должны иметь в виду, что в PSTricks определяется макро `\scalebox`, синтаксис которого отличается от синтаксиса стандартной команды, поэтому следует внимательно следить за тем, в каком порядке загружаются пакеты.
- Прозрачные пленки. Эти макро применяются в пакете seminar, описываемом в разд. 9.4.

4.12 Настройка драйвера для PSTricks

Главному LaTeX'овскому пакету pstricks и некоторым из вспомогательных пакетов требуется доступ к различным PostScript'овским заголовочным файлам. От пользователя не требуется явно загружать их, но драйверу должна быть обеспечена возможность сделать это. Способ загрузки заголовочного файла и других инструментов передачи данных средствами `\special` определяется в конфигурационном файле `pstricks.con`. В этом файле должны быть определены следующие TeX'овские макро (более подробное описание приведено в конфигурационном файле, входящем в PSTricks в качестве примера):

`\pstdriver` задает имя драйвера;

`\pstunit` определяет единицы измерения, используемые драйвером при дословном включении PostScript'а;

`\pstverbscale` записывает текст на PostScript'е, выполняющий изменение масштаба системы координат при переходе от той, что используется в команде `\pstVerb`, к той, что используется в команде `\pstverb`;

`\pstverb` осуществляет дословное включение вводимого в качестве аргумента текста на PostScript'е, ограничивая его инструкциями save и restore;

`\pstVerb` осуществляет дословное включение вводимого в качестве аргумента текста на PostScript'е, *не* ограничивая его инструкциями save и restore;

`\pstheader` осуществляет включение аргумента в виде PostScript'овского заголовочного файла;

`\pstrotate` записывает PostScript'овские инструкции, извлекающие из стека число и поворачивающие оси.

Итак, конфигурация драйвера dvips выглядит следующим образом:

```
\def\pstdriver{Rokicki's dvips}
{\catcode`\"=12\gdef\pstverb#1{\special{" #1}}}
\def\pstunit{1bp}%
\def\pstVerb#1{\special{ps: #1}}
\def\pstverbscale{SDict begin normalscale end}
\def\pstheader#1{\special{header=#1}}
%% In the following, CP is short for currentpoint.
%% The neg is necessary because dvips scales the coordinates by 1 -1.
%% \tx@NET means neg exch neg exch translate.
\def\pstrotate{%
  CP CP translate 3 -1 roll neg rotate \tx@NET}
```

4.13 Перечень команд и параметров пакета PSTricks

4.13.1 Базовые команды пакета PSTricks для рисования

Для использования этих команд загрузите пакет pstricks (или pstcol, если вам требуется поддержка работы с цветом). Кроме того, загрузите pst-text, если вам нужен макро \pstextpath, pst-char, если вам нужен \pscharclip или \pscharpath, и pst-coil, если вам нужен \pscoil. Чтобы воспользоваться «плавным» изменением цвета закрашивания, загрузите pst-grad.

\parabola*[*settings*] {*arrows*} $(x_0,y_0)(x_1,y_1)$
Проводится парабола, начинающаяся в точке (x_0,y_0) и достигающая максимального или минимального значения в точке (x_1,y_1)

\psarc*[*settings*] {*arrows*} (x,y) {*radius*}{*angleA*}{*angleB*}
Проводится дуга окружности, заключенная между углами *angleA* и *angleB* (отсчитываемыми против часовой стрелки)

\psarcn*[*settings*] {*arrows*} (x,y) {*radius*}{*angleA*}{*angleB*}
Команда аналогична \psarc, но дуга окружности проводится *по часовой стрелке*

\psbezier*[*settings*] {*arrows*} $(x_0,y_0)(x_1,y_1)(x_2,y_2)(x_3,y_3)$
На основе четырех контрольных точек проводится кривая Безье

\psccurve*[*settings*] {*arrows*} $(x_1,y_1)\ldots(x_n,y_n)$
Точки соединяются замкнутой кривой

\pscharclip*[*settings*] {*text*} ... \endpscharclip
Задается отсекающая траектория в форме букв текста

\pscharpath*[*settings*] {*text*}
Набор текста *text* подчиняется PSTricks'овским параметрам *linestyle* и *fillstyle*. Это справедливо только при использовании PostScript'овских шрифтов Type1

\pscircle*[*settings*] (x_0,y_0) {*radius*}
Проводится окружность радиуса *radius* с центром в точке (x_0,y_0)

\pscirclebox*[*settings*] {*text*}
Окружность проводится вокруг текста *text*

\pscoil*[*settings*] {*arrows*} (x_0,y_0) (x_1,y_1)
Проводится трехмерная витая линия из точки (x_0,y_0) в точку (x_1,y_1)

\psCoil*[*settings*] {*angle1*}{*angle2*}
Витая линия проводится горизонтально от угла *angle1* до *angle2*

\pscurve*[*settings*] {*arrows*} $(x_1,y_1)\ldots(x_n,y_n)$
Через точки проводится незамкнутая кривая

\psdblframebox*[*settings*] {*text*}
Текст *text* обводится двойной рамкой

\psdiabox*[*settings*] {*text*}
Текст *text* обводится рамкой в форме ромба

\psdiamond (x_0,y_0) (x_1,y_1)
Проводится ромб с центром в точке (x_0,y_0), половина ширины которого задается точкой (x_1,y_1), а высота — величиной y_1

\psdots*[*settings*] (x_1,y_1) (x_2,y_2) ... (x_n,y_n)
В каждой паре координат рисуется точка

\psecurve*[*settings*] {*arrows*} $(x_0,y_0)\ldots(x_n,\ y_n)$
Набор точек, за исключением первой и последней, соединяется незамкнутой кривой

\psellipse*[*settings*] $(x_0,y_0)(x_1,y_1)$
Проводится эллипс с центром в точке (x_0,y_0) и вертикальным и горизонтальным радиусами x_1 и y_1

\psframe*[*settings*] (x_0,y_0) (x_1,y_1)
Проводится прямоугольная рамка с углами в точках (x_0,y_0) и (x_1,y_1)

\psframebox*[*settings*] {*text*}
Вокруг текста *text* проводится рамка

\psgrid (x_0,y_0) (x_1,y_1) (x_2,y_2)
Накладывается сетка с углами в точках (x_1,y_1) и (x_2,y_2) и метками на осях, начинающимися с точки (x_0,y_0). По умолчанию сетка охватывает все пространство, занимаемое соответствующей командой pspicture

\psline*[*settings*] {*arrows*} (x_0,y_0) $(x_1,y_1)\ldots(x_n,y_n)$
Через набор точек проводятся прямые линии

\psovalbox*[*settings*] {*text*}
Текст *text* окружается овалом

\pspolygon*[*settings*] (x_0,y_0) $(x_1,y_1)(x_2,y_2)\ldots(x_n,y_n)$
Через набор точек проводятся прямые линии, после чего линия замыкается для создания объекта, который можно закрасить

\psshadowbox*[*settings*] {*text*}
Вокруг текста *text* строится бокс с тенью

\pstextpath [*pos*] (x,y) {*graphics object*} {*text*}
Текст *text* размещается вдоль траектории, заданной параметром *graphics object*. Параметр *pos* определяет положение текста на траектории. В случае принимаемого по умолчанию значения (l) текст начинает набираться от начала траектории. Значение c центрирует текст относительно траектории, а значение r помещает его в конце траектории. Параметр x,y задает отступ текста от траектории. По умолчанию текст приподнят над линией на 0.7ex. *Не всякий драйвер, преобразующий* dvi *в PostScript, работает с этим макро и командой* \pscharclip!

\pstriangle*[*settings*] (x_0,y_0) (x_1,y_1)
Проводится равнобедренный треугольник, центр основания которого расположен в точке (x_0,y_0), ширина задается величиной x_1 а высота — y_1

\pstribox*[*settings*] {*text*}
Вокруг текста *text* строится треугольник

`\pswedge*[`*settings*`]` (x_0,y_0) `{`*radius*`}{`*angle1*`}{`*angle2*`}`
Рисуется сегмент окружности радиуса *radius*, заключенный между углами *angle1* и *angle2* (против часовой стрелки)

`\pszigzag*[`*settings*`]` `{`*arrows*`}` (x_0,y_0) (x_1,y_1)
Из точки (x_0,y_0) в точку (x_1,y_1) проводится зигзагообразная линия

`\qdisk(`x,y`){`*radius*`}`
Быстрая версия команды `\pscircle` без необязательных параметров

`\qline(`x_1,y_1`)(`x_2,y_2`)`
Быстрая версия команды `\psline` для рисования одного отрезка прямой линии без необязательных параметров

Таблица 4.5. Основные графические параметры пакета PSTricks.

Параметр	*По умолчанию*	*Описание*
Общие		
unit=*dim*	1cm	Размер основной единицы длины в PSTricks
xunit=*dim*	1cm	Размер единицы длины в PSTricks для оси x
yunit=*dim*	1cm	Размер единицы длины в PSTricks для оси y
runit=*dim*	1cm	Размер других единиц длины в PSTricks
showpoints=*true/false*	false	Если установлен равным *true*, то будут показаны контрольные точки большинства объектов. (Представляет особый интерес при использовании кривой Безье)
origin={*coor*}	0pt,0pt	Начало координат координатной системы последующих графических объектов
dimen=*where*	outer	Положение, откуда измеряются расстояния в замкнутых объектах. Возможными значениями могут быть `outer`, `inner` и `middle`
swapaxes=*true*	false	Оси x и y меняются местами
linewidth=*dim*	0.8pt	Ширина линий
linecolor=*color*	black	Цвет линий
fillcolor=*color*	white	Цвет закрашивания объектов
fillstyle=*style*	none	Другими возможными стилями закрашивания являются `solid`, `vlines`, `vlines*`, `hlines`, `hlines*`, `crosshatch` и `crosshatch*`. Версии со звездочкой также заполняют фон. Пакет pst-grad добавляет еще один стиль `gradient` для закрашивания с плавным переходом цветов из одного в другой, имеющий следующие ключи:
gradbegin=*color*	0.0 0.1 0.95	Начальный и конечный цвета для заполнения с плавным переходом
gradend=*color*	0 1 1	Цвет в средней точке
gradlines=*int*	300	Количество линий при заполнении с плавным переходом. Большее число линий означает более плавный переход, но более медленную печать

Основные графические параметры пакета PSTricks (*продолжение*)

Параметр	*По умолчанию*	*Описание*
gradmidpoint=*num*	0.9	Положение средней точки в виде доли расстояния от вершины до основания (значение параметра *num* лежит между 0 и 1)
gradangle=*angle*	0	Градация поворачивается на угол *angle*
hatchwidth=*dim*	0.8pt	Ширина линий для заполнения
hatchsep=*dim*	4pt	Расстояние между линиями для заполнения
hatchcolor=*color*	black	Цвет штриховых линий
hatchangle=*angle*	45	Угол проведения штриховых линий
arrows=*style*	none	Возможные варианты обозначения стилей перечислены в табл. 4.3 на с. 132. На каждом конце линии может быть любой из этих символов или вообще никакого
Прямые и кривые линии, боксы		
linestyle=*style*	solid	Другими возможными значениями являются `dashed`, `dotted` и `none`
dash=*dim1 dim2*	5pt 3pt	Черный/белый шаблон для рисования штриховых линий
dotsep=*dim*	3pt	Расстояние между точками
border=*dim*	0pt	Проводится граница ширины *dim* и цвета *bordercolor* (см. следующий параметр) с каждой стороны от прямой или кривой
bordercolor=*color*	white	Цвет граничной линии
doubleline=*true/false*	false	Линии рисуются двойными, на расстоянии *doublesep*, а этот промежуток заполняется цветом *bordercolor*
doublesep=*dim*		Значение по умолчанию составляет 1.25 × текущее значение `linewidth`
doublecolor=*color*	white	Цвет линий, если параметр *doubleline* имеет значение «true»
shadow=*true/false*	false	Рисуется тень под углом *shadowangle*, имеющая глубину *shadowsize* и цвет *shadowcolor*
shadowsize=*dim*	3pt	Глубина отступа тени
shadowangle=*angle*	-45	Угол, под которым падает тень
shadowcolor=*color*	darkgray	Цвет тени
linearc=*dim*	0pt	Радиус дуги, опирающейся на углы бокса или набор отрезков
framearc=*dim*	0pt	Если значение параметра *cornersize* равно «relative», то радиус закругленных углов боксов, играющих роль рамок, устанавливается равным величине *num*, умноженной на минимум из двух величин, а именно ширины или высоты данной рамки. Значение *num* не может быть больше 1. Если значение параметра *cornersize* равно «absolute», то величина *num* задает радиус закругленных углов рамок

Основные графические параметры пакета PSTricks (*продолжение*)

Параметр	*По умолчанию*	*Описание*
cornersize=*type*	relative	возможными значениями могут быть «relative» и «absolute» (используется вместе с параметром *framearc*, см. выше)
Дуги		
arcsepA=*dim*	0pt	Этот и два следующих параметра позволяют корректировать углы команды `\psarc` таким образом, чтобы дуга лишь коснулась прямой линии, выходящей из центра под углом *dim*
arcsepB=*dim*	0pt	(см. предыдущее описание)
arcsep=*dim*	0pt	(см. предыдущее описание)
Кривые линии		
curvature=*a b c*	1 .1 0	Меньшие значения аргумента *a* делают кривую более натянутой. Меньшие значения аргумента *b* делают кривую более натянутой, если угол, образованный тремя точками, превосходит 45°, и более свободной в противном случае. Аргумент *c* определяет наклон в каждой точке
Рамки вокруг текста		
framesep=*dim*	3pt	Расстояние между рамкой и заключенным в ней текстом
boxsep=*true/false*	true	Включает или нет полученный TeX'овский текст размер самой рамки
Текстовые метки		
labelsep=*dim*	5pt	Сдвиг текста внутри команды `\uput`
Точки		
dotstyle=*style*	*	Возможные стили перечислены в табл. 4.4 на с. 135
dotsize=*dim num*	2pt 2	Диаметр окружности или круга равен *dim* плюс *num* раз текущая ширина линии
dotscale=*num1 num2*	1	Размер точек меняется в *num1* раз в горизонтальном направлении и в *num2* раз в вертикальном направлении
dotangle=*angle*	0	Угол поворота каждой точки
Стрелки-указатели		
arrowsize=*dim num*	1.5pt 2	Ширина стрелки-указателя равна *dim* плюс *num*, умноженному на текущую ширину линии
arrowlength=*num*		Длина стрелки-указателя равна параметру *arrowlength*, умноженному на ширину
arrowinset=*num*		Длина тела стрелки равна доле *arrowinset* от длины головы стрелки
tbarsize=*dim num*		Ширина Т-образного конца, квадратной или круглой скобки равна *num* × *linewidth* плюс *dim*

Основные графические параметры пакета PSTricks (*продолжение*)

Параметр	*По умолчанию*	*Описание*
bracketlength=*num*		Высота квадратной скобки устанавливается равной *num*, умноженному на ее ширину
rbracketlength=*num*		Высота круглой скобки устанавливается равной *num*, умноженному на ее ширину
arrowscale=*num1 num2*		Ширина стрелок-указателей умножается на *num1*, а длина — на *num2*. Если параметр *num2* опущен, то размеры стрелок изменяются одинаково в обоих направлениях
Ромбы и треугольники		
gangle=*angle*	0	Угол поворота ромбов и треугольников
Сетки		
gridwidth=*dim*	0.8pt	Ширина линий сетки
gridcolor=*color*	black	Цвет линий сетки
griddots=*num*	0	Если параметр *num* положителен, то линии сетки проводятся пунктиром, нанося *num* точек на одно деление шкалы
gridlabels=*dim*	10pt	Размер цифр, используемых для разметки сетки
gridlabelcolor=*color*	black	Цвет цифр в метках
subgriddiv=*int*	5	Количество вторичных делений на шкале сетки
subgridwidth=*dim*	0.4pt	Ширина линий вторичной сетки
subgridcolor=*color*	gray	Цвет линий вторичной сетки
subgriddots=*num*	0	Количество точек в одном делении вторичной сетки
Витые и зигзагообразные линии		
coilwidth=*dim*	1cm	Диаметр витков или зигзагов
coilheight=*num*	1	Длина каждого витка или зигзага равна *coilwidth* × *coilheight*
coilarm=*dim*	0.5cm	Длина прямолинейного отрезка на концах витой или зигзагообразной линии. С помощью двух следующих параметров их можно устанавливать отдельно
coilarmA=*dim*	0.5cm	(см. предыдущее описание)
coilarmB=*dim*	0.5cm	(см. предыдущее описание)
coilaspect=*angle*	45	Угол зрения на трехмерные объекты
coilinc=*angle*	10	Гладкость витых линий

4.13.2 Команды создания вершин пакета PSTricks

Создание вершин

Приведенные ниже команды, а также команды из следующего раздела становятся доступны после загрузки пакета pst-node.

`\rnode [`*refpoint*`] {`*name*`}{`*text*`}`

создается вершина под именем *name*, содержащая текст *text*. Соединительные элементы указывают на точку *refpoint*

`\Rnode` (x,y) {*name*}{*text*}
команда аналогична `\rnode`, но точка привязки сдвинута на (x,y) относительно центра базовой линии бокса

`\pnode` (x,y) {*name*}
в точке (x,y) создается вершина, не занимающая места

`\dotnode*`[*settings*] (x,y) {*name*}{*text*}
команда похожа на `\psdots`, но создает вершину

`\fnode*`[*settings*] (x,y) {*name*}{*text*}
команда похожа на `\psframe`, но создает вершину

`\cnode*`[*settings*] (x,y) {*radius*}{*name*}
создается вершина, имеющая вид окружности радиуса *radius*

`\Cnode*`[*settings*] (x,y) {*name*}
используя величину радиуса, установленную с помощью графического параметра *radius*, создается вершина, имеющая вид окружности (иногда оказывается удобно задать радиус сразу для многих окружностей)

`\cnodeput*`[*settings*] {*angle*} (x,y) {*name*}{*text*}
создается вершина, имеющая вид обведенного окружностью текста *text*, наподобие `\cput`

`\circlenode*`[*settings*] {*name*}{*text*}
команда похожа на `\pscirclebox`, но создает вершину

`\ovalnode*`[*settings*] {*name*}{*text*}
команда похожа на `\psovalbox`, но создает вершину

`\dianode*`[*settings*] {*name*}{*text*}
команда похожа на `\psdiabox`, но создает вершину

`\trinode*`[*settings*] {*name*}{*text*}
команда похожа на `\pstribox`, но создает вершину

Соединительные элементы

`\ncline*`[*settings*] {*arrows*} {*firstnode*}{*secondnode*}
прямая линия между вершинами

`\ncLine*`[*settings*] {*arrows*} {*firstnode*}{*secondnode*}
прямая линия между вершинами, но метки размещаются таким образом, будто линия достигает центров вершин

`\ncarc*`[*settings*] {*arrows*} {*firstnode*}{*secondnode*}
вершины соединяются дугой. Использует параметр *arcangle*

`\ncdiag*`[*settings*] {*arrows*} {*firstnode*}{*secondnode*}
используя значения параметров *arm* и *angle*, из каждой вершины проводятся «плечи», которые потом соединяются прямой линией. Форма угла контролируется параметром *linearc*

`\ncdiagg*`[*settings*] {*arrows*} {*firstnode*}{*secondnode*}
команда похожа на `\ncdiag`, но второе плечо не проводится

`\ncbar*`[*settings*] {*arrows*} {*firstnode*}{*secondnode*}
под углом *angleA* проводится линия, плечи которой выходят из вершин под прямыми углами. В случае необходимости длина плеч корректируется.

\ncangle*[*settings*] {*arrows*} {*firstnode*}{*secondnode*}
проводится соединительный элемент, напоминающий результат команды \ncdiag, но угол между плечом A и соединительной линией делается прямым

\ncangles*[*settings*] {*arrows*} {*firstnode*}{*secondnode*}
команда похожа на \ncangle, но плечо A соединяется с плечом B двумя отрезками прямых линий, встречающихся под прямым углом

\ncloop*[*settings*] {*arrows*} {*firstnode*}{*secondnode*}
команда похожа на \ncangles, но используются пять отрезков прямых линий, второй и четвертый из которых имеют длину *loopsize*

\nccurve*[*settings*] {*arrows*} {*firstnode*}{*secondnode*}
между двумя вершинами проводится кривая Безье, причем для указания контрольных точек используются параметры *ncurv*

\nccircle*[*settings*] {*arrows*} {*node*}{*radius*}
проводится окружность или дуга окружности радиуса *radius*, соединяющая вершину саму с собой

Витые и зигзагообразные соединительные элементы

Для использования этих дополнительных соединительных элементов загрузите пакет pst-coil.

\nccoil*[*settings*] {*arrows*} {*firstnode*}{*secondnode*}
две вершины соединяются витой линией

\nczigzag*[*settings*] {*arrows*} {*firstnode*}{*secondnode*}
две вершины соединяются зигзагообразной линией

4.13.3 Команды пакета PSTricks, помещающие метки на соединительных элементах

Размещение меток, использующее при расчетах положения текста длину соединительного элемента

\ncput*[*settings*] {*something*}
текст *something* помещается рядом с соединительной линией

\naput*[*settings*] {*something*}
текст *something* помещается над соединительной линией

\nbput*[*settings*] {*something*}
текст *something* помещается под соединительной линией

Размещение меток, использующее при расчетах положения текста расстояние между вершинами

\tvput*[*settings*] {*something*}
применяется в случае, когда вершины отстоят друг от друга по вертикали. Текст *something* помещается в центре линии

`\tlput*[settings]{something}`
применяется в случае, когда вершины отстоят друг от друга по вертикали. Текст *something* помещается слева от линии

`\trput*[settings]{something}`
применяется в случае, когда вершины отстоят друг от друга по вертикали. Текст *something* помещается справа от линии

`\thput*[settings]{something}`
применяется в случае, когда вершины отстоят друг от друга по горизонтали. Текст *something* помещается в центре линии

`\taput*[settings]{something}`
применяется в случае, когда вершины отстоят друг от друга по горизонтали. Текст *something* помещается над линией

`\tbput*[settings]{something}`
применяется в случае, когда вершины отстоят друг от друга по горизонтали. Текст *something* помещается под линией

Присвоение меток вершинам

`\nput[settings]{angle}{name}{something}`
Текст *something* помещается рядом с вершиной *name* на расстоянии *nodesep* в направлении *angle* от центра вершины

Таблица 4.6. Графические параметры соединительных элементов пакета PSTricks.

Параметр	*По умолчанию*	*Описание*
offset=*dim*	0pt	Сдвиг точки соединения с вершиной
nodesep=*dim*	0pt	Поле вокруг вершин, на которое не заходят соединительные элементы
nodesepA=*dim*	0pt	Поле вокруг первой вершины
nodesepB=*dim*	0pt	Поле вокруг второй вершины
arcangle=*angle*	8	В команде `\ncarc` угол между дугой и прямой линией, соединяющими вершины
angle=*angle*	0	Угол, под которым соединительные элементы касаются вершин
angleA=*angle*	0	Угол, под которым соединительный элемент касается первой вершины
angleB=*angle*	0	Угол, под которым соединительный элемент касается второй вершины
arm=*dim*	10pt	Длина отрезка, являющегося плечом соединяющего вершины элемента
armA=*dim*	10pt	Длина отрезка, являющегося плечом соединительного элемента около первой вершины
armB=*dim*	10pt	Длина отрезка, являющегося плечом соединительного элемента около второй вершины
loopsize=*dim*	1cm	Длина отрезков в команде `\ncloop`

Графические параметры соединительных элементов пакета PSTricks (*продолжение*).

Параметр	*По умолчанию*	*Описание*
ncurv=*num*	0.67	Расстояние до контрольных точек кривой Безье в команде `\nccurve`. Меньшие значения увеличивают натяжение кривой. Расстояние от вершины до первой контрольной точки равно половине *ncurv* × расстояние между двумя крайними точками
ncurvA=*num*	0.67	параметр похож на *ncurv*, но относится только к первой вершине
ncurvB=*num*	0.67	параметр похож на *ncurv*, но относится только ко второй вершине
boxsize=*dim*	0.4cm	половина ширины играющего роль рамки бокса в командах `\ncbox` и `\ncarcbox`
Параметры для размещения меток		
ref=*ref*	c	Задает точку привязки метки
nrot=*rot*	0	Поворот текста метки. Если величине поворота предшествует символ :, то она измеряется относительно соединительной линии. Сокращенные обозначения приведены в табл. 4.2 на с. 130. Распространенным способом выравнивания текста на соединительной линии является последовательность `:U`
npos=*num*		Положение на соединительной линии, в котором помещается метка. Каждая соединительная линия состоит из одного или более сегментов и величина *npos*+1 определяет тот сегмент, на котором помещается метка. Значения по умолчанию см. в руководстве к пакету PSTricks
shortput=*type*	none	Определяет, можно ли применять сокращенные обозначения для нанесения меток на соединительные элементы. Возможными значениями являются `none`, `nab`, `tablr` или `tab`. Обозначения сокращений см. на с. 143
tpos=*num*	0.5	Доля расстояния между вершинами, где соединительному элементу присваивается метка
mnode=*type*	R	(для матриц) принятый по умолчанию тип вершин. Возможными значениями могут быть R (`\Rnode`), r (`\rnode`), C (`\Cnode`), f (`\fnode`), p (`\pnode`), circle (`\circlenode`), oval (`\ovalnode`), dia (`\dianode`), tri (`\trinode`), dot (`\dotnode`) и none
emnode=*type*	none	(для матриц) тип вершины, создаваемый в пустых ячейках матрицы
name=*name*		(для матриц) имя вершины. Параметры наподобие этого могут быть заданы в квадратных скобках в самой ячейке
nodealign=*true/false*	false	(для матриц) проходят ли базовые линии вершин через их центры

Графические параметры соединительных элементов пакета PSTricks (*продолжение*):

Параметр	*По умолчанию*	*Описание*
mcol=*l/r c*	c	для матриц) выравнивание вершины внутри ячейки матрицы
mnodesize=*dim*	-1pt	(для матриц) если величина положительна, то размеры вершин делаются равными этому значению в обязательном порядке
rowsep=*dim*	1.5cm	(для матриц) расстояние между строками
colsep=*dim*	1.5cm	(для матриц) расстояние между столбцами

4.13.4 Команды пакета PSTricks для рисования, аналогичные командам для соединения вершин

`\pcline*[`*settings*`] {`*arrows*`}` $(x_1, y_1)(x_2, y_2)$
`\pccurve*[`*settings*`] {`*arrows*`}` $(x_1, y_1)(x_2, y_2)$
`\pcarc*[`*settings*`] {`*arrows*`}` $(x_1, y_1)(x_2, y_2)$
`\pcbar*[`*settings*`] {`*arrows*`}` $(x_1, y_1)(x_2, y_2)$
`\pcdiag*[`*settings*`] {`*arrows*`}` $(x_1, y_1)(x_2, y_2)$
`\pcangle*[`*settings*`] {`*arrows*`}` $(x_1, y_1)(x_2, y_2)$
`\pcloop*[`*settings*`] {`*arrows*`}` $(x_1, y_1)(x_2, y_2)$
`\pczigzag*[`*settings*`] {`*arrows*`}` $(x_1, y_1)(x_2, y_2)$
`\pccoil*[`*settings*`] {`*arrows*`}` $(x_1, y_1)(x_2, y_2)$

4.13.5 Команда пакета PSTricks для построения деревьев

Если вы хотите воспользоваться этими командами, загрузите пакет pst-tree.

`\pstree{`*node*`}{`*subtrees*`}`
рисуется узел и соединенные с ним поддеревья

`\psTree{`*rootnode*`}` *subtrees* `\endpsTree`
вариант команды `\pstree` в виде окружения

`\Tn`
узел нулевого дерева

`\Tspace{`*dim*`}`
перед следующим уровнем оставляется расстояние *dim*

`\TC*[`*settings*`]`
узел дерева, аналогичный вершине `\Cnode`

`\TR*[`*settings*`] {`*something*`}`
узел дерева, аналогичный вершине `\Rnode`

`\Tcircle*[`*settings*`] {`*something*`}`
узел дерева, аналогичный вершине `\circlenode`

`\Tc*[`*settings*`] {`*dim*`}`
узел дерева, аналогичный вершине `\cnode`

`\Tdia*[settings]{something}`
узел дерева, аналогичный вершине `\dianode`

`\Tdot*[settings]`
узел дерева, аналогичный вершине `\dotnode`

`\Tf*[settings]`
узел дерева, аналогичный вершине `\fnode`

`\Tfan*[settings]`
рисуется треугольник, верхняя вершина которого помещается в предыдущем узле

`\Toval*[settings]{something}`
узел дерева, аналогичный вершине `\ovalnode`

`\Tp*[settings]`
узел дерева, аналогичный вершине `\pnode`

`\Tr*[settings]{something}`
узел дерева, аналогичный вершине `\rnode`

`\Ttri*[settings]{something}`
узел дерева, аналогичный вершине `\trinode`

`\skiplevel*[settings]` *nodes or subtrees*
пропускаются целые уровни определенного поддерева

`\skiplevels*[settings]{n}` *nodes or subtrees*
пропускаются n уровней

Таблица 4.7. Графические параметры пакета PSTricks для деревьев.

Параметр	*По умолчанию*	*Описание*
bbd=*dim*		Нижняя граница обрамляющего бокса устанавливается равной *dim*
bbh=*dim*		Верхняя граница обрамляющего бокса устанавливается равной *dim*
bbl=*dim*		Левая граница обрамляющего бокса устанавливается равной *dim*
bbr=*dim*		Правая граница обрамляющего бокса устанавливается равной *dim*
edge=*command*		Элемент, используемый для соединения узлов дерева
fansize=*dim*	1cm	Размер основания, используемый командой `\Tfan`
levelsep=**dim*	2cm	Расстояние между последовательно идущими уровнями дерева. Вариант со звездочкой *добавляет* этот размер к размеру самих узлов (обычно уровни расположены на фиксированном расстоянии друг от друга)
showbbox=*true/false*	false	Рисуется точечная рамка, соответствующая обрамляющему боксу для дерева
thislevelsep=**dim*		Параметр аналогичен *levelsep*, но применяется только к текущему дереву
thistreefit=*tight/loose*		Параметр аналогичен *treefit*, но применяется только к текущему дереву

Графические параметры пакета PSTricks для деревьев (*продолжение*).

Параметр	*По умолчанию*	*Описание*
thistreenodesize=*dim*		Параметр аналогичен *treenodesize*, но применяется только к текущему дереву
thistreesep=*dim*		Параметр аналогичен *treesep*, но применяется только к текущему дереву
tndepth=*dim*		Минимальная глубина меток узлов
tnheight=*dim*		Минимальная высота меток узлов
tnpos=*l/r/a/b*	b	Положение меток узлов относительно самого узла (l (слева), r (справа), a (сверху), b (снизу))
tnsep=*dim*		Расстояние между меткой узла и самим узлом (по умолчанию оно равно *labelsep*)
treefit=*tight/loose*	tight	Если значение равно «tight», то параметр *treesep* равен минимальному расстоянию между узлами любого уровня. Если значение равно «loose», то параметр *treesep* равен расстоянию между боксами, обрамляющими поддеревья
treeflip=*true/false*	false	Создается зеркальное отражение дерева, узлы переворачиваются
treemode=*R/L/U/D*	D	Направление роста дерева (R (направо), L (налево), U (вверх) и D (вниз))
treenodes=*dim*	-1pt	Положительное значение устанавливает фиксированный размер узлов дерева независимо от их содержимого
treesep=*dim*	0.75cm	Расстояние между последовательно идущими узлами дерева
xbbd=*dim*		Нижняя граница обрамляющего бокса увеличивается на *dim*
xbbh=*dim*		Верхняя граница обрамляющего бокса увеличивается на *dim*
xbbl=*dim*		Левая граница обрамляющего бокса увеличивается на *dim*
xbbr=*dim*		Правая граница окаймляющего бокса увеличивается на *dim*

4.13.6 Команды пакета PSTricks для графического отображения данных

Если вы хотите использовать эти команды, загрузите пакет pst-plot.

`\fileplot*[`*settings*`]{`*file*`}`

Команда считывает данные из файла *file* и отображает их. Поддерживает только стили «line», «polygon» и «dot»

`\dataplot*[`*settings*`]{`*command*`}`

Отображаются данные, предварительно сохраненные с помощью команд `\readdata` или `\savedata`

`\savedata{`*command*`}[`*data*`]`

Данные *data* сохраняются в блоке с именем `\command` для последующей обработки

командой \dataplot. Эти данные обязательно должны быть заключены в квадратные скобки

\readdata{*command*}{*file*}
Данные считываются из файла *file* и сохраняются в блоке с именем \command для последующей обработки командой \dataplot

\listplot*[*settings*] {*file*}
Данные считываются из файла *file* и отображаются, причем в любом стиле, но эта команда выполняет меньше проверок при создании PostScript'овского кода

\psplot*[*settings*] {x_{min}}{x_{max}}{*function*}
Отображается функция *function* $f(x)$. Эта команда написана на языке PostScript с x в качестве зависимой переменной

\parametricplot*[*settings*] {t_{min}}{t_{max}}{*function*}
Отображается параметрическая функция *function* $(x(t); y(t))$

Оси

\psaxes*[*settings*] {*arrows*} (x_0, y_0) (x_1, y_1) (x_2, y_2)
На графике рисуются оси, пересекающиеся в точке (x_0, y_0). По умолчанию точки (x_0, y_0) и (x_1, y_1) считаются равными (0,0). Длины осей задаются параметрами x_2 и y_2

\pshlabel
Макро с одним параметром для переопределения вида и размера шрифта для набора меток на оси x. Таким образом, команда \renewcommand{\pshlabel}[1] {{\tiny #1}} задает набор меток шрифтом размера «tiny»

\psvlabel
Макро переопределяет вид и размер шрифта для набора меток на оси y

Таблица 4.8. Графические параметры пакета PSTricks для отображения данных.

Параметр	*По умолчанию*	*Описание*
plotstyle=*style*	line	Допустимыми значениями являются «dots», «line», «polygon», «curve», «ecurve», «ccurve»
plotpoints=*int*	50	Число точек, используемых на диаграмме
Оси		
axesstyle=*axes/frame/none*	axes	
labels=*all/x/y/none*	all	На какие оси наносить метки?
showorigin=*true/false*	true	Помещать ли метки в начале координат?
tickstyle=*full/top/bottom*	full	Рисуются ли риски на шкале выше или ниже оси?
ticks=*all/x/y/none*	all	На какие оси наносить риски?
ticksize=*dim*	3pt	
Ox=*num*	0	Значение метки в начале оси x
Oy=*num*	0	Значение метки в начале оси y
Dx=*num*	1	Шаг шкалы, на который увеличиваются метки на оси x

Графические параметры пакета PSTricks для отображения данных (*продолжение*).

Параметр	*По умолчанию*	*Описание*
Dy=*num*	1	Шаг шкалы, на который увеличиваются метки на оси y
dx=*dim*	0pt	Расстояние между метками на оси x (если значение равно 0pt, используется текущая единица измерения оси x)
dy=*dim*	0pt	Расстояние между метками на оси y (если значение равно 0pt, используется текущая единица измерения оси y)

4.13.7 Команды пакета PSTricks для работы с третьим измерением

Если вы хотите использовать эти команды, загрузите пакет pst-3d.

`\psshadow*[`*settings*`]{`*text*`}`
Рисуется тень текста *text*

`\pstilt{`*degrees*`}{`*text*`}`
Текст наклоняется *text*

`\ThreeDput(`x_0,y_0,z_0`){`*object*`}`
Объект *object*, помещенный в точку с координатами (x_0),(y_0),(z_0), изображается в соответствии с текущей точкой наблюдения

Таблица 4.9. Графические параметры пакета PSTricks для работы с третьим измерением.

Параметр	*По умолчанию*	*Описание*
Tshadowsize=*size*	1	Длина тени
Tshadowcolor=*color*	lightgray	Цвет тени
Tshadowangle=*angle*	60	Угол падения тени
viewpoint=*x y z*	1 -1 1	Позиция наблюдателя, который смотрит на объект
normal=*x y z*	0 0 1	Вектор, ортогональный плоскости двумерного объекта, описывающий положение объекта в трехмерном пространстве
embedangle=*angle*	0	Поворот вокруг оси, проходящей через точку привязки объекта и направленный вдоль вектора, указывающего положение объекта

акет XY-pic

XY-pic является универсальным пакетом для рисования, основанным на TEX'е. Он хорошо взаимодействует с большинством форматов, включая LATEX, AMS-LATEX, AMS-TEX и plain TEX. Он применяется для набора сложных диаграмм разных предметных областях, включая теорию категорий, теорию автоматов, алгебру, геометрию, теорию нейросетей и теорию узлов. Общий синтаксис акета XY-pic позволяет использовать продуманную мнемоническую систему аписи, основанную на *логической* структуре диаграмм, путем комбинации различных элементарных *наглядных* компонентов. Кроме того можно написать макро, последовательно составляющие из этих элементов конструкции ысокого уровня, специально ориентированные на данное приложение.

Первая версия XY-pic была написана Кристофером Хёгсбро Роузом (Rose, 1992). За последние несколько лет он в сотрудничестве с Россом Муром (Rose, 1995; Rose and Moore, 1995) значительно расширил и улучшил этот пакет.

5.1 Введение в XY-pic

Система XY-pic построена на основе объектно-ориентированного графического языка, называемого *ядром*: это обозначает способ записи создания сложных «объектов» с помощью «методов», соответствующих осмысленным графическим операциям над этими объектами.

Ядро поддерживает следующие базовые графические понятия (см. азд. 5.3):

- *Позиции*, которые могут задаваться в различных форматах. В частности, определяемые пользователем координаты могут быть абсолютными или указанными относительно предыдущих позиций, объектов, краев объектов или точек на соединительных элементах.
- *Объекты*, которые могут иметь несколько форм, например округлую, эллиптическую, прямоугольную, и могут корректироваться несколькими способами, даже в зависимости от *направления* других объектов. В частности, объект может быть использован для *соединения* двух других объектов.

Дополнительные возможности ядра, называемые «опциями», бывают двух видов: *расширения* (см. разд. 5.4), которые делают доступными дополнительные объекты и методы (такие, как «создание кривых» и «создание рамок»), и *свойства* (см. разд. 5.5), которые вводят обозначения для конкретных предметных областей (такие, как «стрелки-указатели», «матрицы», «многоугольники», «решетки», «узлы»). В общем случае расширения добавляют новые графические компоненты, тогда как свойства — специфические для данной предметной области обозначения, облегчающие их логическое построение.

В этой главе приведены примеры использования пакета XY-pic в различных предметных областях. Благодаря такому подходу «обучения на примерах» эта глава дополняет книгу XY-pic *User's Guide* (Rose, 1995), в которой описано большинство функций пакета, и книгу XY-pic *Reference Manual* (Rose and Moore, 1995), в которой описан синтаксис всех команд пакета XY-pic вместе с их параметрами. Изучение наших примеров должно помочь вам быстро начать рисовать ваши собственные диаграммы. Мы также надеемся, что она покажет вам красоту, широкие возможности и гибкость пакета XY-pic.

5.2 Первый пример программы в рамках пакета XY-pic

Пакет XY-pic состоит из целого ряда модулей. Если вы не знаете, какой из них загрузить, то, возможно, лучшим выходом будет загрузить «большой набор» следующим образом[1]:

```
\usepackage[all]{xy}
```

С другой стороны, когда вы изучите пакет XY-pic достаточно хорошо, чтобы определять, какие именно функции вы хотите использовать, то сможете указывать только те расширения или модули, которые вам действительно необходимы, например:

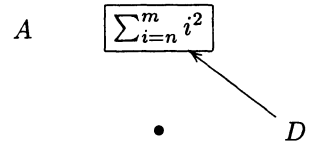
```
\usepackage[curve,arrow,cmactex]{xy}
```

В приведенном выше примере используются расширение `curve` и модуль `arrow`, настроенные на вывод команд `\special`, понимаемых написанной Томасом Киффе программой CMacTeX — разновидностью TEX'а, предназначенной для компьютеров Macintosh.

Для получения представления о философии, лежащей в основе пакета XY-pic, давайте сначала посмотрим, как «конструируется» XY-рисунок. Чтобы сделать пример относительно простым, рассмотрим диаграмму, похожую на матрицу. Как объясняется в разд. 5.5.2, основной способ создания диаграммы состоит в использовании команды `\xymatrix{`*spec*`}`, где *spec* — это описание *элементов матрицы*, которые, в общем случае, выравниваются по *строкам*

[1] Для форматов, отличных от LATEX'а, используйте команду `\input xy`, за которой следует `\xyoption{`*all*`}`. Опция `all` загружает расширения `curve`, `frame`, `tips`, `line`, `rotate` и `color`, а также модули `matrix`, `arrow` и `graph`. Любые другие модули или расширения должны загружаться отдельно.

и *столбцам*. Так же, как и в окружении **tabular**, элементы из одной строки отделяются символом **&**, а последовательные строки отделяются символами ****.



5-2-1

```
\usepackage [all]{xy}
\[
\xymatrix{
 A & *+[F]{\sum_{i=n}^m {i^2}} \\
   & {\bullet} & D \ar[ul]
}\]
```

Этот пример содержит две строки с тремя столбцами в каждой и иллюстрирует многое из того, как в Xy-pic интерпретируются команды:

- По умолчанию внутри окружения пакета Xy-pic элементы набираются в обычном математическом режиме и они центрированы.
- Во многих случаях вы не можете начинать ввод содержимого ячейки просто с имени макро — такие имена должны быть заключены в фигурные скобки или быть «защищенными».
- Как и внутри окружения **tabular**, пустые элементы в конце строк матрицы могут быть опущены, если к ним нет обращений из других мест.
- К элементам можно обращаться на основе их *относительного* («логического») расположения на диаграмме. Именно таким образом инструкция **\ar[ul]** создает стрелку-указатель, идущую из «текущей» позиции к ячейке матрицы, «расположенной на одну строку выше и один столбец левее».
- *Формат* и *форму* элемента можно настраивать с помощью указания «модификаторов элемента» (например, модификатор «**[F]**» указывает пакету Xy-pic заключить элемент в рамку).

5.3 Основные конструкции

Глубокое понимание того, как Xy-pic интерпретирует различные команды, позволит вам полнее использовать возможности его многочисленных функций. Кроме того оно поможет вам вникнуть во все тонкости различных расширений и модулей, которые будут описаны в последующих разделах.

Ядро Xy-рисунка находится внутри окружения **xy**[1]:

```
\begin{xy}...\end{xy}
```

Место, куда «роняют» Xy-pic'овский объект, называется его «позицией». В действительности, в большинстве случаев задаются лишь координаты или форма «текущей позиции».

[1] При использовании пакета Xy-pic с форматами, отличными от LaTeX'а, применяйте **\xy...\endxy**.

5.3.1 Определение положений

Простейшая форма задания позиции в XY-pic называется *абсолютной* и записывается в виде <*X*,*Y*>. Координаты *X* и *Y* представляют собой сдвиги точки *вправо* и *вверх* относительно начала отсчета рисунка, которое помещается в точку с координатами <0cm,0cm>. Для позиционирования текущей точки можно использовать простые арифметические операторы. Для отделения одной позиции от другой используется запятая:

UL *UR*
5,5
DL *DR*

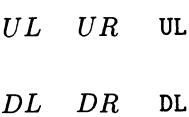
```
\usepackage {xy}
\[\begin{xy}
  0*{DL} ,+/r1cm/*{DR}
 ,<0cm,1cm>*{UL} ,<1cm,1cm>*{UR}
 ,(5,5)*{5,5}
\end{xy}\]
```

5-3-1

Приведенный выше пример иллюстрирует различные способы указания координат[1] В частности, 0 (ноль) служит сокращенной формой записи начала координат, а +/r1cm/ — смещения вправо на 1 см. Следующие две точки <0cm,1cm> и <1cm,1cm> являются явными *X*,*Y*-координатами. Наконец, срединная точка (5,5) использует *принятую по умолчанию систему координат* с единицей измерения, равной 1 мм, для обычных декартовых осей *x* и *y*, имеющую начало координат в точке (0,0). Об операторе «*» более подробно будет рассказано ниже.

В следующем примере единицы измерения системы координат определены явно путем установки их равными 5 мм по *X* и по *Y* с помощью оператора :. Это означает, что все последующие пары координат, в которых явно не указываются единицы измерения, измеряются в величинах, кратных 5 мм. Вы можете прибавлять или вычитать длины из данной позиции. Например, для создания правой части приведенной ниже диаграммы, начиная с третьей строки программы, мы сначала произвели сдвиг системы координат относительно начала координат 0 на четыре единицы измерения (т. е. 2 см) вправо и с помощью оператора «=» обозначили это новое положение "NO" (имея в виду «new origin» (новое начало координат), причем кавычки обозначают, что это — имя). Это имя затем используется для вычисления позиции, в которой мы хотим поместить текстовый объект UR и назвать его "SUR" (что значит «saved upper right» (сохраненный выше и правее)). После этого имя "SUR" используется еще дважды, причем каждый раз из него вычитаются координаты для получения позиций, в которых должны быть помещены тексты UL и DR.

[1] Обратите внимание, что разделительная запятая помещается *перед* следующим элементом, а не *после* предыдущего. Такое соглашение особенно удобно при создании рисунков, поскольку целые строки могут быть легко «закомментированы» и при этом сохранится правильность синтаксиса в целом.

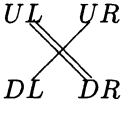


5-3-2

```
\usepackage {xy}
\[\begin{xy} 0;<5mm,0mm>:
 0*{DL} ,(2,0)*{DR}
 ,<0cm,1cm>*{UL} ,(2,2)*{UR}
 ,0+(4,0)="NO"*{\mathtt{DL}}
 ,"NO"+(2,2)="SUR"*{\mathtt{UR}}
 ,"SUR"-<1cm,0cm>*{\mathtt{UL}}
 ,"SUR"-(0,2)*{\mathtt{DR}}
\end{xy}\]
```

5.3.2 Создание соединений

Результаты работы *соединительного* оператора ** могут быть весьма сложными. В первом приближении он «соединяет» *текущую* и *предыдущую* позиции (соответственно *c* и *p*). В качестве простого примера соединим некоторые элементы предыдущей диаграммы.



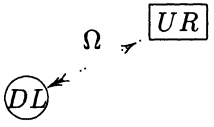
5-3-3

```
\usepackage {xy}
\[\begin{xy}
 0*{DL};<1cm,1cm>*{UR}**@{-}
 ,<0cm,1cm>*{UL};<1cm,0cm>*{DR}**@{=}
\end{xy}\]
```

Здесь операция соединения **@{-} строит соединение с помощью специального объекта ядра @{-}, который строит соединительный элемент в виде сплошной линии. В документации по пакету XY-pic перечислены типы изначально определенных соединений. Новые соединения можно создавать с помощью команды \newdir. У большинства из них есть мнемонические имена: @{.} соединяет точечной линией, @{--} — штриховой линией, @{=} — двойной линией и т. д. Обратите внимание на использование оператора точка с запятой (;) который *меняет местами* позиции *p* и *c*. В третьей строке нашего примера результат его работы заключается в перемещении позиции из *c* (т. е. 0) в *p*, после чего *c* немедленно присваивается новое значение, отвечающее положению следующего объекта (помните, что соединения проводятся оператором ** из *p* в *c*). Пример также демонстрирует, что оператор размещения * по умолчанию не окружает объекты никакими полями. Такое пространство создается благодаря вставке одного или нескольких модификаторов + между оператором * и открывающей объект фигурной скобкой. Подробнее об этом мы поговорим ниже.

Вы можете комбинировать несколько операторов размещения, а оператор вопросительный знак (?) дает вам возможность указать местоположение, в

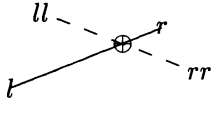
котором нечто должно быть нарисовано «вдоль» только что проведенного соединения без явного задания координат:




```
\usepackage [frame]{xy}
\[\begin{xy}
 0*+[o][F]{DL};<2cm,1cm>*+[F]{UR} **@{.}
 ?<*@{<<} ?(0.5)*!/_3mm/{\Omega} ?>>>*@{>}
\end{xy}\]
```
5-3-4

Оба объекта увеличиваются с помощью модификатора +, а модификатор [o] делает *форму* первого объекта круглой, а не принятой по умолчанию прямоугольной (чтобы подчеркнуть этот факт, мы добавили модификатор [F], который определяется в расширении, строящем рамки). Во второй строке оператор ? позволяет позиционировать объекты в требуемом месте вдоль последнего «соединения» (в данном случае вдоль пунктирной линии между окружностью и рамкой). Сначала двойной кончик стрелки-указателя слева помещается в начальной точке (?<), т. е. в точке, положение которой отвечает позиции p после проведения линии. Далее рядом со срединной точкой (?(0.5)) и в 3мм над линией (с помощью оператора смещения, который будет описан ниже) помещается символ Ω. На расстоянии в несколько единиц длины от конечной точки линии, т. е. в позиции c после проведения соединения, помещается одинарный кончик стрелки-указателя (позиционированный с помощью ?>>>). Действие операторов > и < может «накапливаться», в некоторой степени аналогично действию операторов +, ++ и т. д., изменяющих поле вокруг объекта.

Наконец, с помощью оператора ?! может быть вычислена точка пересечения двух линий:




```
\usepackage {xy}
\[\begin{xy}
  0*=+{l}="l" ;<2cm,8mm>*=+{r}="r" **@{-}
 ,<4mm,10mm>*+{ll} ;<25mm,2mm>*+{rr} **@{--}
 ?!{"r";"l"}*{\oplus}
\end{xy}\]
```
5-3-5

Сначала рисуются две линии (l, r) и (ll, rr) и их конечные точки сохраняются, соответственно, под именами "l" и "r". С помощью записи ?!{<XY1>;<XY2>} (где <XY1>, <XY2> — это две позиции) находится точка пересечения с последним соединением и в этом месте помещается LaTeX'овский символ ⊕.

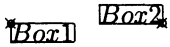
5.3.3 Размещение объектов

В большинстве примеров в предыдущем разделе мы уже использовали *объекты*. Точнее, объекты являются аргументами операторов *размещения* * и *соединения* **. Обычно они оказываются теми элементами, которые реально появляются на рисунке. Как мы видели выше, объекты всегда содержат пару фигурных скобок {...}, указывающих, что именно предстоит нарисовать. Предшествующая открывающей скобке часть, называемая *модификатором*,

позволяет выполнять тонкую настройку того, как именно должен быть представлен объект.

Модификатор *сдвига* «!» позволяет вам смещать точку привязки объекта из начальной центральной позиции в какую-нибудь другую, расположенную внутри обрамляющего бокса объекта:

5-3-6




```
\usepackage [frame]{xy}
\[\begin{xy}
  (0,0)*@{o}*!UL{Box1}*\frm{-}*@{x}
 ,(20,0)*@{o}*!RD{Box2}*\frm{-}*@{x}
\end{xy}\]
```

Верхний левый угол первого бокса (обозначаемый `!UL`) помещается в текущей позиции `(0,0)`, а нижний правый угол второго бокса (`!RD`) помещается в текущей позиции `(20,0)`. В каждом случае позиция точки *с* остается неизменной, поскольку символы (×, набранный с помощью команды `@{x}`) и окружность (○, набранная с помощью команды `@{o}`) на рисунке накладываются друг на друга. Этот пример также иллюстрирует применение модификатора `@`, необходимого для получения из библиотеки ядра «имеющего направление» объекта, мнемоническое обозначение которого указывается в фигурных скобках (мы уже несколько раз пользовались этим, не приводя объяснений).

Сдвиг объекта можно сравнить с оператором *смены* позиции, который тоже использует символ «!», но помещается после описания объекта или позиции:

5-3-7

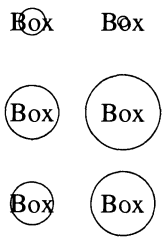

```
\usepackage [frame,arrow,curve]{xy}
\[\begin{xy}
  (0,0)*@{*}*[F]{Box1}!UL="a",*@{x}
 ,(20,0)*@{*}*[F]{Box2}!RD*@{x}
 ,\ar@/^20pt/"a", \POS"a"!DR(.8)*@{+}
\end{xy}\]
```

В данном случае текущая точка *с* перемещается в положение, отмеченное на боксе (в чем легко убедиться, если сравнить позиции точки *с до* (•) и *после* (×) применения оператора `!`). Однако занимаемое объектом «пространство» остается тем же самым пространством, занимаемым самим боксом, что видно из того, где помещается стрелка-указатель на кривой. Различие между перемещением и сменой позиции подчеркивает существенную разницу между операторами позиционирования и модификаторами: только первые могут изменять текущую позицию, тогда как вторые могут использоваться для изменения занимаемых площадей, корректировки и т. п. действий с рассматриваемым объектом по отношению к точке привязки, к которой он «прикреплен». Модификаторы не оказывают никакого воздействия после того, как объект в нее помещен.

Для указания вектора перемещения или сдвига существуют еще несколько команд. Мы уже видели, как для текущего объекта можно изменить позицию с помощью векторов `R` right (вправо), `L` left (влево), `U` up (вверх), `D` down (вниз) и их комбинаций. Эти векторы обозначают смещение от точки привязки до

границ текущего (прямоугольного) объекта. C — это смещение до центра. Эти смещения можно корректировать с помощью «коэффициентов», например, инструкция !DR(.8) в предыдущем примере указывает, куда поместить знак + (@{+}). Другой полезной функцией является возможность задавать определенное расстояние в *текущем направлении*, что сделано в самом последнем соединении с помощью **. Запись /3mm/ означает, что вектор имеет длину 3 мм и ориентирован вдоль текущего направления. Аналогично, запись /_3mm/ означает, что вектор имеет такую же длину, но ориентирован под углом 90° по часовой стрелке к текущему направлению (именно так и было задано положение символа Ω в одном из предыдущих примеров), тогда как запись /^3mm/ означает, что вектор ориентирован против часовой стрелки к текущему направлению.

Среди модификаторов объектов существуют точки, которые изменяют размер и форму обрамляющего бокса объекта (называемого *кромкой* на жаргоне пакета XY-pic). Изначально этот размер равен размеру самого объекта, но может быть изменен с помощью модификаторов *увеличения* «+» и *уменьшения* «-». Они добавляют или удаляют фиксированный размер, называемый *полем вокруг объекта*, чтобы создать дополнительные или уменьшить существующие поля вокруг данного объекта. Принятое по умолчанию значение поля вокруг объекта составляет 3 pt, но легко может быть изменено. Модификатор *установки размера* «=⟨*wid*,*ht*⟩» позволяет определять ширину и высоту любых величин. Без указания размера модификатор «=» используется для преобразования формы объекта в квадратную, т. е. задания ширины и высоты равными наименьшей из их текущих величин. Аналогичным образом, модификатор «+=» устанавливает обе величины равными наибольшей из них. Модификаторы всегда обрабатываются последовательно слева направо.




```
\usepackage [frame]{xy}
\[\begin{xy}
 ,(0,0)*+=[o][F]\txt{Box}
 ,(12,0)*+=[o]=<9mm>[F]\txt{Box}
 ,(0,12)*+=[o]+[F]\txt{Box}
 ,(12,12)*+=[o]++[F]\txt{Box}
 ,(0,24)*+=[o]-[F]\txt{Box}
 ,(12,24)*+=[o]--[F]\txt{Box}
\end{xy}\]
```

5-3-8

В приведенном выше примере в нижней строке слева демонстрируется проведенная вокруг объекта окружность, размер которой выбран по умолчанию, а справа — окружность радиуса 9 мм (точнее говоря, высота и ширина объекта установлены равными 9 мм, поскольку запись <9mm> является сокращенной формой для <9mm,9mm>). В средней строке радиус окружности увеличивается на размер поля вокруг объекта, а потом на удвоенный размер поля вокруг объекта. Наконец, в верхней строке радиус окружности уменьшается на такие же величины.

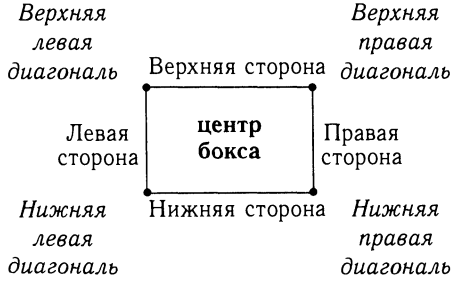
5.3.4 Вставка текста в рисунки

Вставлять текст в созданные пакетом XY-pic рисунки позволяет команда `\txt`. Она набирает текст в виде выровненных по центру абзацев, т. е. контроль за разбиением на строки осуществляется с помощью команды `\\`. Синтаксис этой команды имеет вид:

`\txt` ⟨*wid*⟩*sty*{*text strings*}

Как аргумент ⟨*wid*⟩, задающий ширину, так и аргумент *sty*, определяющий применяемый для набора стиль, могут отсутствовать. В примере ниже показаны существующие возможности. Обратите внимание на использование *противоположно направленных* векторов для получения одной и той же позиции с помощью перемещения и сдвига[1].

```
\usepackage [curve,frame]{xy}
\usepackage [cp866]{inputenc}
\usepackage [T2A]{fontenc}
\usepackage [russian]{babel}
\usepackage {pscyr}
\[
\newcommand{\smbf}{\small\bfseries}  \newcommand{\smit}{\small\itshape}
\begin{xy}
 0*=(22,14)!UR!(-10,-8)[F]\txt\smbf{центр\\бокса}="box";
  "box"+L*+!R\txt\small{Левая\\сторона}
 ,"box"+R*+!L\txt\small{Правая\\сторона}
 ,"box"+D*+!U\txt\small{Нижняя сторона}
 ,"box"+U*+!D\txt\small{Верхняя сторона}
 ,"box"+LD*@{*}*+!RU\txt<2cm>\smit{Нижняя левая диагональ}
 ,"box"+RD*@{*}*+!LU\txt<2cm>\smit{Нижняя правая диагональ}
 ,"box"+LU*@{*}*+!RD\txt<2cm>\smit{Верхняя левая диагональ}
 ,"box"+RU*@{*}*+!LD\txt<2cm>\smit{Верхняя правая диагональ}
\end{xy}\]
```



5-3-9

[1] Этот пример также иллюстрирует, что определения новых командных последовательностей `\newcommand` должны помещаться перед началом окружения `xy`, но внутри выделенного математического режима, чтобы область видимости этих определений не распространялась на последующие диаграммы.

5.4 Расширения

Мы уже пользовались расширением для построения рамок, так как именно в нем определен модификатор `[F]`. После активации других параметров различных расширений становятся доступными следующие сложные графические функции:

- Расширение `curve` позволяет рисовать *кривые* и *сплайны* на основе квадратичных и кубических кривых Безье и B-сплайнов (см. разд. 5.4.1).
- Расширение `frame` обеспечивает удобный способ построения *рамок*, *скобок* и *закрашенных областей* (см. разд. 5.4.2).
- Расширение `tips` позволяет выбирать стили оформления *кончика стрелки-указателя* (дополнительно к выбираемому по умолчанию в пакете XY-pic «техническому» стилю становятся доступными стили Computer Modern или Euler), тогда как *стили оформления прямых* контролируются благодаря расширению `line`.
- С помощью расширения `rotate` становится возможным выполнять как *повороты*, так и *изменение масштаба*. Расширения `color` и `tile` предоставляют возможность работы с *цветами* и *шаблонами*, а также достичь эффекта *паркета*. С помощью расширения `import` импортируется графический материал.

Некоторые из этих функций доступны в полном объеме только в том случае, если их поддерживает ваш dvi-драйвер. Однако следует подчеркнуть, что по умолчанию пакет XY-pic использует лишь стандартный TEX и METAFONT. Поэтому результатом его работы являются стандартные dvi-файлы, содержащие обращения к шрифтам пакета XY-pic. Выводной файл, предназначенный для какого-нибудь определенного драйвера (например, dvips), можно получить, загрузив соответствующий необязательный параметр *вывода*, «подстраивающий» направляемые в dvi-файл инструкции под указанный драйвер (используя команды `\special`). Этот способ *не* расширяет язык XY-pic. Когда требуемого параметра вывода не существует, XY-pic старается добиться как можно лучшего соответствия выдвинутым требованиям, используя в числе прочих средств специальные XY-pic'овские шрифты. В частности, размер рисунка при этом остается неизменным, поэтому изменение параметра вывода никак не влияет на нумерацию страниц и т. д.

В качестве примера рассмотрим расширение `ps`, позволяющее включать инструкции на языке PostScript в XY-рисунки вне зависимости от того, понимает ли их драйвер или нет (конечно, встроенные PostScript'овские команды будут работать только в том случае, если драйвер их действительно понимает). Если этот же файл предполагается обработать LATEX'ом на другом компьютере, то исходный файл не нужно изменять (за исключением добавления описания используемого параметра вывода). В случае если целевой драйвер поддерживает PostScript, шрифты пакета XY-pic не используются, так как все

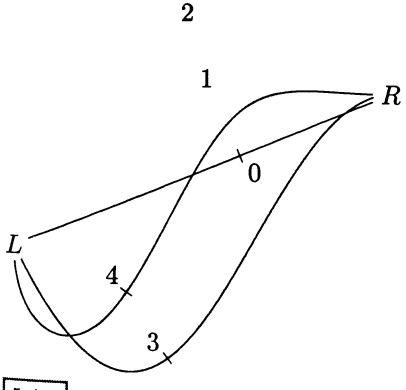
стрелки, их кончики и т. д. создаются средствами самого PostScript'а[1]. Это позволяет улучшить качество вывода на печать, особенно в случае пунктирных или штриховых линий и кривых. Кроме того, уменьшается время, затрачиваемое на обработку LaTeX'ом. Больше в этой книге мы не будем обсуждать параметры вывода. Дальнейшие подробности, включая список поддерживаемых на данный момент драйверов, можно найти в документации по XY-pic'у.

5.4.1 Кривые и сплайны

Расширение `curve` предоставляет возможность рисовать сплайны. Оно также позволяет проводить искривленные соединительные линии, вдоль которых могут располагаться различные объекты. Если объекты обладают направлением, они располагаются вдоль касательной линии. Базовая форма команд рисования имеет вид

`\crv` *setup*`{`*control-points*`}`

где аргумент *control-points* является списком позиций, разделенных знаками `&`. Этот объект приобретает смысл только при наличии соединительного оператора `**`. Как обычно, предыдущая точка p и текущая точка c задают концевые точки соединения.



5-4-1

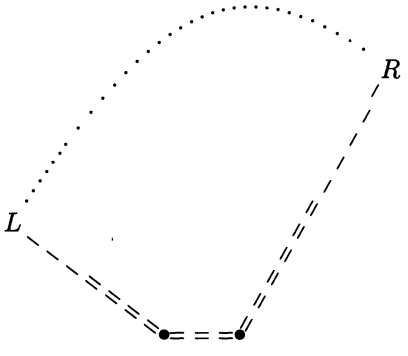
```
\usepackage [curve,frame]{xy}
\[\begin{xy}
 0*[o]+{L};(50,20)*[o]+{R}="R"
   **\crv{} ?(.6)*@{+}*^+!UL{0}
  ,"R" **\crv{(30,30)} ?*@{+}*^+!DR{1}
  ,"R" **\crv{(20,30)&(30,40)}
   ?*@{+}*^+!DR{2}
  ,"R" **\crv{(10,-20)&(25,-20)&(40,20)}
   ?(.4)*@{+}*^+!DR{3}
  ,"R"
   **\crv{(0,-10)&(12,-20)
          &(28,25)&(40,20)}
   ?(.4)*@{+}*^+!DR{4}
 \end{xy}\]
```

В этом примере показаны пять кривых, помеченных цифрами от 0 до 4, которые обозначают число использованных дополнительных контрольных точек. Это число определяет используемый тип кривой Безье, соединяющей начальную (L) и конечную (R) точки. В случае одной или двух контрольных точек строится квадратичная или кубическая кривая Безье, причем направления касательных в точках L и R совпадают с прямыми, соединяющими их

[1] Этот пакет также содержит различные XY-pic-специфические шрифты в формате Type 1: они используются только тогда, когда PostScript-драйвер обрабатывает `dvi`-файл, созданный *без* задания PostScript'овского параметра вывода!

с соседними контрольными точками. Когда указываются три или более контрольные точки, строится кубический B-сплайн. Обратите внимание на применение оператора ? для указания мест расположения чего-либо на соединении. Помимо определения необходимого места он также устанавливает «текущее направление» направлением, под которым проходит касательная к кривой в *данной точке*. Маленькие крестики, заданные с помощью `*@{+}`, показывают, что этот имеющий направление объект выровнен соответствующим образом.

Используя вместе с объектами `\crv` необязательные параметры, очень легко сделать контрольные точки видимыми.




```
\usepackage [curve]{xy}
\[\begin{xy}
  0*+{L};(50,20)*+{R}
  **\crv{~*=<2mm>{.}(20,30)&(30,35)}
  **\crv~Lc{~**@{==}~*{\bullet}%
       (20,-15)&(30,-15)}
\end{xy}\]
```

5-4-2

У первой команды `\crv` нет необязательной части, но зато используется спецификатор объектов `~*`, определяющий, какие объекты должны быть нарисованы вдоль траектории. Рисующая вторую кривую команда имеет необязательный параметр `Lc`, указывающий, что кривая должна быть нарисована вместе с контрольными точками и соединяющими их отрезками. Более того, координатам предшествуют спецификаторы соединительного элемента (`~**`) и оператора размещения (`~*`). Они определяют то, как изображаются контрольные точки и в каком стиле нарисованы соединения.

Если вы не располагаете драйвером вывода со встроенной поддержкой замысловатых кривых, их рисование может потребовать значительной доли ресурсов памяти. В такой ситуации разумно воспользоваться командой `\SloppyCurves` или изменить допуск при наборе кривых с помощью команды `\splinetolerance{`*tol*`}`, единственным параметром которой является длина *tol*. Кривая строится из последовательности близко расположенных точек. Значение параметра *tol* равно минимальному расстоянию между точками, для которого не вычисляется положение точки между ними (это *не* одно и то же, что промежуток между самими точками). Отсюда становится легче понять причину неравномерного расположения объектов на кривых, особенно пунктирных или штриховых.

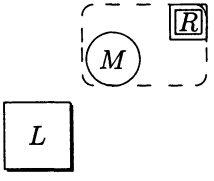
Минимально возможное значение допуска, используемое для рисования красивых «сплошных» кривых, равно 0.2 pt. Именно это значение в действительности используется, когда этот параметр установлен равным 0 pt. Задание `\SloppyCurves` присваивает параметру *tol* значение 0.8 pt.

5.4.2 Рамки и скобки

Расширение `frame` вводит в употребление команды вида

`\frm`⟨*dim*⟩`{`*spec*`}`

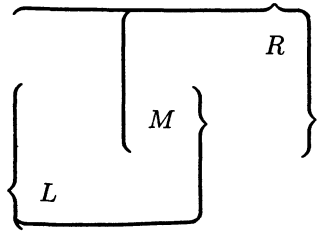
где *spec* — это описание рамки, а ⟨*dim*⟩ — необязательный размер. Когда рамка «размещается» (с помощью оператора `*`), то она строится вокруг объекта в текущей точке *c*, а когда рамка задается в качестве «соединяющего» соединительного элемента (с помощью оператора `**`), то рамкой обводится прямоугольник, заданный совместно предыдущим и текущим объектами. Полный перечень всех возможных рамок приведен в документации к пакету XY-pic.



5-4-3

```
\usepackage [frame]{xy}
\[\begin{xy}
  0*++++{L}*\frm<2pt>{-,}
 ,(10,10)*++{M}*\frm{o};
  (20,15)*+{R}*\frm{=}**\frm{o-}
\end{xy}\]
```

На приведенном выше рисунке после своего размещения буква *L* обводится прямоугольным боксом с тенью, причем размер бокса увеличен на величину, троекратно превосходящую поле вокруг объекта. В точке с координатами `(10,10)` размещается буква *M*, вокруг которой проводится круговая рамка, увеличенная на удвоенный размер поля вокруг объекта. Затем текущая позиция превращается в *предыдущую* позицию *p* (с помощью оператора `;`), а текущая позиция *c* переносится в точку с координатами `(20,15)`. Здесь размещается буква *R*, которая окружается двойной прямоугольной рамкой, расширенной на величину поля вокруг объекта. Наконец, с помощью соединительного оператора `**` строится прямоугольная штриховая рамка с закругленными краями, охватывающая прямоугольник, который задают расположенные в точках *c* и *p* объекты.



5-4-4

```
\usepackage [frame]{xy}
\[\begin{xy}
  (0,0)*++++{L}="l";
  p+(15,10)*++++{M}**\frm{_)}% 1
                  **\frm{\{}  % 2
                 ;**\frm{\}}  % 3
 ,p+(15,10)*++++{R}**\frm{(} % 4
                  **\frm{\}}  % 5
             ;"l"**\frm{^\}} % 6
\end{xy}\]
```

При конструировании скобок с помощью расширения `frame` необходимо внимательно следить за предыдущей позицией p и текущей позицией c. В приведенном выше примере происходит размещение буквы L, после чего данной позиции присваивается имя `"1"` и она сохраняется в качестве предыдущей позиции p (оператор `;`). Затем позиция p используется для вычисления координат нового объекта M. На следующем шаге объекты в позициях c и p соединяются нижней круглой скобкой (в строке «1») и открывающей фигурной скобкой (в строке «2»). После этого позиции p и c меняются местами, так что M становится p, а $L - c$ и рисуется закрывающая фигурная скобка. Обратите особое внимание на то, как выровнены меньшие части скобок из строчек «2» и «3». Далее позицией c становится точка R, после чего строится открывающая круглая скобка (в строке «4»), а затем закрывающая фигурная скобка (в строке «5»), охватывающие точки M и R. Вновь применив оператор `;`, объект R делается «предыдущей» позицией p и восстанавливается сохраненная ранее позиция `"1"` точки L, которая становится «текущей» позицией c. Это позволяет на заключительном этапе нарисовать верхнюю фигурную скобку (в строке «6»).

Ради удобства существует возможность вместо того, чтобы размещать рамку идущей после объекта отдельной командой `*\frm`⟨*dim*⟩`{`*spec*`}`, получить тот же результат с помощью модификатора самого объекта `[F` *spec*`:`⟨*dim*⟩`]` или даже `[F` *spec*`]`, если не указывается никакого размера ⟨*dim*⟩. Самый простой вариант `[F]` соответствует команде `*\frm{-}`, которая требуется чаще всего.

5.5 Дополнительные особенности

Пакет XY-pic содержит достаточно полный набор интересных дополнений, расширяющих возможности XY-pic для конкретных предметных областей. Эти так называемые «дополнительные особенности» должны загружаться по мере необходимости. Последняя версия пакета XY-pic позволяет работать в следующих областях:

- *Стрелки-указатели.* Рисуются простые и составные стрелки с настраиваемым видом меток (разд. 5.5.1).
- *Матрицы.* Создаются двумерные матрицеподобные изображения, в которых обращение к объекту может быть основано на номере строки и столбца, где он находится, или на «шагах», пройденных вдоль сетки от другого объекта (разд. 5.5.2).
- *Графы.* Рисуются ориентированные графы, блок-схемы, деревья и т. д. (разд. 5.5.3).
- *Двойные ячейки.* Набираются «двухэлементные» диаграммы из теории категорий, содержащие пары (помеченных) искривленных стрелок-указателей (разд. 5.5.4).

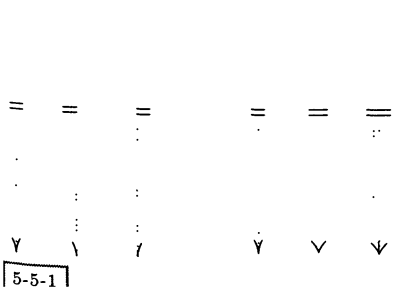
- *Многоугольники.* Определяются позиции вершин правильных многоугольников (разд. 5.5.5).
- *Окружности, эллипсы и дуги.* Строятся (полностью или частично) окружности и эллипсы, основные и вспомогательные оси которых выровнены вдоль произвольного направления (разд. 5.5.6).
- *Решетки и сетки.* Рисуются объекты, расположенные на регулярной двумерной решетке (разд. 5.5.7).
- *Узлы и сцепления.* Рисуются и помечаются структуры, имеющие вид узлов и сцеплений (разд. 5.5.8).

5.5.1 Стрелки-указатели

При разработке пакета XY-pic построение «красивых» стрелок-указателей считалось первоочередной задачей, по словам его автора (Rose, 1992): «Нашей первой задачей [по отношению к XY-pic] было создание таких стрелок-указателей, которые выглядели бы хорошо, даже будучи очень длинными ...»

Дополнительный пакет `arrow` автоматически загружается при задании необязательного параметра `all`. На самом деле многие другие расширения также используют этот пакет, поэтому он автоматически загружается и ими тоже. Стрелки-указатели реализованы в виде расширений соединительных элементов, что позволяет явно обращаться к *хвосту*, *телу* и *голове* стрелки. Поскольку указатели встречаются во многих местах, для их набора создан простой и удобный синтаксис, начинающийся командой `\ar`. Наше изложение здесь ограничится несколькими простыми примерами. Разнообразие возможностей этого пакета станет очевидным позже, при описании пакетов `matrix` и `knot`.

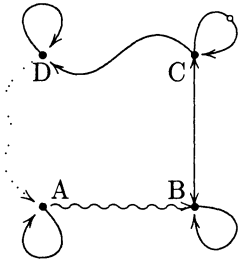
Вообще говоря, стиль изображения стрелки настраивается с помощью символа `@`. Заключенная в фигурные скобки часть описывает, какие хвост, тело и голова стрелки должны быть использованы. Ему предшествуют символы `^`, `_`, `0`, `1`, `2` или `3`, обозначающие, соответственно, верхний, нижний, невидимый, одинарный, двойной и тройной варианты, показанные в следующем примере (обратите внимание на отсутствие в центре изображения для `\ar@0`):



5-5-1

```
\usepackage [arrow]{xy}
\[\begin{xy}
  (0,-20)="a", (0,0)="b"
  \ar@{<.||}  @<24mm>  "a";"b"
  \ar@^{<.||} @<16mm>  "a";"b"
  \ar@_{<.||} @<8mm>   "a";"b"
  \ar@0{<.||}          "a";"b"
  \ar@1{<.||} @<-8mm>  "a";"b"
  \ar@2{<.||} @<-16mm> "a";"b"
  \ar@3{<.||} @<-24mm> "a";"b"
  \end{xy}\]
```

В следующем примере, где присутствуют разные уже описанные выше команды, показаны несколько полезных приемов (для их работы обязательно должны быть загружены расширения curve, frame и пакет arrow).



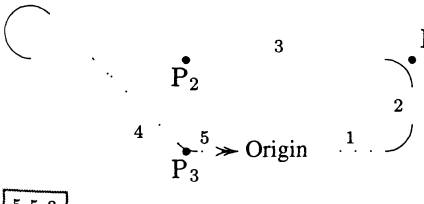

```
\usepackage [curve,arrow,frame]{xy}
\[\begin{xy} <1cm,0cm>:
             (0,0)*+@{*}="a"*+!DL{\mathrm{A}},"a"
 \ar@(dr,dl)
 \ar@{~>}    (2,0)*+@{*}="b"*+!DR{\mathrm{B}},"b"
 \ar@(r,d)   "b";"b"
 \ar@{<->}   "b";(2,2)*+@{*}="c"*+!UR{\mathrm{C}},"c"
 \ar@(u,r)   "c";"c" |*=<2pt>[o][F]{}
 \ar@(ul,dr) "c"; (0,2)*+@{*}="d"*+!U{\mathrm{D}},"d"
 \ar@(ul,ur) "d";"d"
 \ar@{.>}@`{(-0.8,1.5),(0.,1.0),(-0.8,0.5)} "d";
\end{xy}\]
```

5-5-2

После задания единицы измерения координат равной 1 см из точки A (получившей имя "a" с помощью оператора =), набираются несколько стрелок-указателей. Аналогично соединительным элементам стрелки проводятся в направлении из точки p в точку c, описание которой следует сразу за описанием стрелки. По умолчанию местом расположения обеих точек служит позиция точки c, которая является текущей перед началом построения стрелки: это используется в первой команде \ar для рисования петли, т. е. стрелки, начало и конец которой выходят из вершины A. Направления, в которых стрелка выходит из вершины и входит в нее, указываются с помощью @(<dir>,<dir>), где <dir> представляет собой *направление*, обозначение которого содержит одну из букв d или u и одну из букв l или r (другие варианты описаны в документации к пакету Xy-pic). Затем из точки A к черному кругу с меткой B, расположенному в точке (2,0), получившему имя "b", проводится волнистая стрелка. Обратите внимание, как с помощью оператора «запятая» предотвращается набор относящейся к вершине B метки, чтобы финальным местонахождением точки c оказалось имя "b", а не сама метка. Затем проводится вторая стрелка в виде петли, на этот раз идущая от явно указанного имени "b" к самому себе, следом проводится двусторонняя стрелка к вершине C, рядом с которой помещается еще одна петля, «разорванная» в середине окружностью радиуса 2 pt (набранной командной последовательностью *=<2pt>[o][F]{}). Следующая стрелка, соединяющая C и вершину D, расположенную в точке с координатами (0,2), является «искривленной стрелкой-указателем» с заданными направлениями касания в концевых точках. На последнем шаге рисуется еще одна искривленная стрелка, контрольные точки которой на этот раз явно заданы с помощью @`{cp, cp,...}, где каждая пара букв cp обозначает позицию контрольной точки.

Синтаксический анализ позиции, определяющей конечную точку стрелки-указателя (а также начальную точку, если применяется оператор ;), может продолжаться столь долго, насколько это необходимо. Как было показано выше, текущая позиция *c* является единственной целевой точкой, хотя вдоль траектории могут размещаться другие объекты. Аналогично, текущая позиция *p*, которой по умолчанию считается точка, предшествующая стрелке-указателю, является единственной исходной точкой. Область видимости этих значений заканчивается тогда, когда встречается команда `\relax`, или `\ar`, или `\endxy`, или `\end`, или когда встречается метка или символ разрыва (один из `^`, `_` или `|`), если только они не представляют собой аргумент другого оператора (что имело место в приведенном примере в случае команды `\mathrm`). Однако материал, следующий сразу после завершения команды `\ar`, *не* интерпретируется как описание позиции. Для ее описания следует использовать специальный макро `\POS`, что будет продемонстрировано в дальнейших примерах.

Наконец, стрелка-указатель, как любая траектория, может быть составлена из нескольких частей, разделенных ’ или ‘ в зависимости от того, должны ли они быть соединены друг с другом непосредственно или с помощью дуг окружностей. Каждая часть может содержать свой собственный набор меток и разрывов, расположенных вдоль прямой части:




```
\usepackage [arrow]{xy}
\[\begin{xy} <6mm,0mm>:
  (0,0)*+\txt\small{Origin}="O"
  \ar@{.>>}
    'r  (3,2) *\dir{*}*+!DL{\mathrm{P_1}}
                                    ^-1
    '    (-2,2)*\dir{*}*+!U{\mathrm{P_2}}
                                    ^2
    '_dr(-2,0)*\dir{*}*+!U{\mathrm{P_3}}
                                    _(.3)3
    'dr^r "O"                       _(.8)4
          "O"                       ^-(.2)5
\end{xy}\]
```

5-5-3

В этом примере движение начинается из позиции, помеченной «Origin» (идентификатор "O"), продолжается вправо, затем выполняется поворот к точке «P_1». Как только на обработку пакету XY-pic поступает точка «P_2», становится понятно, что далее необходимо продвигаться вверх и влево (обратите внимание на построенные четверти окружностей). Расположение двойной стрелки-указателя и меток 1–5 демонстрирует построение различных частей траектории. Чтобы сделать задачу более интересной, было задано, что для возвращения к началу, расположенному в точке "O", надо двигаться вниз и вправо, предварительно повернув по часовой стрелке (охватывающая три четверти окружности дуга, расположенная в верхнем левом углу рисунка), а затем пройдя через точку «P_3» с северо-запада на восток (вправо).

Метки в этом примере размещались с помощью позиционирующих «коэффициентов», указывающих долю общей длины траектории, которую надо пройти, чтобы достичь места расположения метки. В частности, метки «3» и «4» были размещены на расстоянии, составляющем 30% и 80% длины соответствующих прямых частей траектории. В случае метки «2», когда явно никакое значение указано не было, коэффициент был равным принятой по умолчанию величине 0.5. Знак минус в случаях меток «1» и «5» является удобным способом поручить пакету XY-pic вычислить позицию метки в виде доли «видимой части» отрезка прямой, т. е. за исключением отрезков в начале и конце траектории. Таким образом, метка «5» помещена на расстоянии, составляющем 20% видимой части последнего отрезка прямой, а метка «1» расположена посередине видимой части — (значение по умолчанию вновь равно 0.5). Использование знака - особенно удобно, когда либо начальная, либо конечная позиция занимает значительное пространство (как в приведенном выше примере).

5.5.2 Диаграммы, напоминающие матрицы

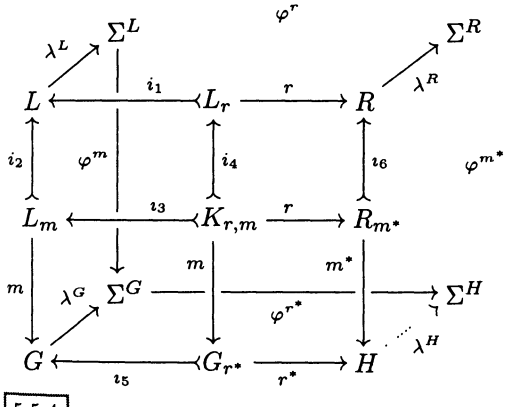
Дополнительный пакет `matrix` является прекрасным инструментом для набора диаграмм с регулярной структурой, напоминающих матрицы (на самом деле в первой версии пакета XY-pic были реализованы только возможности свойств `arrow` и `matrix`). Формат этой команды имеет вид

`\xymatrix` *setup*`{`*entries*`}`

Часть *setup* может содержать параметры выбора, параметры, задающие внешний вид, художественное оформление и т. п. параметры, которые могут быть отнесены к каждому элементу. Аргумент *entries* служит описанием следующего текста или объектов, расположенных в *ячейках* матрицы. Эти ячейки группируются в *столбцы*, разделенные символом `&`, и в *строки*, разделенные `\\`. Каждая ячейка может иметь произвольное XY-pic'овское художественное оформление, причем текущее значение величины *c* считается равным рассматриваемой ячейке. В частности, это означает, что данная ячейка может служить исходной точкой для команд `\ar`, указанных в данной ячейке. Сама матрица также является XY-pic'овским объектом, точка привязки которого расположена в верхней левой ячейке. Если `*` должен быть первым символом в ячейке, его следует вводить, поместив в фигурные скобки `{*}`, поскольку символ «звездочка» имеет специальный смысл. Большинство этих соображений иллюстрируется простым примером из разд. 5.2.

Поскольку бóльшая часть руководства XY-pic *User's Guide* (Rose, 1995) посвящена описанию пакета `matrix`, здесь мы рассмотрим только примеры, демонстрирующие некоторые интересные моменты. Первым из них является коммутативная диаграмма, использованная в обзорной статье (Valiente Feruglio, 1994), где сравнивались десять различных TEX'овских пакетов для набора ком-

мутативных диаграмм[1]. В данном примере нам потребуется также загрузить расширение tips и пакет arrow.



5-5-4

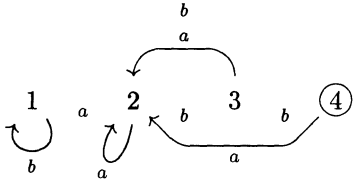
```
\usepackage [arrow,matrix,tips]{xy}
\[\UseTips
\newdir{ >}{!/-5pt/\dir{>}}
\xymatrix @=1pc @*[r] {
  & \Sigma^L \ar[rrrr]^{\varphi^r}
     \ar'[d]'[ddd]_{\varphi^m}
        [dddd]
  & & & & \Sigma^R
     \ar[dddd]^{\varphi^{m^*}}
\\ L   \ar[ur]^{\lambda^L}
  & & L_r \ar@{ >->}[ll]_(.32){i_1}
        \ar[rr]^r
  & & R \ar[ur]_{\lambda^R}
\\ \\
   L_m  \ar@{ >->}[uu]^{i_2}
        \ar[dd]_m
  & & K_{r,m}
        \ar@{ >->}[ll]_(.3){i_3}
        \ar@{ >->}[uu]_{i_4}
        \ar[dd]_<<<<m \ar[rr]^r
  & & R_{m^*}
        \ar@{ >->}[uu]_{i_6}
        \ar[dd]_<<<<{m^*}
\\  &  \Sigma^G
  \ar'[r]'[rrr]_{\varphi^{r^*}}
     [rrrr]
  & & & & \Sigma^H
\\ G \ar[ur]^(.7){\lambda^G}
  & & G_{r^*} \ar@{ >->}[ll]^{i_5}
        \ar[rr]_{r^*}
  & & H \ar@{.>}[ur]_{\lambda^H}
}\]
```

Сначала с помощью команды \newdir создается модификация стрелки-указателя @{>} — ее наконечник немного смещается с тем, чтобы хвост стрелки выглядел лучше. Все остальное — это просто матрица, сконструированная так, чтобы содержать элементы во всех «интересных» местах. Обратите внимание на использование правой кавычки ' для указания, что стрелка должна пройти сквозь некоторые ячейки, располагаясь «под» другой стрелкой (без добавления «поворотов», в отличие от случая использования левой кавычки `). Например, при создании стрелки, указывающей из вершины Σ^L вниз, эта форма записи была использована, чтобы указать относительные номера пропускаемых вершин, расположенных на одну ('[d]) и три ('[ddd]) строки ниже

[1]Вариант диаграммы, набираемый с помощью XY-pic, изменен по сравнению с тем, который дан в приложении к статье, поскольку старый вариант предназначался для второй версии пакета XY-pic. Наш вариант, основанный на третьей версии, является более полным, легким для чтения и использует более однородную форму записи.

текущей. Сходные инструкции использованы, чтобы идущая из вершины Σ^G вправо стрелка-указатель проходила под другими стрелками, расположенными на один (`'[r]`) и три (`'[rrr]`) «пролета» правее. Операторы `<<<<` и `(...)` позволяют точнее регулировать размещение связанного со стрелками текста, указывая, насколько далеко от начальной точки стрелки-указателя должен быть помещен данный текст. Последняя форма особенно полезна, поскольку дает возможность учитывать небольшие расхождения в длинах стрелок, возникающие в случае элементов разной ширины, что позволяет с большой точностью выравнивать набираемый материал. Кроме этого, обратите внимание на три параметра установки в рассмотренном примере: `@=1pc` делает расстояние *между* столбцами равным 1 pc, `@*[r]` распространяет формат выравнивания `[r]` на все элементы, что приводит к хорошим с эстетической точки зрения результатам, поскольку все элементы состоят из одной прописной буквы с различными верхними и нижними индексами, а `@*[o]` придает всем элементам круглую форму, что позволяет диагональным стрелкам подходить чуть ближе к центру.

Так же просто может быть набрана диаграмма переходов:

in 


```
\usepackage [matrix,curve,arrow,tips,frame]{xy}
\[\UseTips
        \entrymodifiers={++[o][F]}
        \xymatrix @-1mm {
         *+\txt{in} \ar[r]
        & 1 \ar@(dr,dl)[]^b \ar[r]_a
        & 2 \ar@(d,dl)[]^a  \ar[r]_b
        & 3 \ar `u[l] `^d[l]_a [l] \ar[r]_b
        & *++[o][F=]{4}
          \ar `dl_l[ll]+/d6mm/`l_ul[ll]^a [ll]
          \ar `u^l[lll]+/u1cm/`l^d[lll]_b [lll]
        }\]
```

5-5-5

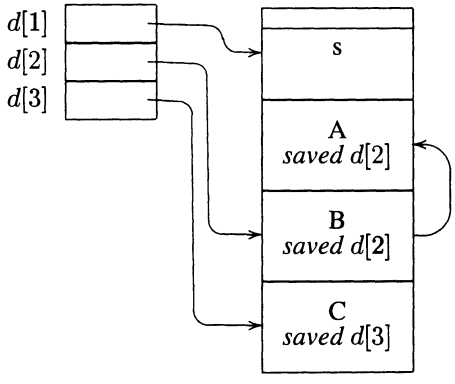
На диаграммах такого рода[1] все состояния (элементы) обведены окружностями. В данном случае задание модификатора, используемого по умолчанию для выполнения этой операции, было выполнено командой `\entrymodifiers`. Для получения красивых головок у искривленных стрелок-указателей были использованы значки из семейства Computer Modern, а для придания диаграмме более компактного вида расстояние между элементами уменьшено на 1 мм (выражением `@-1mm`, расположенным перед открывающей фигурной скобкой команды `\xymatrix`). Если описание элемента начинается со звездочки (т. е. имеет вид `*`⟨*object*⟩), то отменяются все значения, принятые по умолчанию командой `\entrymodifiers`. Это используется в самой левой ячейке для отмены рисования рамки и в самой правой ячейке для рисования двойной окружности.

[1] Наш пример основан на диаграмме детерминированного конечного автомата из Aho et al. (1986), с. 136. Другое представление этой же диаграммы можно найти в Rose (1995), разд. 3.4. Она уже использовалась в примере 3-3-6 на с. 97.

Следует заметить, что в последнем случае необходимо дать полное описание модификатора. Последним каверзным местом является задание дополнительного смещения во внешнюю область, добавившего 6 мм (в случае *a*) и 1 см (в случае *b*) при определении мест, где стрелки должны повернуть.

В качестве последнего примера рассмотрим следующую диаграмму стека:

```
\usepackage [matrix,arrow,frame]{xy}
\newcommand{\topbar}{\vrule height 0.4pt width 20mm}
\newcommand{\previous}{% turning-width of 15mm
  \save\ar`r[u]+/r15mm/`[u][u]\restore }
\newcommand{\saved}[2]{\txt{#1\\\emph{saved} $d[#2]$\\}}
\newcommand{\bendto}[2]{% creates a bendy arrow, offset 5mm
  \save c!C+/r5mm/\ar `r#1!C+/l#2/ `^r#1!C #1!C\restore}
\newcommand{\dinput}[1]{% label-offset 11mm
  \save +/l11mm/*{d[#1]}\restore}
\[\begin{xy}
\xymatrix"R" @M=0mm @H=12mm @W=20mm @R=0mm @*[F] {%
 {\txt{\topbar\\s\\\\ }}  %1,1
\\ \saved{A}{2}           %2,1
\\ \saved{B}{2}\previous  %3,1
\\ \saved{C}{3}           %4,1
}\turnradius{2mm}\POS(-30,+4)
\xymatrix @M=0mm @H=5mm @W=12mm @R=0mm @*[F] {%
   \dinput{1}\bendto{"R1,1"}{15mm}  %1,1
\\ \dinput{2}\bendto{"R3,1"}{17mm}  %2,1
\\ \dinput{3}\bendto{"R4,1"}{19mm}  %3,1
}
\end{xy}\]
```



5-5-6

Рисунок создается с помощью двух команд `\xymatrix`, некоторые элементы которых снабжены текстом или метками. Стрелки-указатели, соединяющие отдельные элементы, построены благодаря *возвратам* в уже пройденные точки, заключенным внутри пар `\save`...`\restore`, чтобы не влиять на расположение последующих элементов матриц. То, что сначала рисуется правый стек, получающий имя `"R"`, дает возможность использовать его элементы в качестве позиций при построении левого стека (некоторые из этих ячеек используются

в качестве целевых точек для стрелок-указателей, берущих начало в левом стеке). С целью достижения более полного контроля за расположением объектов внутри каждой из команд `\xymatrix` необходимо сначала избавиться от окружающих объекты полей (`@M=0mm`) и разделяющих строки (`@R=0mm`) промежутков, поместив соответствующие инструкции в *преамбуле* матрицы. Затем соответствующим образом подбираются значения высоты (`@H=...`) и ширины (`@W=...`) ячеек и вокруг каждого элемента проводится рамка (`@*[F]`).

Поскольку для различных элементов матриц требуются одни и те же структуры, удобно определить некоторые макро. Это не только уменьшает размер программы благодаря устранению повторения длинных конструкций, но и позволяет в случае необходимости вносить согласованные изменения. Вдобавок макро помогают сохранить прозрачность основной части программы, вынося детали во вспомогательную часть. Имя макро может быть выбрано в соответствии с его содержимым. Так, макро `\saved` помещает текст в три ячейки правого стека, в каждом случае изменяя всего два символа. Самая верхняя ячейка немного отличается от остальных наличием дополнительной линии, созданной с помощью макро `\topbar`. Кроме этого, идущая из третьего элемента (`"R3,1"`) во второй (`"R2,1"`) дополнительная стрелка-указатель создается путем возврата в ранее пройденную точку с помощью `\save...\restore`, определенных в макро `\previous`. Внешний вид этой загнутой стрелки контролируется с помощью смещения на 15 мм, заданного в `'r[u]+/r15mm/`.

Теперь обратимся к левому стеку, который помещается в точке с координатами `(-30,4)`. Уменьшение `\turnradius` (радиуса поворота) до 2 мм (с принятых по умолчанию 10 pt) позволяет сделать повороты более резкими. Промежутки и размеры снова определяются в *преамбуле* матрицы. Макро `\dinput` помещает каждую указанную метку смещенной на фиксированное расстояние левее (`+/l11mm/`) ранее пройденной точки. Каждая стрелка-указатель, идущая к правому стеку, строится путем задания целевой ячейки (например, `"R4,1"`), некоторого смещения от этой ячейки (в данном примере — всегда 13 мм) и смещения, определяющего, где расположен изгиб (например, 21 мм). На основе этой информации макро `\bendto` проводит соответствующую стрелку.

5.5.3 Графы

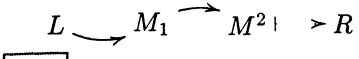
Блок-схемы, ориентированные графы, деревья и другие математические объекты, обладающие характерной структурой, можно изображать с помощью пакета **graph**, реализующего комбинаторный стиль рисования, несколько напоминающий язык pic. Пакет **graph** при работе использует пакет **arrow**, который по этой причине всегда загружается следом. Синтаксис команды рисования имеет вид

`\xygraph{`*graphdesc*`}`

где аргумент *graphdesc* описывает различные компоненты графа. Оператор `&` организует объекты в столбцы. В отличие от пакета **matrix** пакет **graph** не

обеспечивает никакого дополнительного выравнивания и не изменяет расстояний между объектами.

Основной принцип состоит в том, чтобы провести линию или стрелку-указатель из *текущей* в *целевую* вершину, после чего целевая вершина становится текущей. Основными операторами для изображения линии и стрелки-указателя являются, соответственно, - и :. За операторами в случае необходимости следует описание стрелки, вершины или метки. Сказанное станет понятнее из примера:

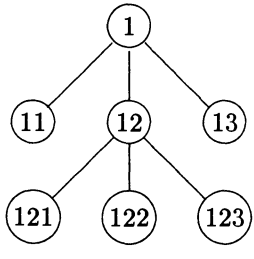


5-5-7

```
\usepackage [graph,curve]{xy}
\[\xygraph{
[]L :@/_/ [r]{M_1} :@/^/ [r]{M^2}
    :@{|.>}[r]R    :@/_1em/"L"
}\]
```

Все начинается с определения точки L и присвоения ей символического имени "L", поскольку в дальнейшем предполагается обращаться к этой точке. Затем создаются три небольшие стрелки, соединяющие точки L, M_1, M^2 и R с помощью стрелок разного вида. Наконец проводится длинная стрелка, направленная от точки R к L, благодаря упоминанию имени "L", которое автоматически относится к последней из вершин, содержащей только букву L.

Удобный синтаксис для изображения ветвей дерева имеет вид ⟨*parent*⟩(⟨*child*⟩,...,...⟨*child*⟩): каждый граф-потомок (⟨*child*⟩) набирается так, словно он идет непосредственно за графом-предком (⟨*parent*⟩). Следующий пример содержит два списка потомков, один расположен на верхнем уровне и выходит из узла 1, а второй — на втором уровне и выходит из узла 12.



5-5-8

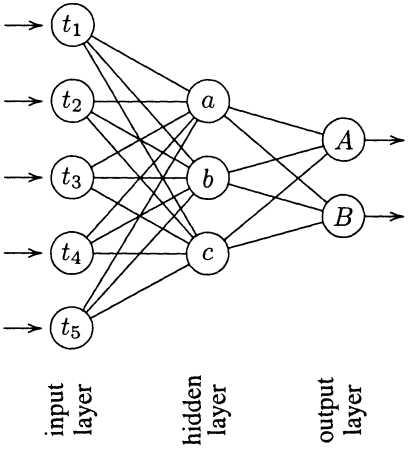
```
\usepackage [graph,frame]{xy}
\[\xygraph{
[] *+[o]+[F]{1}
  (-[dl] *+=[o]+[F]{11}
  ,-[d]  *+=[o]+[F]{12}
    (-[dl] *+=[o]+[F]{121}
    ,-[d]  *+=[o]+[F]{122}
    ,-[dr] *+=[o]+[F]{123}
    )
  ,-[dr] *+=[o]+[F]{13}
  )
}\]
```

В этом примере также показано, что любой XY-pic'овский объект может быть помещен в узле, как обычно, с помощью оператора *. На самом деле символ ! позволяет размещать *любую* конструкцию из ядра пакета XY-pic, указанную в фигурных скобках {}. Это будет проиллюстрировано в следующих двух примерах. В них представлены два «минипакета», разработанных авторами XY-pic, для рисования нейронных сетей и электронных схем на логических

элементах. Вместе с тем эти примеры иллюстрируют большинство возможностей пакета **graph**. Оба минипакета используют команду **\newgraphescape** для задания в графах новых объектов, обозначаемых **!**⟨*letter*⟩.

Сначала посмотрим на подход, избранный Россом Муром для набора нейросетевых диаграмм. Он использует парадигму структурного программирования, руководствуясь принципом, что объекты высокого уровня, с которыми придется оперировать пользователю, должны иметь знакомый вид. Для иллюстрации этого подхода выбрана простая диаграмма сети *прямого распространения* (названная так потому, что выходные сигналы нейронов каждого слоя служат входными сигналами нейронам следующего слоя):

```
\usepackage [all,dvips]{xy}
\newcommand{\Neuron}[1]{\POS*+=<1em>[o]+[F]{#1}}
\newcommand{\Link}[1]{\ar @{-} "#1"}
\newcommand{\Out}{\ar +/r8mm/}
\newcommand{\In}{\save +/l9mm/*{}\ar +/r5mm/\restore}
\newgraphescape{O}[1]{!{\Neuron{#1}="#1"\Out}}
\newgraphescape{H}[1]{!{\Neuron{#1}="#1"\Link{A}\Link{B}}}
\newgraphescape{I}[1]{!{\Neuron{#1}\In\Link{a}\Link{b}\Link{c}}}
\[\xygraph{!{0;<18mm,0mm>:<0mm,10mm>::}
 [] !O{A}       [d]!O{B}
 [dd]*[left]!U\txt<12mm>{output\\layer}="T"
"A" [u(.5)l]!H{a}
    [d]!H{b}    [d]!H{c}
"T" [l] *[left]!U\txt<12mm>{hidden\\layer}
"a" [ul]!I{t_1}    [d] !I{t_2}    [d] !I{t_3}
    [d] !I{t_4}    [d] !I{t_5}
"T"[ll] *[left]!U\txt<12mm>{input\\layer}
}\]
```



5-5-9

Сначала определяются несколько макро, имена которых напоминают описываемые ими части нейронной сети. Использование этих имен в последующей команде `\xygraph` позволяет задавать логическую структуру диаграммы, опуская при этом подробности низкого уровня реализации ее компонент средствами пакета XY-pic. Команда `\newgraphescape` дает возможность вводить новые типы узлов, помещенных после символа `!`. Как видно из приведенного выше примера, эти описывающие узлы макро могут даже иметь аргументы.

После задания единичных векторов в двух основных направлениях начинается размещение элементов нейронной сети. При движении справа налево сначала встречаются объекты A и расположенный под ним B (созданные командой `!O`), которые оба имеют по стрелке-указателю, направленной вправо в сторону от основной части диаграммы. Все это называется *output layer* (*выходным слоем*). Затем наступает очередь трех объектов a, b, c (созданных командой `!H`), которые все соединены с узлами A и B и расположены в промежуточном слое, называемом *hidden layer* (*скрытым слоем*). Наконец, наступает черед *input layer* (*входного слоя*) (созданного при помощи команды `!I`), элементы которого обозначены t_i.

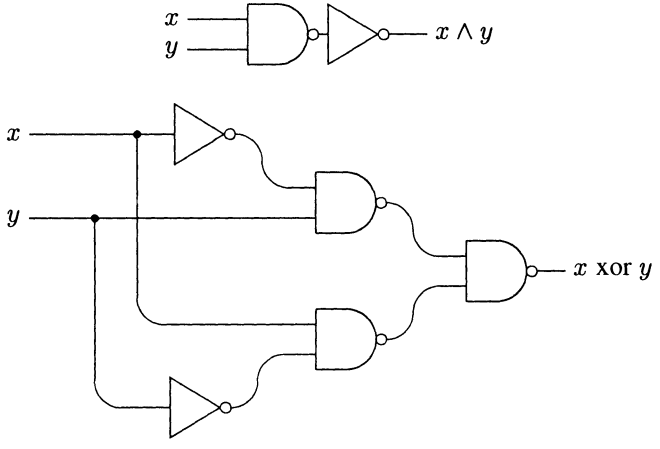
При выполнении приведенной выше программы в двух случаях вызова описывающего узел макро `!O` происходит задание меток `"A"` и `"B"`, к которым обращаются три `!H`-узла, задающих метки `"a"`, `"b"` и `"c"`. К ним в свою очередь обращаются пять команд `!I`, рисующих связи между всеми элементами входного и скрытого слоев. Для вертикального набора текста применяется спецификатор `[left]`, являющийся частью расширения `rotate` и требующий поддержки PostScript'овским драйвером.

Из этого описания должно быть очевидно, что эти или подобные им команды позволяют универсальным способом набирать все виды диаграмм, описывающих нейронные сети.

Вторым минипакетом является команда `\circuit` для набора электронных схем на логических элементах. Компонентами, которые мы рассмотрим, будут логический элемент «И-НЕ» и инвертор, обозначаемые, соответственно, `!N` и `!I`. Они определены в духе XY-pic *не зависящими от текущего направления*. Однако задано и используется только одно направление `!R` (что означает «right» (вправо)). Элемент вместе с его выходной линией помещается в текущую позицию. Он должен иметь имя, которое можно вызывать впоследствии. Кроме того, входы элементов получают имена с суффиксами `a` или `b`. Обратите внимание на то, как мало элементов в приведенных ниже двух простых примерах имеют абсолютные координаты — модульность такого способа задания очень удобна.

```
\usepackage [graph,curve,arc]{xy}
\newgraphescape{N}[1]{!{\save-/4mm/-/4pt/;p+/4mm/:
  (-1,1);(-1,-1)**@{-} ?(.25)="N#1a" ?(.75)="N#1b",
  (-1,1);(0,1)**@{-} ; (0,-1),{\ellipse_{}} ; (-1,-1)**@{-},
  (0,0);(1,0)**{}*!E\cir<2pt>{}!C-E="N#1"\restore \POS"N#1"}}
\newgraphescape{I}[1]{!{\save-/4mm/-/2pt/;p+/4mm/:
```

```
  (-1,1);(-1,-1)**@{-} ?="I#1a",  (-1,1);(.667,0)**@{-};(-1,-1)**@{-},
  (0,0);**{}*!E\cir<2pt>{}!C-E="I#1"\restore \POS"I#1"}}
\newgraphescape{B}{!{*=0@{*}}}
\newgraphescape{R}{!{;p+/r4mm/**{};}}
\newgraphescape{p}[2]{[#1!{"#2";p+/^/}]}
\newcommand{\circuit}[1]{\xygraph{~{0;<10mm,0mm>:<0mm,9mm>::0}#1}}
\[\begin{array}{c}
\circuit{  []!R!N1 ("N1a"([l]x - ?), "N1b"([l]y - ?))
  [r]!I2 ("N1" - "I2a") - [r]{x\land y}}\\
\circuit{  []x
   [rrr]!R!I1 ("I1a"("x" - ?))
    [drr]!R!N1 ("I1" -`r[d]`"N1a""N1a", "N1b"!plx y - "N1b")
   "I1"[dddd]!R!I2 ("I2a"[l]!puy!B - `d"I2a""I2a")
    "N1"[dd]!R!N2 ("I1a"[l(.5)]!pux!B -`d"N2a""N2a", "I2" -`r[u]`"N2b""N2b")
     [urr]!R!N3 ("N1" -`r[d]`"N3a""N3a", "N2" -`r[u]`"N3b""N3b")
    - [r]{x\mathrel{\textrm{xor}}y}}
\end{array}\]
```



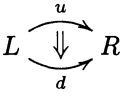
5-5-10

Обратите внимание на то, как при размещении элемента используются команды ядра, защищенные парой `\save...\restore`, для задания локальной системы координат, отражающей текущее направление. Чтобы сделать округлую часть элемента «И-НЕ» не зависящей от выбранного направления, используется команда `\ellipse` (см. разд. 5.5.6). Кроме того, следует заметить, что допускаются искривленные стрелки, изгибы которых вставляются с помощью символа `.

5.5.4 Двухэлементные диаграммы

В теории категорий часто встречаются «двуместные» морфизмы, изображаемые в виде пар изогнутых стрелок-указателей с помещенными на них или рядом с ними метками. Для набора таких диаграмм существует пакет `2cell`.

Самая простая диаграмма с двойной ячейкой имеет вид:



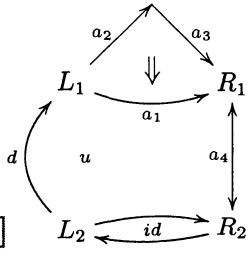
```
\usepackage [all,2cell]{xy}
\UseTwocells
\[
\xymatrix{L\rtwocell^u_d & R}
\]
```

5-5-11

В теории категорий большинство диаграмм имеют общую структуру, напоминающую матрицу, поэтому мы используем команду `\xymatrix`. Поскольку обычно все доступные для набора компонентов этих диаграмм команды не требуются, эффективнее загружать лишь подмножества реально используемых команд. В частности, инструкция `\UseTwocells` определяет команды вида `\⟨cc⟩twocell`. В приведенном выше примере команда `\rtwocell` служит для создания стрелок, указывающих направо. В качестве части имени (префикса) команды может быть использовано до трех «указателей» (`l`, `r`, `u` и `d`). Помимо этого существует команда `\xtwocell` (см. пример ниже).

В приведенном выше примере были загружены только команды, относящиеся к созданию симметричных ячеек с двумя ветками, представляющими собой две кривые. Для набора отдельных стрелок двухэлементных диаграмм следует указать `\UseHalfTwocells` (что определяет команды вида `\⟨cc⟩uppertwocell` и `\⟨cc⟩lowertwocell`). Еще более сложные (асимметричные) конструкции становятся возможными благодаря команде `\UseCompositeMaps` (определяющей команды вида `\⟨cc⟩compositemap`). Инструкция `\UseAllTwocells` обеспечивает загрузку всех возможных команд. Эти команды требуется вводить только один раз, обычно в преамбуле LaTeX'овского документа. В приведенных ниже примерах команда `\UseAllTwocells` расположена вне математической выделенной формулы, представляя собой только напоминание (она не является частью диаграммы).

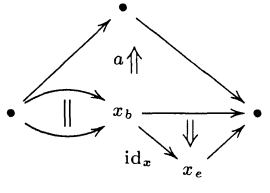
В следующем примере демонстрируются различные команды и их некоторые опции.



```
\usepackage [all,2cell]{xy}
\UseAllTwocells
\[
\xymatrix @=15mm {
  L_1 \rlowertwocell<-3>_{a_1}{<-1>}
      \rcompositemap<6>_{a_2}^{a_3}{\omit}
& R_1 \dtwocell<0>_{a_4}{"}
\\ L_2 \uuppertwocell_u^d{\omit}
       \rtwocell<2>{'id}
&  R_2
 }\]
```

5-5-12

Мы использовали команду \UseAllTwocells потому, что хотим показать все возможные формы двухэлементных диаграмм. Расстояние между строками и столбцами было установлено равным 15 мм. Степень изогнутости искривленных стрелок-указателей контролируется с помощью коэффициента «отталкивания», имеющего вид <nb>, т. е. числа, заключенного в угловые скобки. Так, для изогнутой вниз стрелки, простирающейся от L_1 до R_1, был установлен коэффициент отталкивания, равный <-3>, который уменьшает кривизну примерно вдвое по сравнению со значением по умолчанию. Чтобы быть правильно позиционированной, двойная стрелка отодвигается на одну единицу (<-1>). Затем рисуется составная стрелка, достигающая точки, на шесть единиц (<6>) превышающей принятую по умолчанию высоту, а центральная стрелка не проводится (\omit). Из второй ячейки R_1 проводится вертикальная стрелка вниз (нулевой коэффициент отталкивания (<0>) соответствует прямолинейной стрелке) и подавляется печать центральной двойной стрелки. В данном случае то, что стрелка указывает в обоих направлениях, было достигнуто благодаря указанию знака двойных кавычек (") в качестве первого символа в тексте метки. Аналогичным образом символ правой кавычки (') заставляет стрелки указывать в направлении по часовой стрелке, а символ левой кавычки (`) — в направлении против часовой стрелки, тогда как восклицательный знак (!) вообще подавляет рисование головок стрелок. Произвольный текст, помещенный в аргументе команды \rtwocell, следующий за одним из этих символов, печатается в качестве метки двухэлементной диаграммы.




```
\usepackage [all,2cell]{xy}
\UseAllTwocells
\[
\renewcommand{\objectstyle}{\scriptstyle}
\renewcommand{\labelstyle}{\scriptstyle}
\xymatrix @=1pc {
&& \bullet \ar[2,2]
\\ \\
   \bullet \ar[-2,2]
        \rrtwocell {=}
        \xtwocell[0,4]{}\omit{^<-4>a}
&&  x_b \ar[rr] \ar[dr]_{\mathrm{id}_x}
        \rrtwocell\omit{<1.5>}
&& \bullet
\\ &&& x_e \ar[ur]
}\]
```

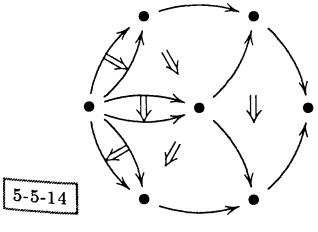
5-5-13

Приведенная выше диаграмма иллюстрирует еще некоторые возможности, доступные при построении двухэлементных диаграмм. Сначала мы установили меньший размер шрифта путем присвоения стилю меток (\labelstyle) и стилю объектов (\objectstyle) значения \scriptstyle. Затем мы уменьшили промежутки между строками и столбцами до одной пики (1pc). После

этого построим матрицу, состоящую из стрелок-указателей (соответствующих прямолинейным частям) и двухэлементных диаграмм. Обратите внимание на использование знака равенства (=) внутри команды `\rrtwocell`: он задает вместо двойной стрелки-указателя двойную линию (обозначающую равенство). Размещение в этой позиции команды `\omit` подавило бы вывод этого объекта. Для изменения направления центральной двойной стрелки-указателя по ходу часовой стрелки требуется, чтобы первым символом была «крышечка» (`^`), как в команде `\xtwocell`. По умолчанию выбирается направление по часовой стрелке (`_`).

Главная задача команды `\xtwocell` состоит в том, чтобы дать возможность «соединительным элементам» связывать ячейки, удаленные друг от друга более чем на три «пролета». Первым аргументом является место назначения (в нашем примере расположенное на четыре ячейки правее), затем следует произвольное смещение относительно центра целевой ячейки (поскольку этот аргумент пуст, местом назначения служит сам центр целевой ячейки `"3,5"`). Следующий аргумент `\omit` означает, что искривлять стрелки-указатели не требуется, а последний аргумент задает отрицательный коэффициент отталкивания и метку a. Это дает нам удобный способ позиционирования верхней двойной стрелки. То же самое относится и к двойной стрелке, расположенной справа: она нарисована с помощью команды `\rrtwocell`, первым аргументом которой является `\omit`, подавляющий искривление стрелок.

Средства для рисования двухэлементных диаграмм можно комбинировать с изогнутыми стрелками для получения следующего интересного результата:



5-5-14

```
\usepackage [all,2cell]{xy}
\UseTwocells
\[
\xymatrix @R=2.0pc @C=.8pc {
& \bullet \ar @/^1ex/ [rr]
&& \bullet \ar @/^1ex/ [dr]
\\ \bullet \urtwocell<2> \rrtwocell<2>
      \drtwocell<2> \xtwocell[1,3]{}\omit
      \xtwocell[-1,3]{}\omit
&& \bullet \ar @/_1ex/ [ur]
      \ar @/^1ex/ [dr]
      \rrtwocell\omit
&& \bullet
\\ & \bullet \ar @/_1ex/ [rr]
&&   \bullet \ar @/_1ex/ [ur]
}\]
```

Две команды `\xtwocell` создают расположенные в центре диаграммы двойные стрелки, направленные к центральному черному кружку и от него. Расположенная правее остальных двойная стрелка является единственным результатом последней команды `\rrtwocell`. Присутствующий во всех этих командах `\omit` подавляет набор изогнутых стрелок.

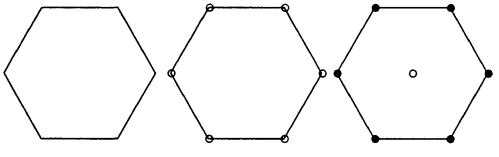
5.5.5 Многоугольники

Правильные многоугольники легко строить с помощью пакета `poly`. Кроме того, если перейти в систему координат с произвольной ориентацией осей, можно построить несколько типов неправильных многоугольников. Этот пакет использует функции пакета `arrow`, поэтому загружает последний автоматически.

Общий вид команды для рисования многоугольников таков:

`\xypolygon`⟨*nb*⟩`"`⟨*pref*⟩`"{`⟨*swit*⟩`...}`

Обязательный аргумент *nb* указывает число сторон многоугольника. Другая обязательная часть, обозначенная выше через ..., содержит описание объектов, которые должны появиться рядом с вершинами. Необязательный префикс `"`⟨*pref*⟩`"` в явном виде задает имя данного многоугольника, чтобы к нему можно было бы обратиться из любого места внутри окружения `xy`. Если первый символ внутри фигурных скобок отличен от символа тильда, то материал внутри скобок интерпретируется как объект, помещаемый рядом с каждой вершиной. Аргумент, начинающийся с `~`, обозначает наличие одного или более ключей, модифицирующих вид вершин, сторон или диагоналей. Несколько примеров сделают все это более понятным.



```
\usepackage [all,poly]{xy}
\[\begin{xy} /r10mm/:
 ,   0    ,{\xypolygon6{}}
 ,+/r22mm/,{\xypolygon6{@{o}}}
 ,+/r22mm/,{*@{o}\xypolygon6{@{*}}}
\end{xy}\]
```

5-5-15

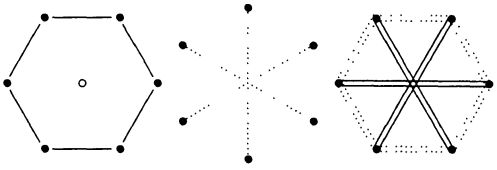
Выше представлены три варианта шестиугольников: принятая по умолчанию (без каких-либо украшений), с простыми объектами в каждой вершине и, наконец, с маленькой окружностью в центре (указывающей, что точка привязки данного шестиугольника расположена в его центре).

Не сложно получить некоторые модификации приведенных выше фигур с помощью ключей. Во-первых, при описании *вершин* можно использовать модификатор `~*{`*obj*`}`, чтобы разместить некоторый объект, и `~={`*ang*`}`, чтобы указать угол первой изображаемой вершины. Во-вторых, при наборе *диагоналей* (*сторон*) их внешний вид задается с помощью `~<{...}` (`~>{...}`), внешний вид стрелок — с помощью `~<<{`*arr*`}` (`~>>{`*arr*`}`), а внешний вид меток и разрывов — с помощью `~<>{...}` (`~><{...}`).

Давайте посмотрим, как это все работает на практике.

```
\usepackage [all,poly]{xy}
\[\begin{xy} /r10mm/:
 ,   0    ,{*@{o}\xypolygon6{~*{\dir{*}}}}
 ,+/r22mm/,{\xypolygon6{~<{.}~>{}~={30}{\dir{*}}}}
```

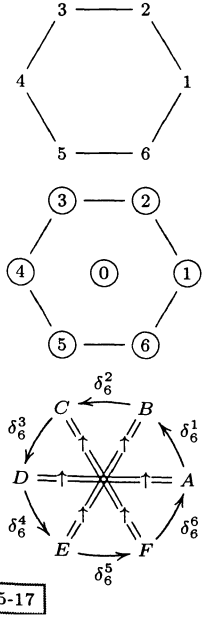
```
,+/r22mm/,{\xypolygon6{~<{=}~>{:}{\dir{*}}}}
\end{xy}\]
```



5-5-16

По умолчанию помещаемый в вершине объект набирается в боксе нулевого размера (см. третью команду в предыдущем примере, где стоит `\xypolygon6{@{*}}`). Однако из первой строки данного примера видно, что в случае наличия ключа `~*{...}` при создании объекта в каждой вершине учитывается размер поля вокруг объектов `\objectmargin`. Во второй и третьей строках определяется внешний вид диагоналей и сторон данных многоугольников. В случае пустого определения (`~>{}`) стороны не рисуются. Более того, во второй строке инструкция `~={30}` поворачивает шестиугольник на тридцать градусов. В третьей строке внешний вид диагоналей и сторон мы определили, соответственно, как `~<{=}` и `~>{:}`.

Вершины автоматически нумеруются "1", "2", ..., "nb", причем центру присваивается номер "0". При наборе меток команда `\xypolynode` указывает реальный номер стороны, диагонали или вершины в момент выполнения команды. Кроме того, с помощью команды `\xypolynum` можно набирать номера сторон. Давайте вернемся к нашему шестиугольнику и посмотрим, чего можно добиться, вооружившись этими знаниями:

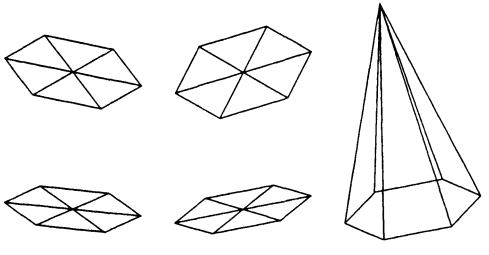


5-5-17

```
\usepackage [all,poly]{xy}
\newcounter{node}
%
\[
\renewcommand{\objectstyle}{\scriptstyle}
\newcommand{\Letter}%
    {{\setcounter{node}%
     {\xypolynode}\Alph{node}}}
%
\begin{xy} /r11mm/:
 (2,4.8), {\xypolygon6{~*{\xypolynode}}}
,(2,2.5), {*{0}*\cir<5pt>{}
                \xypolygon6{~*{\xybox{%
              *{\xypolynode}*\cir<2mm>{}}}}}
,(2,0)   , {\xypolygon6{%
           ~><{@/_.5ex/}
           ~>>{_{\delta^{\xypolynode}_{\xypolynum}}}
           ~<<{@{=}}
           ~<>{|\uparrow}
           ~*{\Letter}}}
\end{xy}
\]
```

После указания того, что объекты должны быть набраны в стиле индексов, сначала определяется команда \Letter, преобразующая номер вершины (принимающий значения от одного до шести) в прописную букву. Верхний шестиугольник демонстрирует, как в качестве вершин могут печататься их номера. В среднем шестиугольнике номер каждой вершины обведен окружностью, но чтобы это выполнить, потребовалось задать сложную командную последовательность в качестве аргумента команды \xybox. Наконец, в последнем шестиугольнике мы построили весьма сложный объект, в котором сторонами являются изогнутые стрелки, помеченные греческой буквой δ с номером вершины в качестве верхнего индекса и общим числом сторон в качестве нижнего индекса. Диагонали проведены двойными линиями, в середины которых вставлены направленные вверх стрелки. В вершинах стоят прописные буквы, соответствующие номеру вершины, который задается макро \Letter.

Существует еще один ключ, который до сих пор не был описан, а именно ~:{...}. Он позволяет изменять масштаб осей координат для построения неправильных многоугольников. По-видимому, наиболее интересно то, что он дает возможность имитировать трехмерные рисунки и рисунки с перспективой, что иллюстрирует следующий пример.



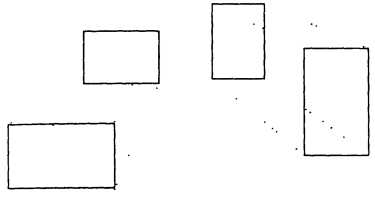
```
\usepackage [all,poly]{xy}
\[\begin{xy}/r9mm/:
  (0,0),{\xypolygon6{%
   ~:{(1,-.1):(0,.33)::}~<{-}}}
 ,(0,2),{\xypolygon6{%
   ~:{(1,-.2):(0,.5)::}~<{-}}}
 ,(2.5,0),{\xypolygon6{%
   ~:{(1,.2):(0,-.3)::}~<{-}}}
 ,(2.5,2),{\xypolygon6{%
   ~:{(1,.3):(0,-.6)::}~<{-}}}
 ,(5,0)="O", +(-.5,3)="T","O"
 ,{\xypolygon6{~:{(1,0.2):(0,.4)::}
   ~<>{;"T"**@{-}}}}
\end{xy}\]
```

5-5-18

Слева показаны результаты четырех различных преобразований координат, примененных к шестиугольнику. Чтобы понять смысл использования записи с двоеточием внутри аргументов команд \xypolygon, сначала рассмотрим первый шестиугольник, у которого конец единичного вектора, отвечающего оси x, перенесен в точку с координатами $(1,-.1)$, а конец единичного вектора, отвечающего оси y, — в точку с координатами $(0,.33)$, порождая, таким образом, эффект деформации. На рисунке справа «начало координат» "O" располагается в позиции $(5,0)$, а вершина "T" — в позиции $(-.5,3)$ по отношению к этой точке. При построении шестиугольника дополнительные линии, проведенные к вершине "T", нарисованы благодаря использованию ключа ~<>{...}. Это выполняется так: сначала точка с запятой загружает вершину основания в качестве «предыдущего» объекта p, затем определяет в качестве «текущего»

объекта *c* вершину "T" и, наконец, проводит между ними соединительный элемент **@{-}.

Другой областью применения этого пакета являются рисунки с перспективой. В следующем примере также показано, как можно использовать префиксы для идентификации различных многоугольников.

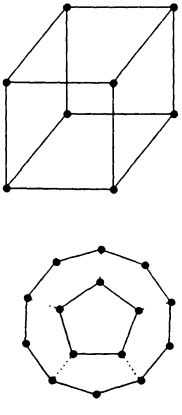


5-5-19

```
\usepackage [all,poly]{xy}
\[\begin{xy} /r1cm/:
  {\xypolygon4"F"{~:{(0,.6)::}}},+(.8,1.3)
 ,{\xypolygon4"B"{~:{(.7,0):(0,.7)::}}}
 ,"F1";"B1"**@{.} ,"F2";"B2"**@{.}
 ,"F3";"B3"**@{.} ,"F4";"B4"**@{.}
\end{xy}
\qquad
\begin{xy} /1cm/:
  {\xypolygon4"F"{~:{(0,.6)::}}},+(.8,1.3)
 ,{\xypolygon4"B"{~:{(.7,0):(0,.7)::}}}
 ,"F1";"B1"**@{.} ,"F2";"B2"**@{.}
 ,"F3";"B3"**@{.} ,"F4";"B4"**@{.}
\end{xy}\]
```

На этих двух рисунках сначала базис делается асимметричным. У прямоугольника на переднем плане масштаб вдоль оси y составляет 60% от масштаба вдоль оси x, а у прямоугольника на заднем плане размер стороны вдоль оси x уменьшен до 70%, а масштаб вдоль оси y дополнительно уменьшен до 70%, чтобы создать впечатление перспективы. Поскольку прямоугольникам присвоены имена "F" (front (передний)) и "B" (back (задний)), к различным вершинам можно обращаться, используя эти имена в качестве префикса к номерам вершин. После этого нетрудно соединить соответствующие вершины прямоугольников, и получить эффект трехмерного ящика.

Модификация этих операций позволяет перейти к вложенным многоугольникам:

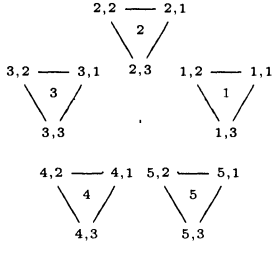


5-5-20

```
\usepackage [all,poly]{xy}
\[\begin{xy} /r10mm/:,
  {\xypolygon4"F"{\dir{*}}}
 ,+(.8,1.0),{\xypolygon4"B"{\dir{*}}}
 ,"F1";"B1"**@{-},"F2";"B2"**@{-}
 ,"F3";"B3"**@{-},"F4";"B4"**@{-}
 ,-(1.0,2.8)
 ,{\xypolygon10"O"{~={16}\dir{*}}}
 ,{\xypolygon5"I"{~:{(0.55,0):}
      ~={16}{\dir{*}}}}
 ,"O1";"I1"**@{.},"O3";"I2"**@{.}
 ,"O5";"I3"**@{.},"O7";"I4"**@{.}
 ,"O9";"I5"**@{.}
\end{xy}\]
```

После того как иы научились смещать прямоугольники для достижения эффекта глубины, теперь абсолютно ясно, как построить кубическую форму: надо всего лишь поместить два квадрата на некотором расстоянии друг от друга, а потом соединить соответствующие вершины прямоугольников на переднем ("F") и заднем ("B") планах. В нижней части данного рисунка показано, как, потратив такое же количество усилий, можно нарисовать правильный пятиугольник внутри правильного десятиугольника, просто разместив их в одной и той же точке. Поскольку нам захотелось, чтобы верхние вершины обоих многоугольников смотрели вверх, оба они повернуты на 16°. Из эстетических соображений внутренний многоугольник уменьшен (в нашем случае до 55%) и, наконец, вершины внутреннего ("I") пятиугольника соединены с соответствующими вершинами внешнего ("O") десятиугольника.

Этот раздел, посвященный многоугольникам, мы завершим демонстрацией того, как их можно вкладывать друг в друга. В каждой вершине правильного пятиугольника будет нарисован правильный треугольник.




```
\usepackage [all,poly]{xy}
\[\renewcommand{\objectstyle}
                {\scriptscriptstyle}
\begin{xy}
\xypolygon5{~:{/r12mm/:}
  ~<>{\xypolygon3{~:{(0.45,0):}
      ~*{\xypolyname\xypolynode}
      ~={30}}}
  ~>{}
  ~<{.}
  =<3mm>[o]{\xypolynode}}
\end{xy}\]
```

5-5-21

Внутри «внешнего» пятиугольника в каждой вершине находится треугольник, размер которого составляет 45% от размера пятиугольника. Каждая вершина треугольника помечена номером вершины, которому предшествует команда `\xypolyname`, добавляющая номер вершины объемлющего многоугольника: так, метка вида "4,2" указывает на вторую вершину треугольника, находящегося на четвертой вершине пятиугольника. Все треугольники ориентированы под одним и тем же углом 30° (`~={30}`). Кроме этого, мы отключили рисование сторон и соединили центры треугольников с центром пятиугольника пунктирными линиями. В каждой из вершин пятиугольника помещается ее номер, обведенный невидимой окружностью радиуса 3 мм.

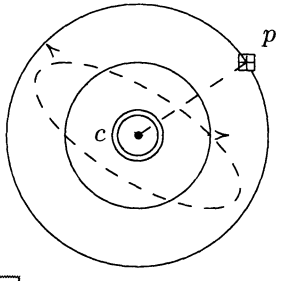
5.5.6 Дуги, окружности и эллипсы

Пакет `arc` делает доступным множество вариантов окружностей и эллипсов. В общем случае могут быть заданы дуги окружностей, соединяющие две точки или охватывающие некоторый угол с центром в заданной точке. Это свойство опирается на расширение `curve`, обсуждавшееся в разд. 5.4.1, поэтому не за-

будьте убедиться, что это расширение тоже загружено. Как было замечено ранее, процесс рисования кривых может очень быстро исчерпать ресурсы TEX'a, так что вам следует использовать такое устройство вывода (dvi-драйвер), которое прямо поддерживает рисование кривых.

Простые окружности и эллипсы

С помощью команды \ellipse становится возможным построение линиями произвольного типа (включая изогнутые стрелки-указатели) окружностей произвольного радиуса. Базовая форма команды \ellipse рисует «окружность» с центром в текущей точке. Как показано в следующем примере, у этой команды есть несколько разновидностей.



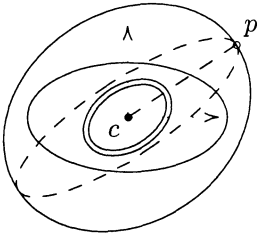
5-5-22

```
\usepackage [all,arc,dvips]{xy}
\[\begin{xy} /r12mm/:
   (0,0)*@{*}="c",*++++!R{c}
  ,(1.2,.8)*+[F]@{+}="p"
  ,*++!LD{p},"p";"c"**@{--}
  ,{\ellipse<>{}}
  ,{\ellipse(0.8){}}
  ,0;(-1,1)::
  ,{\ar@{.>} 0;(1,0)}
  ,{\ar@{.>} 0;(0,1)}
  ,"p";"c",{\ellipse(.8){--}}
  ,{\ellipse<3mm>{=}}
\end{xy}\]
```

Построение начинается с задания прямоугольного базиса, векторы которого в обоих направлениях имеют длину 12 мм. Выбираются точки p (сокращение от слова «previous» (предыдущая)) и c (сокращение от слова «current» (текущая)), причем для удобства последняя помещается в начало координат. Наиболее простой способ построения окружности иллюстрирует первая команда \ellipse, создающая окружность с центром в текущей точке и радиусом $\overline{cp}$. Поскольку внутри фигурных скобок ничего нет, окружность проводится принятым по умолчанию типом линии (сплошной линией). Обратите внимание, что окружность не пересекает объект, расположенный в точке p. Вторая команда \ellipse рисует окружность, радиус которой задан долей текущей длины базисного вектора (в нашем случае — это 80% от 12 мм). После этого мы деформируем систему координат и пунктиром проводим оси координат. Затем с помощью команды, сходной с предыдущей, но определяющей штриховой тип линий, строится эллипс. Здесь наглядно демонстрируется результат применения деформированного базиса, который, однако, не оказывает влияния на выполнение последней команды \ellipse, в которой радиус указан в угловых скобках.

Поскольку использование непрямоугольной системы координат бывает не всегда удобным и даже нежелательным, эллипсы можно определять другими

вариантами команды \ellipse. В них вам обычно необходимо указать длины и расположение осей эллипса. Как и в предыдущем примере, текущая точка *c* расположена в центре эллипса, а вектор, соединяющий предыдущую точку *p* с точкой *c*, задает направление одной из осей.




```
\usepackage [all,arc,dvips]{xy}
\[\begin{xy} 0;/r12mm/:
   {\ar@{.>} 0;(1,0)}
  ,{\ar@{.>} 0;(0,1)}
  ,(0,0)*@{*}="c",*+!RU{c}
  ,(1.2,0.8)*@{o}="p",*+!LD{p}
  ,"p";"c"**@{--}
  ,{\ellipse(1.1,.6){}}
  ,{\ellipse(,0.8){}}
  ,{\ellipse<6mm,4mm>{=}}
  ,{\ellipse<,5mm>{--}}
\end{xy}\]
```

5-5-23

Построение начинается с той же самой системы координат, что и в предыдущем примере, и так же определяются точки *c* и *p*, но теперь нулевого размера. Если перед обязательным аргументом стоят круглые скобки, в качестве единицы длины используется длина базисного вектора. В том случае, когда размеры заданы явно, обе оси эллипса располагаются вдоль координатных осей, а их длины указываются как доли длин базисных векторов. Когда перед запятой отсутствует число, одна из осей эллипса располагается параллельно линии $\overline{cp}$, а масштаб перпендикулярной оси устанавливается в соответствии с числом, указанным *после* запятой. Следовательно, в приведенных выше примерах первый эллипс ориентирован в соответствии с направлениями осей координат, длина его горизонтальной оси равна 1.1, а длина вертикальной оси — 0.6 длины базисных векторов. Ось второго эллипса перпендикулярна $\overline{cp}$, а ее длина составляет 0.8 длины базисного вектора. В другом основном варианте команды \ellipse перед фигурными скобками ставятся угловые скобки и в них явным образом задаются настоящие длины осей в единицах длины LaTeX'а. В этом случае эллипс всегда ориентирован в соответствии с направлением $\overline{cp}$. Если первый размер отсутствует, то (как и в случае с круглыми скобками) линия $\overline{cp}$ становится одной из осей, а размер, указанный после запятой, относится к перпендикулярной оси.

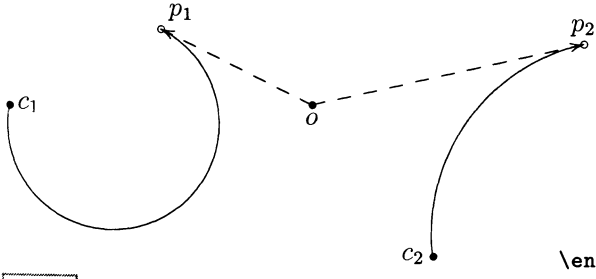
Построение дуг

Часто вам требуется не только набирать целую окружность или эллипс, а также строить дуги окружностей или эллипсов. В общем случае встречаются две ситуации: (i) заданы концевые точки, а радиус неопределен; (ii) радиус известен, а концевые точки надо найти.

В действительности, на практике в большинстве случаев концевые точки известны. Однако в силу того обстоятельства, что через них можно провести

бесконечно много дуг окружностей (эллипсов), требуется дополнительная информация. Например, для того чтобы однозначно описать дугу, вы можете задать угол касательной в одной из концевых точек. Этот способ реализован путем определения текущего направления в точке p, т. е. направление задается самым последним соединительным элементом или устанавливается равным направлению «вверх», когда такой элемент отсутствует.

Рассмотрим пример.



5-5-24

```
\usepackage [all,arc,dvips]{xy}
\[\begin{xy} 0;/r20mm/:
      (0,0)="o" ,*@{*}
     ,*+!U{o}
     ,(-1.0,0.5)="p"*@{o}
     ,*+!LD{p_1}
     ,{\ar @{-->} "o";"p"};
     (-2.,0.)="c",*@{*}
     ,*+!L{c_1}
     ,"c",{\ellipse_{}}
     ,{\ellipse^{.}}
     ,(1.8,0.4)="p",*@{o}
     ,*+!D{p_2}
     ,{\ar @{-->} "o";"p"};
     (0.8,-1)="c",*@{*}
     ,*+!R{c_2}
     ,"c",{\ellipse{}}
\end{xy}\]
```

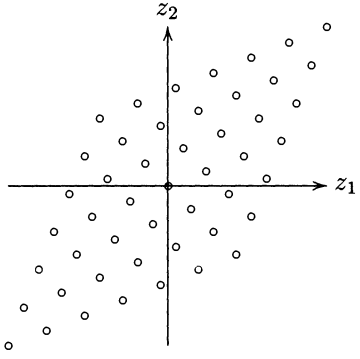
После задания точек `"o"` и `"p"` (в p_1) рисуется стрелка-указатель, одновременно задается направление $\overline{op}$, которое будет далее использовано в качестве угла касательной дуги окружности. После этого задается текущая точка (c_1) и с помощью команды `\ellipse` точки p_1 и c_1 соединяются дугой окружности. Обратите внимание, что команда с символом подчеркивания рисует дугу по часовой стрелке, а команда с символом «крышечка» (`^`) рисует дугу против часовой стрелки (пунктирная часть окружности). Похожим образом определяются точки p_2 и c_2 и проводится дуга окружности в правой части рисунка. В случае отсутствия упомянутых символов дуги проводятся против часовой стрелки.

В более общем случае дугу можно строить на основе углов касательных в ее концевых точках, а также можно задать другие типы кривых, такие, как отрезки парабол или кубических кривых, «интерполяционные» сплайны Безье или «возвратные» уравнения третьего порядка (дальнейшие подробности см. в Rose and Moore (1995)).

5.5.7 Решетки и паутинообразные структуры

С помощью пакета web можно манипулировать двумерными решетками и другими структурами, напоминающими сеть. В настоящий момент возможности этого пакета ограничены способностью располагать объекты в узлах дискретной решетки. Эту решетку можно деформировать, т. е. ее базис не обязан быть прямоугольным. При этом может использоваться любой базис системы координат, определяемый пакетом XY-pic.

Простейшей командой является \xylattice, четыре целочисленных аргумента которой описывают, какая часть решетки должна быть нарисована (на самом деле они задают позиции левого нижнего и правого верхнего углов решетки в виде величин, кратных длинам базисных векторов):

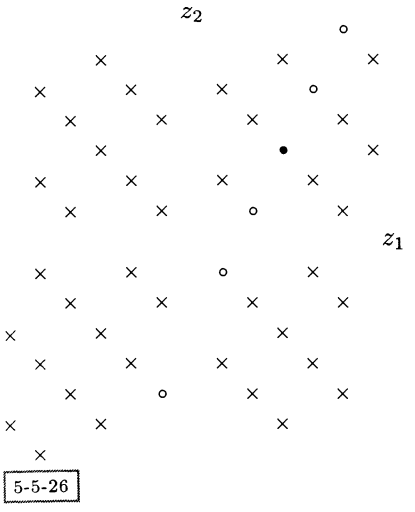



```
\usepackage [all,web]{xy}
\[\renewcommand{\latticebody}{\drop@{o}}
\begin{xy}
 *\xybox{0;<5mm,2mm>:<2mm,5mm>::
 ,0,{\xylattice{-3}{3}{-3}{3}}}="S"
 ,{"S"+L \ar "S"+R*+!L{z_1}}
 ,{"S"+D \ar "S"+U*+!D{z_2}}
\end{xy}\]
```

5-5-25

Команда \latticebody раскрывается в каждом узле решетки, поэтому ее можно использовать для размещения объектов (команда \drop похожа на *, но является автономной). В приведенном выше примере для указания местоположения узлов была выбраны кружки. Команда \xybox может использовать свой собственный базис системы координат при построении решетки, изолированный от остальной части рисунка. Так, в примере выше базисные векторы устанавливаются равными <5mm,2mm> и <2mm,5mm>, после чего рисуются все узлы решетки, расположенные между -3 и $+3$ в направлениях обоих базисных векторов. Чтобы размеры получившегося в результате бокса можно было учесть при проведении координатных осей z_1 и z_2, ему присваивается имя "S".

Иногда возникает потребность расположить объекты в определенных узлах решетки. Для определения позиции узла решетки служат команды \latticeA и \latticeB, которые возвращают его «координаты», и команды \latticeX и \latticeY, которые возвращают его смещения вдоль осей x и y (в пунктах) по отношению к центральной точке решетки. Кроме того, размер рисунка может быть ограничен командой \croplattice, четыре дополнительных параметра которой указывают отсекающий прямоугольник, снаружи которого узлы решетки не рисуются.




```
\usepackage [all,web]{xy}
\usepackage {ifthen}
\[\renewcommand{\latticebody}{%
  \ifthenelse{\latticeA=1}{%
    \ifthenelse{\latticeB=1}
                    {\drop{\dir{*}}}%
                    {\drop{\dir{o}}}}%
    {\drop{\dir{x}}}}
\begin{xy}
 *\xybox{0;<8mm,4mm>:<4mm,8mm>::
 ,0,{\croplattice{-6}{6}{-6}{6}%
                  {-3}{3}{-3.5}{3.5}}%
     }="S"
 ,{"S"+L \ar "S"+R*+!L{z_1}}
 ,{"S"+D \ar "S"+U*+!D{z_2}}
\end{xy}\]
```

Задающие отсекающий прямоугольник четыре числа не обязаны быть целыми (как видно из приведенного выше примера): они определяют интервалы x- и y-координат, внутри которых узлы решетки должны быть расположены в точках с координатами, кратными базисным векторам решетки. В том случае, когда объект должен быть помещен в определенном узле решетки, зависящем от его x- и y-координат, возможно (как было уже отмечено), удобнее использовать команды `\latticeX` и `\latticeY`. Учтите, что для использования конструкции `\ifthenelse` необходимо загрузить пакет ifthen.

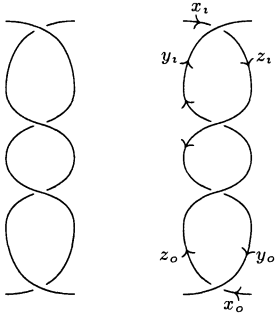
5.5.8 Звенья и узлы

Исследования струн стали очень популярны во многих областях физики и математики, и пакет XY-pic предоставляет интересный набор инструментов для составления различных схем расположения разнообразных узлов, пересечений струн и звеньев.

Пакет `knot` предлагает два типа базовых строительных блоков: «пересечения», позволяющие одной струне пройти над или под другой, и «соединения», позволяющие состыковать концевые точки струн. Пакет `knot` использует расширение `curve` и пакет `arrow`, поэтому их все следует загружать одновременно. Кроме того, обработка диаграмм узлов является наиболее трудоемкой и требовательной к ресурсам операцией из всех, обсуждавшихся до сих пор, поэтому такие диаграммы часто приходится выносить на отдельные страницы или создавать в виде отдельных инкапсулированных PostScript'овских файлов. Последние могут затем использоваться при последующих запусках LaTeX'а, что дает возможность сэкономить и время, и ресурсы (обсуждение этого метода см. в разд. 11.6 на с. 529).

Построение пересечений

Струны «пересекаются» тогда, когда близко подходят друг к другу, не касаясь при этом. Отсюда следует, что существуют три вида пересечений: струна может пройти *над*, *под* или *вдоль* другой струны. Эти варианты в различных конфигурациях систематизированы в руководстве по использованию пакета XY-pic. Будьте внимательны, чтобы не перепутать варианты команд «h» и «v», которые служат строительными блоками для набора в *горизонтальном* (текущая точка перемещается вверх и вправо) и в *вертикальном* (текущая точка перемещается вниз и влево) направлениях.



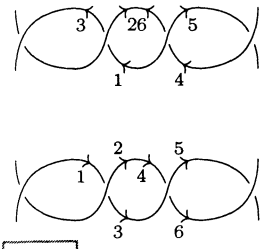

```
\usepackage [all,poly,knot,dvips]{xy}
\[\UseTips
\renewcommand{\objectstyle}{\scriptstyle}
\renewcommand{\labelstyle}{\scriptstyle}
\begin{xy} /r9mm/:
  \vover\vcross\vcross\vover-
\end{xy}
\hspace{1cm}
\begin{xy} /r9mm/:
  \vunder<><{x_i}|>|{y_i}>>>{z_i}%
  \vtwist|><<\vtwist
  \vunder-<><{x_o}|>|{y_o}>>>{z_o}%
\end{xy}\]
```

5-5-27

В приведенном примере показаны некоторые комбинации вертикальных пересечений. Слева представлен результат выполнения самих команд, а справа показано, как можно добавить метки и стрелки. Обратите внимание, что команды `\⟨c⟩twist` и `\⟨c⟩cross` создают пересечения струн в противоположных направлениях. Обратите внимание, что любой конец струны может служить исходной точкой изогнутой стрелки-указателя; различие становится важным, когда возникает необходимость размещать указатели и метки. Указание знака переноса (или знака «минус») в качестве первого символа, следующего сразу же после имени любой части, создает зеркальное отражение этой части. Оно может быть идентичным исходному, но ориентация струны будет противоположной (см. следующий пример).

Расположение меток и указателей контролируется операторами < и >, которые должны предшествовать объекту, помещаемому на начальную или конечную часть пересекающейся струны. Оператор | применяется для указания материала, который следует добавить к пересекающейся строке. Если первым символом, следующим за <, > или |, оказывается другой > (или <), то в заранее определенном месте помещается указатель стрелки, повернутый в «естественном» или в обратном направлении. Эти операторы могут повторяться любое необходимое число раз. Изучите текст программы в приведенном выше примере, чтобы разобраться, каким образом на пересечениях узлов были помещены каждый указатель и каждая метка, а полный перечень строительных блоков, их возможных ориентаций и используемых по умолчанию позиций меток на них см. в Rose and Moore (1995).

Горизонтальные ряды узлов могут создаваться командами, аналогичными описанным выше «вертикальным» командам построения узлов. В следующем примере показано, что одна и та же конфигурация узлов может быть построена путем замены команд типа `..cross..` командами типа `..twist...` Однако благодаря помеченным стрелкам можно видеть, что эти внешне одинаковые кривые в действительности имеют разную ориентацию и позиции для размещения меток.



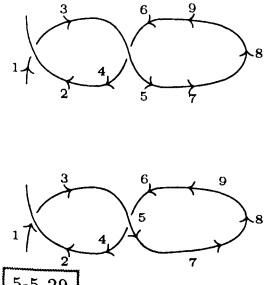
5-5-28

```
\usepackage [all,poly,knot,dvips]{xy}
\[\UseTips
\renewcommand{\labelstyle}{\scriptstyle}
\begin{xy} 0;/r8mm/:
  (0,0)   ,{\hover\hcross<>><{1}|>|{2}>>>{3}%
            \hcross<>><{4}|>|{5}>>>{6}\hover-}
 ,(0,-2.5) ,{\hover\htwistneg<>><{1}|>|{2}>>>{3}%
            \htwistneg<>><{4}|>|{5}>>>{6}\hover-}
\end{xy}\]
```

Добавление соединений

Концы пересекающихся струн могут быть соединены с помощью «соединений», а именно *петель* и *шапок*. Следующие два примера иллюстрируют использование соединений с горизонтальными или вертикальными командами пересечений типа `..cross..` и `..twist..` .

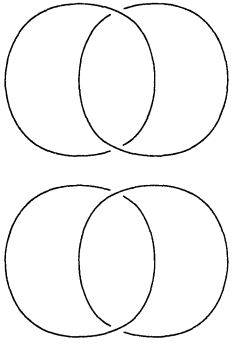
Как и в случае пересечений, на соединениях можно размещать метки и указатели стрелок. Поскольку теперь струна представляет собой только один отрезок, оператор | соответствует центру кривой, тогда как операторы < и > соответствуют позициям *до* и *после* центральной точки. Каждому отрезку струны может быть сопоставлен коэффициент масштабирования, расположенный в квадратных скобках сразу же после имени команды. Более того, позиция метки и указателя стрелки может быть изменена с помощью числа от 0 до 1, расположенного между оператором и набираемым объектом, или путем добавления/вычитания небольшой величины:



5-5-29

```
\usepackage [all,poly,knot,dvips]{xy}
\[\UseTips
\renewcommand{\labelstyle}{\scriptscriptstyle}
\begin{xy} /r9mm/:
 ,(0,0) ,{\hunder<>><{1}|>|{2}>>>{3}%
           \htwist<<<{4}|>|{5}><>{6}%
            \hloop<>><{7}|>|{8}>>>{9}}
 ,(0,-2.5) ,{\hunder<(+0.1)><{1}|>|{2}>>>{3}%
            \htwist<<<{4}|(-.2)>|(-.2){^{5}}><>{6}%
             \hloop<(+.1)><{7}|>|{8}>>>(-.1){9}}
\end{xy}\]
```

Здесь скомбинированы два горизонтальных пересечения и одна команда соединения. Верхний рисунок демонстрирует принятые по умолчанию позиции меток и указателей. При создании нижнего рисунка использовалась тонкая настройка положения указателей и меток. Положительные величины вызывают смещение в «естественном» направлении. Обратите внимание на использование символа ^ в качестве первого символа метки «5», что приводит к ее размещению «над» стрелкой, тогда как используемый по умолчанию символ _ помещает ее «под» стрелкой.

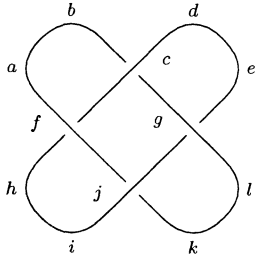


```
\usepackage [curve,knot,graph,dvips]{xy}
\[\xygraph{!{0;/r10mm/:}
          !{\vover}
      [u] !{\hcap[-2]}
      [d] !{\vover-}
    [ruu] !{\hcap[2]}
}\]
\[\begin{xy} 0;/r10mm/:
  ,\hcap[-2]\vunder\vunder-
  ,+(1,2),\hcap[2]
\end{xy}\]
```

5-5-30

Поскольку по умолчанию все пересечения узлов ограничены прямоугольниками со стороной, равной длине одного базисного вектора, и поскольку петли и шапки не меняют текущей точки, для составления единой конфигурации из пересечений и соединений удобно воспользоваться пакетом graph. Это продемонстрировано в верхней части приведенного выше примера, где команды \vover и \hcap позиционируют элементы с помощью «относительных» перемещений (вверх, вниз, влево, вправо). В нижней части представлена модификация этой диаграммы, на которой при позиционировании последней команды \hcap смещение явно указано в абсолютных величинах. Обратите внимание на использование масштабирующих коэффициентов [2] и [-2].

Существуют также команды для комбинирования блоков, в которых струны в основном располагаются под углом 45°, как в следующем примере.



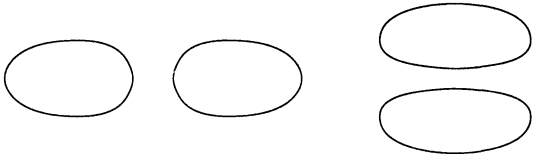

```
\usepackage [curve,knot,arrow,dvips]{xy}
\[\renewcommand{\labelstyle}{\scriptstyle}
\begin{xy} 0;/r8mm/:
 ,{\xcapv-|{a}}          , +(0,1)
 ,{\xcaph|{b}\xunderh|{c}\xcaph|{d}\xcapv|{e}}
 ,-(3,0) ,{\xoverh|{f}}   ,+(1,0),{\xoverh|{g}}
 ,-(3,1) ,{\xcapv-|{h}\xcaph-|{i}}
 ,+(0,1) ,{\xunderh-|{j}} ,+(0,-1),{\xcaph-|{k}}
 ,+(0,1) ,{\xcapv|{l}}
\end{xy}\]
```

5-5-31

Порядок размещения различных частей этой конструкции легко понять, если проследить за метками.

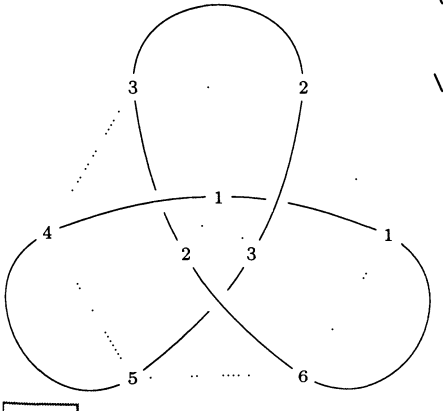
Кроме этого существуют некоторые «изогнутые» блоки, дающие возможность легко соединять блоки, расположенные под углом 45°, с вертикальными или горизонтальными блоками. Однако с помощью неортогонального базиса можно добиться результатов еще более общего вида.

```
\usepackage [curve,knot,dvips]{xy}
\[\begin{xy} /r2cm/:(0,.5)::
 ,{\hcap-\huncross\hcap}
 ,+(1,0),{\vcap\vuncross\vcap-}
\end{xy}\]
```



5-5-32

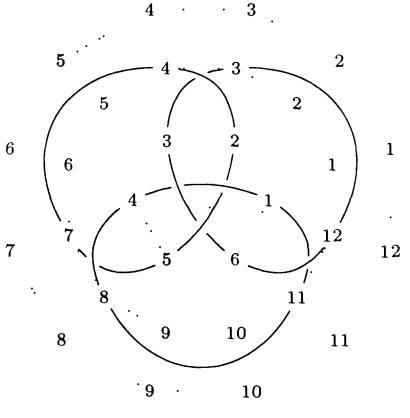
Наибольшее разнообразие формы узлов получается, если каждому блоку с помощью ключа ~: присваивается собственная система координат. Остальные примеры этой главы служат иллюстрацией этого приема; в этих примерах дополнительно используется команда \xypolygon из пакета poly (в виде отдельного макро !P, входящего в этот графический пакет).



5-5-33

```
\usepackage [all,knot,poly,dvips]{xy}
\[
\renewcommand{\labelstyle}{\scriptstyle}
\renewcommand{\objectstyle}{\scriptstyle}
\knotholesize{3mm}
\renewcommand\Vcap[2]{\save 0;#2-#1:#1,
  \vcap~{#1+(0,1)}{#2+(0,1)}{#1}{#2}
  \restore}
\xygraph{!{0;/r5mm/:}
  !P3"t"{~>{.}~*{\xypolynode}}
  !P6"h"{~:{(4.5,0):}~>{.}~*{\xypolynode}}
  !{\xunderv~{"h2"}{"h1"}
              {"t1"}{"t3"}@(.62)}
  !{\xunderv~{"h4"}{"h3"}
              {"t2"}{"t1"}@(.62)}
  !{\xunderv~{"h6"}{"h5"}
              {"t3"}{"t2"}@(.62)}
  !{\Vcap{"h3"}{"h2"}
    \Vcap{"h5"}{"h4"}%
    \Vcap{"h1"}{"h6"}}
}\]
```

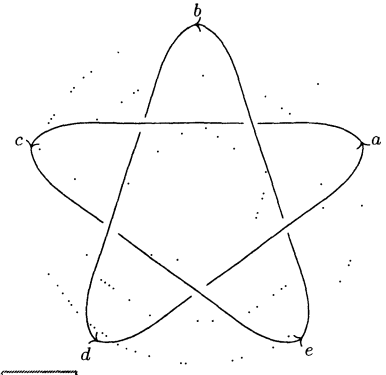
Эта трехлистная фигура нарисована с помощью вершин, определяемых внутренним (показанным пунктиром) треугольником (получившим имя "t") и внешним (показанным пунктиром) шестиугольником (получившим имя "h"). Чтобы сделать этапы построения наглядными, также показан номер каждой вершины. Используются три пересечения типа \xunderv, а применение символа ~ позволяет точно регулировать их положение между парами вершин треугольника и шестиугольника (подробности см. в разд. 5.5.5). Для построения внешних шапок необходимо нормировать базисный вектор, поскольку команда \vcap создает перемычку, длина которой вдоль оси x составляет одну единицу измерения. По этой причине внутри команды \Vcap осуществляется изменение базиса, которое не оказывает влияния на остальную часть диаграммы благодаря паре команд \save...\restore. Обратите внимание на коэффициент масштабирования 4.5, встречающийся внутри описания шестиугольника, на тонкую регулировку положения разрыва в месте пересечения с помощью коэффициента @(.62) и на задание протяженности разрыва командой \knotholesize.




```
\usepackage [all,knot,poly,dvips]{xy}
\renewcommand{\labelstyle}{\scriptstyle}
\renewcommand{\objectstyle}{\scriptstyle}
\[\xygraph{!{0;/r2mm/:}
  !P3"t"{~>{.}}
  !P6"h"{~:{(4.5,0):}~>{.}~*{\xypolynode}}
  !P12"d"{~:{(9,0):}~>{.}~*{\xypolynode}}
  !P12"D"{~:{(13.,0):}~>{.}~*{\xypolynode}}
  !{\xoverv~{"h2"}{"h1"}{"t1"}{"t3"}@(.62)}
  !{\xoverv~{"h4"}{"h3"}{"t2"}{"t1"}@(.62)}
  !{\xoverv~{"h6"}{"h5"}{"t3"}{"t2"}@(.62)}
  !{\vover~{"d4"}{"d3"}{"h3"}{"h2"}}
                ,"d4"-@`{"D5","D6"}"d7",
  !{\vover~{"d8"}{"d7"}{"h5"}{"h4"}}
                ,"d8"-@`{"D9","D10"}"d11",
  !{\vover~{"d12"}{"d11"}{"h1"}{"h6"}}
                ,"d12"-@`{"D1","D2"}"d3"
}\]
```

5-5-34

Приведенный выше рисунок немного более сложен, так как содержит четыре многоугольника. Расположенные в центре треугольник и шестиугольник похожи на те, что обсуждались выше. Но затем добавляется второй уровень пересечений, определяемых парами вершин шестиугольника и внутреннего двенадцатиугольника (получившего имя "d"). Для того чтобы замкнуть незамкнутые струны, соответствующие вершины с помощью обсуждавшейся в разд. 5.5.1 операции @` соединяются кривыми, где в качестве контрольных точек используются вершины *внешнего* двенадцатиугольника (получившего имя "D").



5-5-35

```
\usepackage [all,knot,poly,dvips]{xy}
\[
\UseTips
\knotholesize{2mm}
\xygraph{!{0;/r1cm/:}
  !P5"p"{~>{.}}
  !P10"d"{~:{(1.7,0):}~>{.}}
  !P20"D"{~={-9}~:{(2.2,0):}~>{.}}
  !{\xunderv~{"d3"}{"d2"}{"p2"}{"p1"}}
  !{\xunderv~{"d5"}{"d4"}{"p3"}{"p2"}}
  !{\xunderv~{"d7"}{"d6"}{"p4"}{"p3"}}
  !{\xunderv~{"d9"}{"d8"}{"p5"}{"p4"}}
  !{\xunderv~{"d1"}{"d10"}{"p1"}{"p5"}}
  !{\vloop~{"D3"}{"D2"}{"d2"}{"d1"}|>|{a}}
  !{\vloop~{"D7"}{"D6"}{"d4"}{"d3"}|>|{b}}
  !{\vloop~{"D11"}{"D10"}
           {"d6"}{"d5"}|>|{c}>}
  !{\vloop~{"D15"}{"D14"}
           {"d8"}{"d7"}|>|{d}}
  !{\vloop~{"D19"}{"D18"}
           {"d10"}{"d9"}|>|{e}}
}\]
```

Наконец, следующая пятиконечная звезда получается с помощью пяти-, десяти- и двадцатиконечных симметричных многоугольников (показанных пунктирными линиями). Внутреннему пятиугольнику присвоен идентификатор `"p"`, среднему десятиугольнику — `"d"`, а внешнему многоугольнику — `"D"`. Углы относительного поворота вершин и коэффициенты масштабирования многоугольников определены внутри круглых скобок, следующих за `!P...` Помимо этого, использование символа `~` в командах `\xunderv` и `\vloop` дает возможность точно контролировать положение и размер блоков пересечений и соединений. Лучи звезды можно делать короче или длиннее путем коррекции коэффициента масштабирования внешнего многоугольника (для данного рисунка использовалось значение 2.2).

Применения в химии, физике и технике

Понятно, что из-за своего непревзойденного качества набора математических текстов TEX намного шире используется учеными-естественниками, чем другими группами людей. Поэтому неудивительно, что научное и инженерное сообщество разработало множество пакетов для набора диаграмм и схем, необходимых для различных дисциплин. В этой главе будут рассмотрены некоторые из этих средств. Следует отметить, что для многих задач пакеты общего назначения, такие, как PSTricks (гл. 4) или METAPOST (гл. 3), или более специализированные пакеты типа XY-pic (гл. 5), предоставляют все необходимые функциональные возможности. Рассматриваемые в настоящей главе пакеты предлагают решение проблем, возникающих в некоторых еще более специфичных областях.

Мы начнем с краткого обсуждения типографских правил набора научных текстов. В следующих двух разделах будут описаны два совершенно различных пакета для рисования химических структур — XYMTEX и ppchtex, а в разд. 6.4 мы разберем, как конструировать фейнмановские диаграммы, которые используются физиками. В последних трех разделах мы обратимся к электронике и опишем специальные пакеты для рисования временны́х диаграмм и электронных схем.

6.1 Типографские правила набора научных текстов

В научных текстах типографское представление символа часто несет в себе семантическое значение и поэтому авторы, представляющие документы для издания в форме camera-ready, должны знать многие типографские правила. В то время как общие схемы кодирования (как LATEX или SGML) облегчают получение приемлемого расположения текста, тем не менее остается ряд ловушек и даже большие мастера TEX'а, такие, как Дональд Кнут (Knuth, 1989), вынуждены признать поражение. Поэтому мы кратко изложим наиболее важные правила подготовки научных текстов (см. также International Union of Pure and Applied Physics (1987); Knappen (1992)).

Наиболее важным во всех обстоятельствах правилом является *согласованность*: данный символ должен всегда выглядеть одним и тем же способом, независимо от того, появляется он в формуле или в тексте, на основной строке или как верхний или нижний индекс. Отсюда следует важный для пользователей LaTeX'а совет: всегда набирайте символ либо в математическом, либо текстовом режиме, но никогда не смешивайте их вместе, даже если результат выглядит одинаковым. Основанием этому является то, что в системах типа LaTeX после смены файла класса или добавления нового пакета эти символы могут начать выглядеть совершенно по-разному. Например, при использовании PostScript'овских шрифтов цифры в тексте берутся из шрифта PostScript, которым набирается текст, и могут выглядеть совершенно иначе, чем цифры в формулах, которые набираются математическими шрифтами. Следовательно, числа, которые указывают на результаты или являются частью формул, всегда должны набираться в математическом режиме, т. е. окружаться $.

В научных текстах многие символы традиционно набираются буквами в *прямом начертании*[1] и не могут быть правильно интерпретированы в другом случае. Ниже представлены наиболее важные из этих случаев[2]:

- *единицы измерения*, например g, cm, s, keV. Заметим, что физические *постоянные* обычно набираются курсивом, так что константы с единицами измерений набираются смешанным образом, например GeV/c (где c — скорость света, константа);
- *химические элементы*, например Ne, O, Cu, и *имена элементарных частиц*, например p, K, q, H. Для набора типографски корректных текстов в помощь наборщику разработаны пакеты, которые содержат команды для представления подобных имен. В частности, химики могут использовать пакет chemsym (см. разд. 6.1.1), тогда как для физики высоких энергий предложена схема PEN (Particle Entity Notation) (Goossens and van Herwijnen, 1992);
- стандартные математические функции (sin, det, cos, $\Re$, $\Im$ и т. д.), для набора которых должны использоваться встроенные функции LaTeX'а;
- цифры;
- имена волн и состояний (p-волна) и ковариантных связей (A для аксиального вектора, V для полярного вектора); имена монополей (E для электрического, M для магнитного);
- аббревиатуры, представляющие собой части слов (exp для экспериментального; min для минимального);
- буква «d» в интегралах (например, dp).

[1] Чтобы получить прямое начертание букв в математическом режиме LaTeX'а, используется команда `\mathrm`.

[2] Речь идет, разумеется, о правилах набора англоязычной научно-технической литературы. Относительно правил набора русскоязычной литературы см., например, П. Г. Гиленсон, «Справочник художественного и технического редактора». М., 1988. — *Прим. ред.*

Таблица 6.1. Важность способа представления символов в научных текстах.

прямой шрифт		*курсивный шрифт*	
A	ампер (единица измерений)	A	атомный номер (переменная)
e	электрон (имя частицы)	e	заряд электрона (константа)
g	глюон (имя частицы)	g	гравитационная постоянная (константа)
l	литр (единица объема)	l	длина (переменная)
m	метр (единица длины)	m	масса (переменная)
p	протон (имя частицы)	p	момент (переменная)
q	кварк (имя частицы)	q	электрический заряд (переменная)
s	секунда (единица времени)	s	скорость (переменная)
t	тонна (единица веса)	t	время (переменная)
V	вольт (электрическая единица)	V	объем (переменная)
Z	Z-бозон (имя частицы)	Z	атомный заряд (переменная)

Следование этим типографским соглашениям помогает читателю с первого взгляда понимать значение символа. В табл. 6.1 приведено несколько примеров, в которых значение символа зависит от его типографского представления.

6.1.1 Набор химических символов

Химические тексты часто включают названия элементов. Из обсуждения в предыдущем разделе мы знаем, что эти названия должны набираться прямым шрифтом. Матс Дальгрен разработал пакет chemsym, позволяющий гарантировать типографскую корректность при наборе названий элементов. В пакете определены команды для набора символов 109 химических элементов с использованием имен, совпадающих с их названиями в Периодической системе. Несколько дополнительных команд позволяют вводить группы (типа `\OH`, `\COOH` и `\CH`), а команда `\kemtkn` позволяет определить другие привычные вам химические символы. Поскольку в LaTeX'е уже есть команды, имена которых совпадают с названиями некоторых химических элементов, пакет chemsym переименовывает эти команды следующим образом:[1]

`\H` → `\h`	Венгерский умляут ő,
`\O` → `\OO`	Датский и норвежский Ø,
`\P` → `\PP`	Знак абзаца ¶,
`\S` → `\SS`	Знак параграфа §,
`\Re` → `\re`	Действительная часть $\Re$ (в математическом режиме),
`\Pr` → `\pr`	Функция вероятности Pr (в математическом режиме).

Кроме того, окружение `Sb` в пакете amstex переименовывается в `SB`.

[1] Следует отметить, что переименование команд, которые зависят от кодировки LaTeX'а, т. е. команд, которые могут менять внутреннее значение при изменении кодировки шрифта, такие, как `\H` или `\O`), никогда не может быть полностью успешным. В какой-то степени эти команды напоминают их старые функции; например, они могут записывать имя во внешние файлы, что может привести к сюрпризам при печати оглавления книги и т. п.

Для упрощения ввода химических формул символы ^ и _, используемые в LaTeX'е для ввода верхних и нижних индексов, переопределены так, чтобы быть доступными также и вне математического режима. Так, строка в тексте:

```
2\H_2 и \O_2 дает 2\H_2\O в реакции со взрывом.
```

дает:

$2H_2$ и O_2 дает $2H_2O$ в реакции со взрывом.

Заметим, что некоторые пакеты, в частности такие, как longtable, требуют исходного значения символа ^, который используется для работы с управляющими кодами. В этом случае переопределение приводит к хаосу и должно быть отключено путем добавления опции `collision` при загрузке chemsym с помощью команды `\usepackage`.

Сложным и интересным примером использования пакета chemsym может быть файл `pertab.tex`, созданный Матсом Дальгреном. Он набирает полную Периодическую таблицу элементов (см. рис. 6.1) и является частью дистрибутива chemsym. Например, период, начинающийся с калия (K), вводится в окружении `tabular` следующим образом:

```
_{19}       & _{20}       & _{21}        & _{22}      &
_{23}       & _{24}       & _{25}        & _{26}      &
_{27}       & _{28}       & _{29}        & _{30}      &
_{31}       & _{32}       & _{33}        & _{34}      &
_{35}       & _{36}                                   \\
\K          & \Ca         & \Sc          & \Ti        &
\V          & \Cr         & \Mn          & \Fe        &
\Co         & \Ni         & \Cu          & \Zn        &
\Ga         & \Ge         & \As          & \Se        &
\Br         & \Kr                                     \\
^{39.0983}  & ^{40.078}   & ^{44.955910} & ^{47.867}  &
^{50.9415}  & ^{51.9961}  & ^{54.93805}  & ^{55.845}  &
^{58.93320} & ^{58.6934}  & ^{63.546}    & ^{65.39}   &
^{69.723}   & ^{72.61}    & ^{74.92159}  & ^{78.96}   &
^{79.904}   & ^{83.80}                                \\\hline
```

6.2 Система XLMTEX

Химические диаграммы достаточно сложны и в (LA)TeX'е есть несколько утилит для их набора.[1] Появление таких утилит частично связано с тем, что хотя окружение `picture` в LaTeX'е вполне адекватно для рисования простых рисунков, для набора химических формул требуется более структурированная система макрокоманд.

Первыми попытками создания специальных химических пакетов было создание пакета ChemTeX (авторы — Росвита Хас и Кевин Кейн (1987)) и макро

[1] См. интересный обзор Norris and Oakley (1990), касающийся электронных публикаций и обработки химических текстов.

Periodic Table of the Elements

with relative atomic masses 1993 according to IUPAC

1 (I)	2 (II)	3	4	5	6	7	8	9	10	11	12	13 (III)	14 (IV)	15 (V)	16 (VI)	17 (VII)	18 (VIII)
1 H 1.00794																	2 He 4.002602
3 Li 6.941	4 Be 9.012182					Atomic number Symbol Relative atomic mass*						5 B 10.811	6 C 12.011	7 N 14.00674	8 O 15.9994	9 F 18.9984032	10 Ne 20.1797
11 Na 22.989768	12 Mg 24.3050											13 Al 26.981539	14 Si 28.0855	15 P 30.973762	16 S 32.066	17 Cl 35.4527	18 Ar 39.948
19 K 39.0983	20 Ca 40.078	21 Sc 44.955910	22 Ti 47.867	23 V 50.9415	24 Cr 51.9961	25 Mn 54.93805	26 Fe 55.845	27 Co 58.93320	28 Ni 58.6934	29 Cu 63.546	30 Zn 65.39	31 Ga 69.723	32 Ge 72.61	33 As 74.92159	34 Se 78.96	35 Br 79.904	36 Kr 83.80
37 Rb 85.4678	38 Sr 87.62	39 Y 88.90585	40 Zr 91.224	41 Nb 92.90638	42 Mo 95.94	43 Tc (98)	44 Ru 101.07	45 Rh 102.90550	46 Pd 106.42	47 Ag 107.8682	48 Cd 112.411	49 In 114.818	50 Sn 118.710	51 Sb 121.760	52 Te 127.60	53 I 126.90447	54 Xe 131.29
55 Cs 132.90543	56 Ba 137.327	La–Lu	72 Hf 178.49	73 Ta 180.9479	74 W 183.84	75 Re 186.207	76 Os 190.23	77 Ir 192.217	78 Pt 195.08	79 Au 196.96654	80 Hg 200.59	81 Tl 204.3833	82 Pb 207.2	83 Bi 208.98037	84 Po (209)	85 At (210)	86 Rn (222)
87 Fr (223)	88 Ra (226)	Ac–Lr	104 Db (261)	105 Jl (262)	106 Rf (263)	107 Bh (262)	108 Hn (265)	109 Mt (266)	**								

57 La 138.9055	58 Ce 140.115	59 Pr 140.90765	60 Nd 144.24	61 Pm (145)	62 Sm 150.36	63 Eu 151.965	64 Gd 157.25	65 Tb 158.92534	66 Dy 162.50	67 Ho 164.93032	68 Er 167.26	69 Tm 168.93421	70 Yb 173.04	71 Lu 174.967
89 Ac (227)	90 Th (232.0381)	91 Pa (231.03588)	92 U (238.0289)	93 Np (237)	94 Pu (239)	95 Am (243)	96 Cm (247)	97 Bk (247)	98 Cf (251)	99 Es (252)	100 Fm (257)	101 Md (258)	102 No (259)	103 Lr (262)

* Relative atomic mass based on $A_r(^{12}C) \equiv 12$ (after IUPAC "Atomic Weights of the Elements 1993", *Pure and Applied Chemistry,* **1994,** *66*(12), 2423-2444). For elements which lack stable isotope(s) the mass number for the most stable isotope is given in parentheses, or for Th, Pa and U the relative atomic mass given by IUPAC for the isotopic mixture present on Earth.

** Chemical symbols for elements 104 – 109 according to IUPAC "Names and Symbols of Transfermium Elements (IUPAC Recommendations 1994)", *Pure and Applied Chemistry,* **1994,** *66*(12), 2419-2421.

Рис. 6.1. Периодическая таблица элементов, набранная в LaTeX'е с помощью пакета chemsym.

\structure для plain TEX'а (Ramek, 1990). Хотя эти попытки были шагом вперед, в них оказывалось трудным расположить достаточно большое число составляющих групп систематическим и прозрачным способом. Недавно Синсаку Фудзита (Fujita, 1993, 1995) разработал систему XyMTEX, которая состоит из набора LATEX'овских пакетов для рисования широкого круга химических структурных формул.

В командах XyMTEX'а используется систематический подход указания аргументов для замещающих групп, их положений, двойных связей и рисунков связей, а с помощью дополнительного аргумента можно указать гетероатомы в вершинах гетероциклов. XyMTEX использует только окружение picture из LATEX'а и лишь в некоторых случаях пакет-расширение epic, так что он может работать в разных реализациях TEX'а. XyMTEX состоит из набора пакетов, каждый из которых специализируется на рисовании определенного типа химических структур. Это:

chemstr общие команды рисования атомов и связей;

aliphat алифатические соединения;

carom карбоциклические соединения (в вертикальной и горизонтальной ориентации);

ccycle бициклические соединения и т. п.;

lowcycle карбоциклы с пятью или меньшим числом атомов в цикле;

hcycle производные пиранозы и фуранозы;

hetarom гетероциклические соединения (в вертикальной ориентации);

hetaromh гетероциклические соединения (в горизонтальной ориентации);

locant дополнительный пакет для вывода номеров групп-заместителей и букв-обозначений связей, используемый для подготовки руководства по XyMTEX'у;

xymtex стилевой файл документа LATEX 2.09 (необходим только для подготовки руководства по XyMTEX'у).

Пакеты могут загружаться индивидуально с помощью команды LATEX'а \usepackage, однако пакет chemstr загружается автоматически при вызове любого из других пакетов.

6.2.1 Общие соглашения

Имена команд XyMTEX'а основаны на стандартной номенклатуре, используемой для органических соединений (International Union of Pure and Applied Chemistry, 1979). Неизменная часть структуры, т. е. базовый скелет, который содержит фиксированные связи и атомы, выводится автоматически, в то время как изменяемая часть (замещающие группы, дополнительные связи и атомы) указываются с помощью набора аргументов.

Эти команды могут быть подразделены на команды *общего* и *специального* назначения. Специальные команды предлагают удобные сокращенные обозначения для рисования небольшого набора структур определенной категории,

тогда как общие команды поддерживают более широкий круг объектов, но требуют дополнительной спецификации. Это различие отражается и в самих именах команд. Например, специальные команды для рисования определенного типа N-гетероциклов будут называться `\isoindolev` или `\purinev` (в которые атомы N будут вставляться автоматически), в то время как имя соответствующей общей команды будет `\nonaheterov` (которая описывает геометрическую структуру базового скелета без каких-либо атомов в предопределенных положениях).

Синтаксис специальных команд

Для карбоциклов и гетероциклов мы имеем абстрактный синтаксис

`\Com[`*opt*`]{`*subslist*`}`

где `\Com` — имя команды, отвечающей набираемому соединению. Оно обычно имеет суффикс `v` или `h`, указывающий на вертикальную или горизонтальную ориентацию структуры. Если возможны альтернативные ориентации, имя команды может иметь в качестве суффикса еще и букву `i`.

Предопределенный для структуры рисунок связей можно модифицировать с помощью аргумента *opt*, который состоит из комбинаций одной или двух букв, обозначающих определенный рисунок связей. Допустимые значения аргументов и их точный смысл зависят от команды; их детальное описание приведено в руководстве (Fujita, 1993).

Аргументы *subslist* перечисляют замещающие атомы или группы с их связями, которые (если в структуре есть более одного замещения) отделяются друг от друга точками с запятой. Связь кодируется числом, указывающим положение атома в структуре, к которому эта связь относится; за этим числом при необходимости может следовать модификатор (одна или несколько букв), определяющий тип связи, который будет нарисован. Атомы замещения отделяются от номеров связей двумя знаками равенства. Например, `1D==O;3==OCH$_3$;...` означает, что атом в позиции 1 связан с атомом кислорода (`O`) через двойную связь (`D`), а атом в положении 3 связан с метоксигруппой через одинарную связь и т. д. Положения атомов нумеруются последовательно, обычно по часовой стрелке, начиная от вершины для структур с вертикальной ориентацией или слева для структур с горизонтальной ориентацией. Возможными модификаторами связи для атома в положении n являются:

- n`S` направленная за пределы цикла одинарная связь на атоме n (эквивалентно простому указанию n);
- n`D` направленная за пределы цикла двойная связь;
- n`A` одинарная alpha-связь, изображаемая точечной линией;
- n`B` одинарная beta-связь, изображаемая жирной линией.

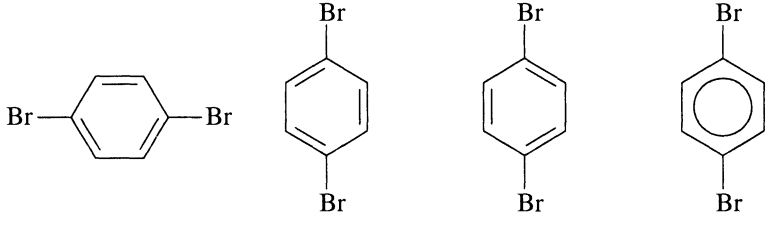
Работа модификатора n`A` основана на использовании команды `\dottedline`, определенной в пакетах epic и eepic. Если ни один из этих пакетов не загру-

жен, то связь будет нарисована одинарной линией обычной ширины, т. е. будет выглядеть так же, как если бы было указано просто n.

Если число n больше девяти, то оно должно быть заключено в двойные фигурные скобки, иначе команды не смогут интерпретировать их аргумент *subslist* правильно. Так, следует писать `1D==O`, чтобы добавить атом O с помощью двойной связи к первому атому в структуре, и `{{13}}D==O`, чтобы добавить его к тринадцатому атому. К сожалению, при неправильном определении не выдается никакого сообщения об ошибке, и единственным указанием на проблему является неправильно нарисованная структура.

В качестве первого примера рассмотрим соединение 1,4-дибромбензол (также называемый пара-дибромбензолом или n-дибромбензолом). Мы можем нарисовать несколько представлений молекулы, используя суффиксы `h` и `v` в имени команды и указывая разные форматы рисунка связей в необязательном аргументе, например:

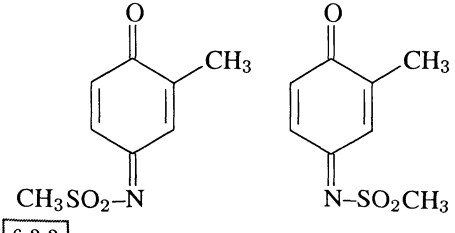
```
\usepackage {carom}
\bzdrh{1==Br;4==Br}     \bzdrv{1==Br;4==Br}
\bzdrv[l]{1==Br;4==Br} \bzdrv[c]{1==Br;4==Br}
```



6-2-1

Необязательный параметр `[l]` указывает на «лево-ориентированный» набор двойных связей (по умолчанию используется «правая ориентация» `[r]`), а параметр `[c]` указывает, что надо изобразить ароматическое кольцо. Другими допустимыми значениями этого необязательного аргумента для `\bzdrh` и `\bzdrv` являются `[pa]` .. `[pc]` для трех n-хинонных вариантов (два примера будут показаны в следующем примере) и `[oa]` .. `[of]` для шести o-хинонных вариантов.

Вместо того чтобы предоставить команде самой решать, как сориентировать заместитель, вы можете указать эту ориентацию явным образом с помощью команд `\rmoiety` и `\lmoiety`, как это сделано в следующем примере с левой и правой ориентацией метансульфонимидных групп:



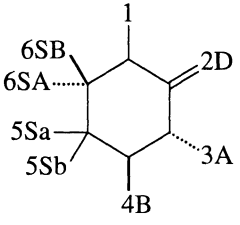

```
\usepackage {carom}
\bzdrv[pa]{1D==O;4D==%
 \lmoiety{CH$_{3}$SO$_{2}$--N};%
 2==CH$_{3}$}
\bzdrv[pa]{1D==O;4D==%
 \rmoiety{N--SO$_{2}$CH$_{3}$};%
 2==CH$_{3}$}
```

6-2-2

В зависимости от базовой структуры, атом скелета может иметь две внешние связи. В этом случае модификаторами связи в положении n могут быть:

nSA одинарная alpha-связь, изображаемая точечной линией;
nSB одинарная beta-связь, изображаемая жирной линией;
nSa не выделяемая одинарная alpha-связь;
nSb не выделяемая одинарная beta-связь.

Для создания точечной линии для alpha-связи здесь также необходимы пакеты epic или eepic. В абстрактном примере, показанном ниже, в одной структуре использованы все допустимые модификаторы; реальные примеры будут даны ниже.




```
\usepackage {carom,epic}
\cyclohexanev{1==1;%
   2D==2D;%
   3A==3A;%
   4B==4B;%
   5Sa==5Sa;5Sb==5Sb;%
   6SA==6SA;6SB==6SB}
```

6-2-3

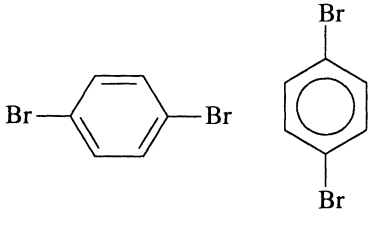
Синтаксис общих команд

Команды общего типа, такие, как `\cyclohexanev` в предыдущем примере, подобны специальным командам, но могут иметь произвольный набор гетероатомов в скелете и тем самым перекрывают более широкий набор структур соединений. Для карбоциклов общий синтаксис команды таков:

`\Com[`*bondlist*`]{`*subslist*`}`

где вместо необязательного аргумента, указывавшего предопределенную картину связей, здесь используется необязательный аргумент *bondlist*, позволяющий указывать положение двойных связей индивидуально. Возможные двойные связи в структуре обозначаются буквами a, b, c и т. д. и для того, чтобы на рисунке появилась двойная связь, необходимо указать соответствующие буквы в аргументе *bondlist*. Если структура достаточно сложна и имеет более чем один цикл (как, например, в случае производных пергидроантрацена), то ароматические циклы обозначаются заглавными буквами A, B и т. д.

Все специальные команды в пакетах реализованы как вызовы соответствующих общих команд с передачей им определенных аргументов. Например, рассмотренный выше пример с вызовом `\bzdrh` также может быть набран с использованием общей команды `\cyclohexaneh`:



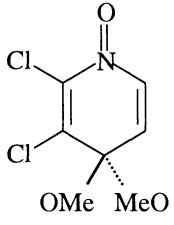

```
\usepackage {carom}
\cyclohexaneh
   [bdf]{1==Br;4==Br}
\cyclohexanev
   [A]{1==Br;4==Br}
```

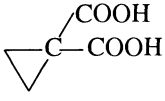
6-2-4

Синтаксис для гетероциклов оказывается чуть более сложным, поскольку нам нужно предусмотреть способ, чтобы указывать положение индивидуальных атомов в базовом скелете:

\Com[*bondlist*]{*atomlist*}{*subslist*}

Как и выше, двойные связи в скелете указываются с помощью аргумента *bondlist*, а заместители (включая их связи) — с помощью аргумента *subslist*. Эти команды различает только дополнительный обязательный аргумент *atomlist*, в котором при необходимости можно указать положение любого индивидуального атома в базовом скелете. Синтаксис аргумента *atomlist* такой же, как и в *subslist*: спецификации атомов разделяются точками с запятой, а позиция атома указывается числом, за которым следует два знака равенства и строка, обозначающая атом(ы) замещения:



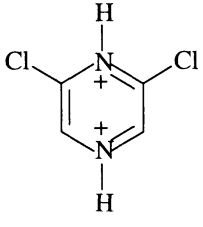



```
\usepackage {hetarom,epic}
\sixheterov[eb]{1==N}
    {1D==O;
     4SA==MeO;4SB==OMe;
     5==Cl;
     6==Cl}
\threehetero[H]{2==C}
 {2Sa==COOH;2Sb==COOH}
```

6-2-5

Заряды на атомах

В некоторых случаях заряды на индивидуальных атомах базового скелета могут быть переданы через аргументы *opt* или *bondlist*, с указанием номера атома, за которым следует знак плюс, заключенный в фигурные скобки:



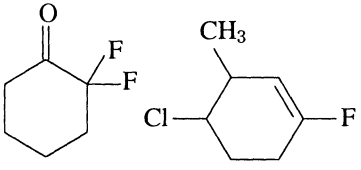

```
\usepackage {hetarom,epic}
\pyrazinev[1{1+}{4+}]
   {1==H;4==H;2==Cl;6==Cl}
```

6-2-6

Теперь, разобрав принципы устройства системы XyMTEX, мы готовы более подробно рассмотреть некоторые из его пакетов для рисования химических формул.

6.2.2 Пакет carom

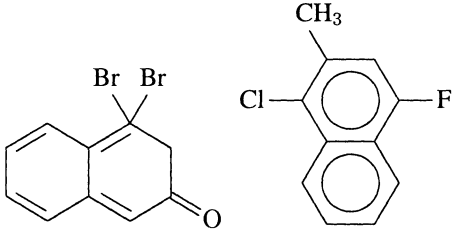
Пакет carom, входящий в XeTeX, позволяет рисовать множество различных карбоциклов. Мы уже рассмотрели несколько примеров команд \bzdrv и \bzdrh для рисования производных бензола. Для рисования производных циклогексана используются команды \cyclohexanev и \cyclohexaneh. С помощью описанного выше параметра *bondlist* вы можете определить двойные связи в произвольном месте, например:




```
\usepackage {epic,carom}
\cyclohexanev
  {1D==O;2Sa==F;2Sb==F}
\cyclohexaneh[c]
  {1==Cl;4==F;2==CH$_{3}$}
```

6-2-7

Конденсированные циклы, такие, как у нафталина и нафтохинонов, рисуются с помощью набора команд \naphdrv. Аргумент *opt* для рисунка связей поддерживает все варианты связей в *o*-хинонах и ароматические кольца:

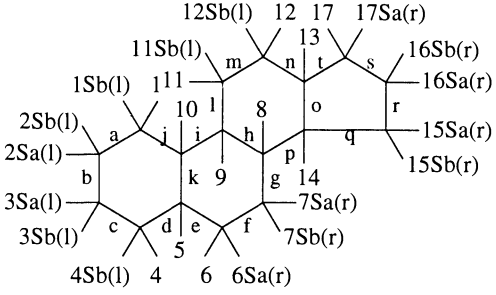



```
\usepackage {epic,carom}
\naphdrv[ob]
  {1Sb==Br;1Sa==Br;3D==O}
\naphdrh[A]
  {1==Cl;4==F;2==CH$_{3}$}
```

6-2-8

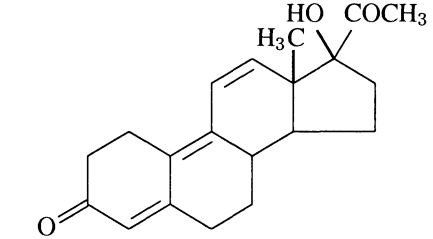
Кроме указанных, пакет включает команды для рисования других сложных структур: производных тетралина (команды \tetralinev, \tetralineh) и декалина (\decalinev, \decalineh), трициклических карбоциклов типа производных антрацена (\anthracenev, \hanthracenv) и фенантрена (\phenanthrenev, \hphenanthrenev), а также производных стероидов с боковыми цепями и без них (\steroid, \steroidchain). Все эти команды имеют такой же общий синтаксис, какой был описан в предыдущем разделе, и поэтому его легко использовать, хотя запоминание номеров положений атомов и буквенных обозначений связей может потребовать определенного времени. Эти обозначения хорошо документированы как в самих файлах пакета, так и

в подробном online-руководстве (более 120 страниц диаграмм и примеров), где эти обозначения изображены на таких диаграммах:



6-2-9

С помощью диаграмм подобного вида и таблиц, в которых перечислены допустимые значения аргументов *opt* и *bondlist*, не так трудно набрать формулу такого вида:



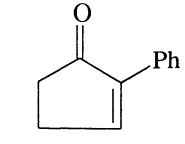

```
\usepackage {carom}
\steroid[dim]
 {3D==0;%
 {{13}B}==\lmoiety{H$_3$C};%
 {{17}SB}==HO;%
 {{17}SA}==COCH$_3$}
```

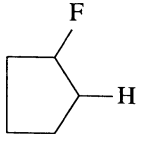
6-2-10

Необязательный аргумент `[dim]` в этом примере указывает положения трех двойных связей, а команда `\lmoiety` соединяет углерод метила с соответствующим (13-м) атомом в атомном скелете.

6.2.3 Пакет lowcycle

Циклы с небольшим числом атомов (менее шести атомов углерода в одном из циклов) рисуются с помощью пакета lowcycle. Пятичленные циклы рисуются набором команд `\cyclopentanev` (всего имеется четыре типа суффиксов). Примеры таких рисунков показаны ниже.






```
\usepackage {lowcycle}
\cyclopentanevi[b]{1D==0;%
  2==Ph}
\cyclopentaneh[H]{1==H;2==F}
```

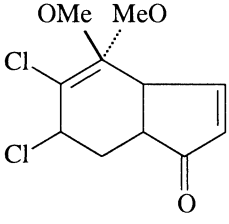
6-2-11

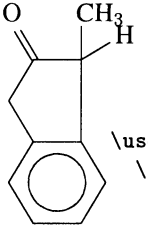
Кроме помещения заряда на каждом из атомов, в этих структурах можно также указать заряд в центре цикла, который вводится с помощью {0+} или {0$-$}:

```
\usepackage {epic,lowcycle}
\cyclopentanev[A{0{$-$}}]{}
```

6-2-12

Кроме того, в этом пакете имеется возможность рисовать и производные индена (конденсированные шести- и пятичленные кольца) с помощью набора команд \indanev (здесь также возможны горизонтальная, вертикальная и обращенная формы):

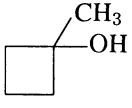


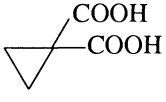



```
\usepackage {epic,lowcycle}
 \indanev[eb]
  {1D==O;4SA==MeO;%
   4SB==OMe;5==Cl;6==Cl}
 \indanehi[A]{2D==O;%
   3Sa==CH$_3$;3Sb==H}
```

6-2-13

Наконец, рисование четырех- и трехатомных углеродных циклов осуществляется командами \cyclobutane и \cyclopropane. Неудивительно, что благодаря их простой геометрической форме они не нуждаются в суффиксах для обозначения вариантов:






```
\usepackage {lowcycle}
\cyclobutane
  {3Sa==OH;3Sb==CH$_3$}
\cyclopropane{2Sa==COOH;%
   2Sb==COOH}
```

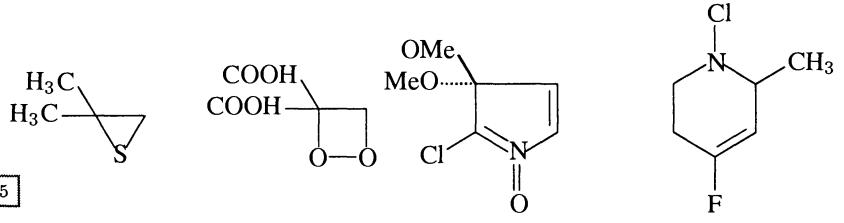
6-2-14

6.2.4 Пакеты hetarom и hetaromh

Гетероциклические соединения, т. е. такие соединения, в которых один или несколько атомов углерода в цикле замещено другими атомами, рисуются с помощью пакетов hetarom (команды рисования в вертикальной ориентации) или hetaromh (соответственно в горизонтальной ориентации).

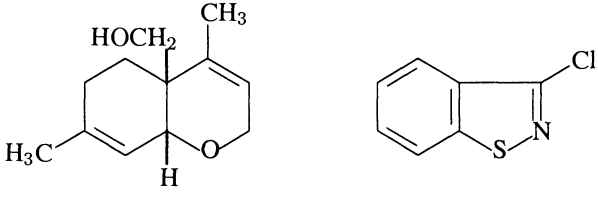
Общие команды для гетероциклов с длиной цикла от 3 до 6 имеют общепонятные имена типа \sixheterov. Вот несколько примеров использования таких команд:

```
\usepackage {epic,hetarom}
\threehetero      {1==S}     {3Sa==H$_3$C;3Sb==H$_3$C}
\fourhetero       {1==O;2==O}{4Sa==COOH;4Sb==COOH}
\fiveheterov[eb]  {1==N}     {1D==O;4SA==MeO;4SB==OMe;5==Cl;6==Cl}
\sixheterov [c]   {1==N}     {1==Cl;4==F;2==CH$_3$}
```



6-2-15

Кроме того, конденсированные шестичленные циклы и конденсированные шести- и пятичленные циклы поддерживаются командами общего типа. Все они, как и две формулы в примере выше, имеют горизонтальную и (там, где это возможно) инверсную формы.



6-2-16

```
\usepackage {eepic,hetarom}
\decaheterov[af]{4==O}
  {1==CH$_3$;6==H$_3$C;%
  9A==H;{{10}A}==%
  \lmoiety{HOCH$_2$}}
\nonaheterov[bjge]
  {1==S;2==N}{3==Cl}
```

Ясно, что с помощью такого большого набора общих команд достаточно просто определить еще более богатый набор специальных команд. И действительно, эти два пакета содержат более 80 таких команд (если считать все варианты суффиксов для рисования структур различными способами). Они включают производные пиридина и другие гетероциклические структуры с шестичленными циклами; для пятичленных структур — производные пиррола и другие структуры. Как детализация общих команд с конденсированным циклом \nonaheterov и \decaheterov, существуют команды для рисования N-гетероциклов и N,O-гетероциклов типа индолов и изобензофуранов. В табл. 6.2 приведен список этих команд.

Таблица 6.2. Специальные команды пакета hetarom. Соответствующие горизонтальные варианты определены в пакете hetaromh.

Шестичленные гетероциклы

```
\pyrazinev \pyridazinevi \pyridazinev \pyridinevi \pyridinev
\pyrimidinevi \pyrimidinev \triazinevi \triazinev
```

Пятичленные гетероциклы

```
\imidazolevi \imidazolev \isoxazolevi \isoxazolev \oxazolevi
\oxazolev \pyrazolevi \pyrazolev \pyrrolevi \pyrrolev
```

6+6 конденсированные гетероциклы

```
\cinnolinevi \cinnolinev \isoquinolinevi \isoquinolinev \pteridinevi
\pteridinev \quinazolinevi \quinazolinev \quinolinevi \quinoxalinev
```

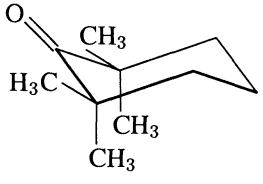
6+5 конденсированные гетероциклы

```
\benzofuranevi \benzofuranev \benzoxazolevi \benzoxazolev \indolevi
\indolev \indolizinevi \indolizinev \isobenzofuranevi
\isobenzofuranev \isoindolevi \isoindolev \purinevi \purinev
```

6.2.5 Пакеты ccycle и hcycle

Рисование формул стереохимических соединений упрощается командами для построения конформационных диаграмм для циклогексана, бицикло[2.2.1]гептана и производных адамантана, которые загружаются с помощью пакета ccycle.

Чтобы указать одинарную связь в экваториальной плоскости у n-го атома, в команде `\chair` используется модификатор связи nSe, например:

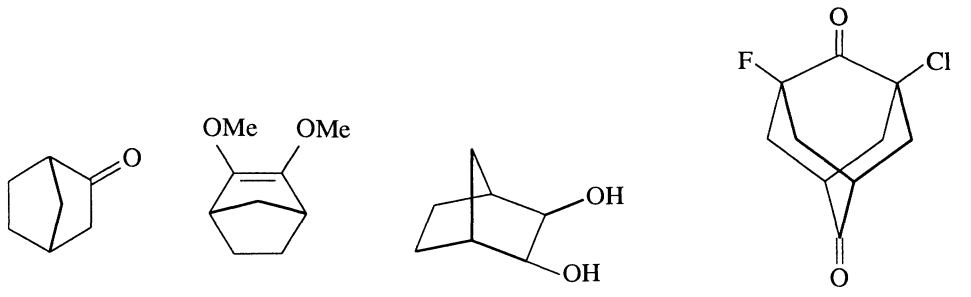



```
\usepackage {ccycle}
\chair{1D==O;%
  2Se==H$_3$C;2Sa==CH$_3$;%
  6Se==CH$_3$;6Sa==CH$_3$}
```

6-2-17

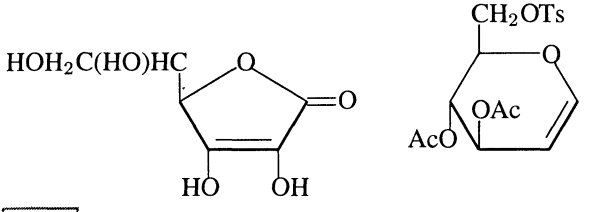
Другие команды имеют стандартный синтаксис за исключением того, что ориентация заместителей не может быть задана явно:

```
\usepackage {ccycle}
\bicychepv{2D==O}          \bicycheph[b]{2==OMe;3==OMe}
\bornane{3B==OH;2A==OH}    \adamantane{2D==O;6D==O;1==F;3==Cl}
```



6-2-18

Команды для рисования производных фуранозы и пиранозы определены в пакете hcycle. Ниже приведены примеры обеих структур:

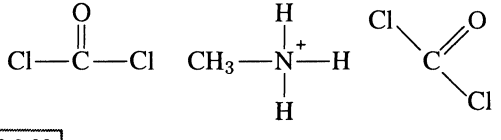


```
\usepackage {hcycle}
\furanose[b]{1D==O;2Sa==OH;%
  3Sa==\lmoiety{HO};%
  4Sb==HOH$_2$C(HO)HC}
\qquad
\pyranose[a]{3Sb==OAc;%
  4Sa==AcO;5Sb==CH$_{2}$OTs}
```

6-2-19

6.2.6 Пакет aliphat

Для рисования алифатических соединений вам понадобится пакет aliphat. Поскольку структурная единица с четырьмя или тремя связями содержит один атом, команды для их рисования не имеют аргумента *bondlist*. Однако они могут иметь необязательный аргумент, в котором может быть указан заряд этого атома (см. пример ниже). Другой особенностью этой группы команд является возможность задать символ химического элемента для атома в центральном положении (например, O==C).

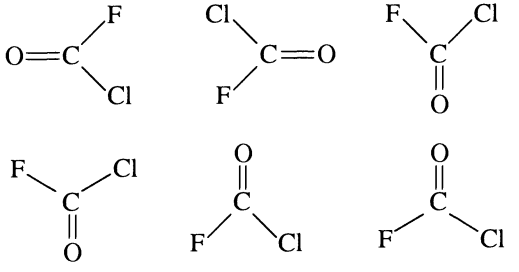


```
\usepackage {aliphat}
\tetrahedral{O==C;1D==O;2==Cl;%
  4==Cl}
\qquad
\tetrahedral[{O+}]{O==N;%
  1==H;2==CH$_3$;3==H;4==H}
\square{O==C;1D==O;2==Cl;4==Cl}
```

6-2-20

Для рисования ячеек с тремя связями существует шесть разных команд. Команды, начинающиеся с заглавной буквы, рисуют диагональные связи под

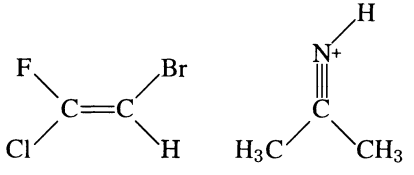
углом 120° между собой, в то время как остальные — под углом 90° между собой:



```
\usepackage {aliphat}
\centering
\rtrigonal{0==C;1D==O;2==Cl;3==F}
\qquad
\ltrigonal{0==C;1D==O;2==Cl;3==F}
\utrigonal{0==C;1D==O;2==Cl;3==F}
\\
\Utrigonal{0==C;1D==O;2==Cl;3==F}
\dtrigonal{0==C;1D==O;2==Cl;3==F}
\qquad
\Dtrigonal{0==C;1D==O;2==Cl;3==F}
```

6-2-21

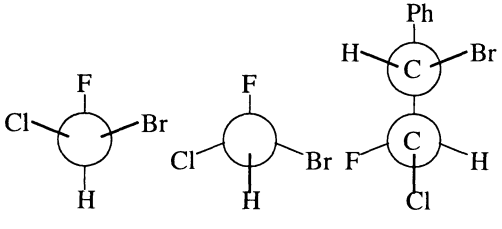
Для рисования производных этилена служит команда `\ethylene` (обратите внимание на отсутствие суффикса) и ее «вертикальные» формы `\ethylenev` и `\Ethylenev` (для последней характерно расположение диагональных связей под углом 120° между собой). Внутренние двойные и тройные связи указываются с помощью необязательных аргументов `d` (который принят по умолчанию) и `t`:



```
\usepackage {aliphat}
\ethylene{1==C;2==C}
    {1==F;2==Cl;3==H;4==Br}
\qquad
\ethylenev[t{2+}]{1==C;2==N}
    {1==H$_3$C;2==CH$_3$;3==H}
```

6-2-22

Наконец, в этом пакете с помощью команд `\tetrastereo` и `\dtetrastereo` поддерживается рисование пространственных конфигураций тетраэдрического атома углерода в различных проекциях, а с помощью команды `\ethanestereo` — и конфигураций этана. При этом имена центральных атомов могут быть заданы явным образом:



```
\usepackage {aliphat}
\setlength\unitlength{0.1pt}
\tetrastereo{1==F;%
   2==Cl;3==H;4==Br}
\dtetrastereo{1==F;%
   2==Cl;3==H;4==Br}
\ethanestereo{1==C;2==C}
  {1==F;2==Cl;3==H;4==Br;%
   5==Ph;6==H}
```

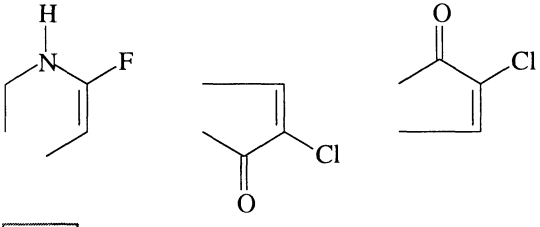
6-2-23

6.2.7 Комбинирование структур

Пакеты hetarom и hetaromh также поддерживают «строительные блоки» из шести- и пятичленных колец, которые могут быть пририсованы к другим циклическим структурам, чтобы получить новые структуры. Синтаксис этих команд похож на синтаксис команд общего вида для гетероциклов, но имеет один обязательный дополнительный аргумент *omit*:

`\Com[`*bondlist*`]{`*atomlist*`}{`*subslist*`}{`*omit*`}`

Аргумент *omit* указывает, которая из связей должна быть стерта из структуры. Например:




```
\usepackage {hetarom}
\sixunitv[b]{1==N}{1==H;2==F}{d}
\fiveunitv[b]{}{1D==O;2==Cl}{d}
\fiveunitvi[b]{}{1D==O;2==Cl}{d}
```

6-2-24

Соответствующие «горизонтальные» команды определены в пакете hetaromh.

Используя окружение `picture` (и технику, описанную в гл. 14 руководства Fujita (1993)) и комбинируя две или большее число структур, построенных из описанных фрагментов или с помощью команд, которые были рассмотрены в предыдущих разделах, можно построить более сложные структурные формулы. Другим способом конструирования сложных структур является использование одной команды XeLaTeX'а в качестве аргумента другой команды (Fujita, 1993, гл. 15).

6.2.8 Определение собственных структур

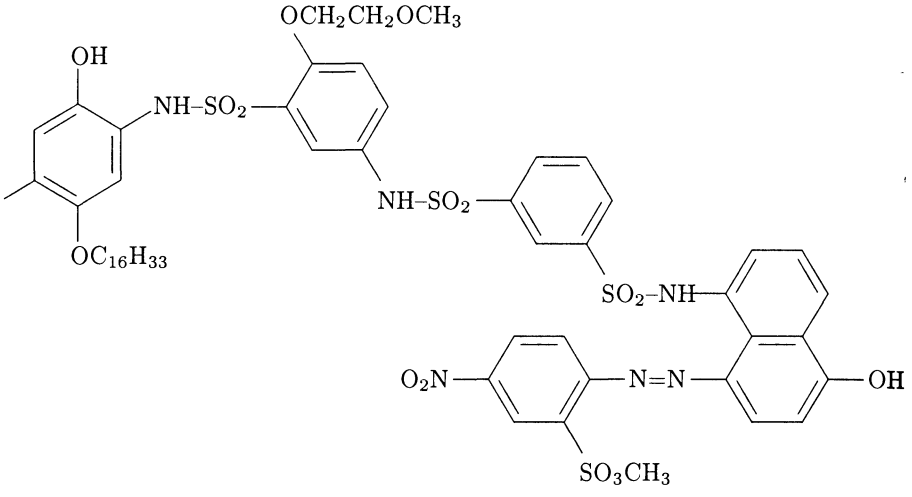
С помощью команд общего типа нетрудно определить новые специальные макрокоманды. Например, если бы команды `\pyridinev` не существовало, ее можно было бы легко добавить с помощью следующего определения в преамбуле вашего документа:

```
\newcommand{\pyridinev}[2][r]{\sixheterov[#1]{1==N}{#2}}
```

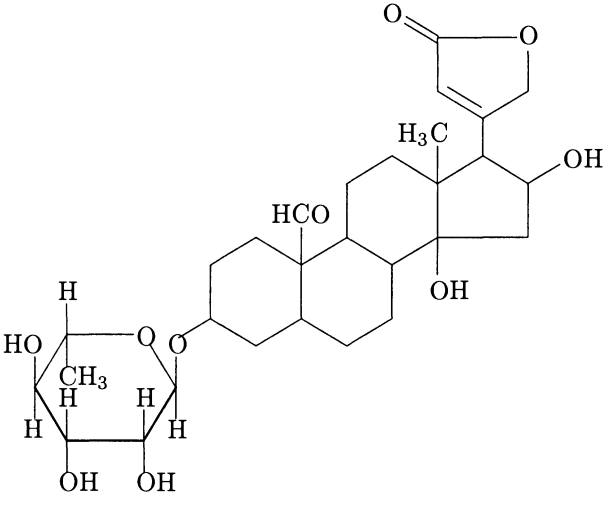
Обратите внимание на второй необязательный аргумент в `\newcommand`, который указывает, что команда `\pyridinev` сама должна иметь необязательный аргумент, по умолчанию принимающий значение `r`.

Мы завершим наш обзор пакета XeLaTeX, показав два примера сложных структур, которые приводятся без кода. Обе структуры подробно обсуждают-

руководстве (Fujita, 1993, с. 115–117). Первая из них представляет собой мулу проявителя голубого красителя для мгновенной цветной фотографии:



Вторым примером является формула адонитоксина, в котором два сложзаместителя помещены в стероидный скелет:



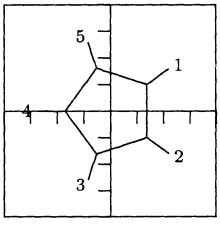
6.3 Пакет ppchtex

Совершенной иной подход к рисованию химических формул был недавно предложен Гансом Хагеном и А. Ф. Оттеном (1996), которые создали PPCHTEX — пакет для набора химических формул, работа которого основана на PICTEX (Wichura, 1987); интерфейс этого пакета с PSTricks находится в стадии разработки. В этом пакете химические структуры описываются скорее на языке их графического представления, чем на основе их химического строения. В отличие от XYMTEX'а, который базируется на LATEX'овском окружении `picture`, этот пакет может использоваться как с plain TEX'ом, так и с LATEX'ом.

Согласно авторам, пакет продолжает развиваться. Вместо того чтобы пытаться подробно объяснять синтаксис команд, мы просто приведем несколько примеров с целью дать вам почувствовать используемый подход. Подробности вы сможете найти в документации, которая прилагается к системе.

6.3.1 Структуры

Структурная химическая формула набирается с помощью четырех базовых команд, которые все использованы в следующем примере:

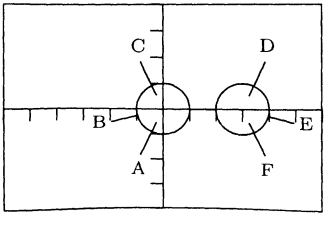



```
\usepackage {pictex,m-ch-en}
\setupchemical
  [axis=on,border=on,scale=small,
   size=medium]
\startchemical
  \chemical[FIVE,B,R,RZ][1,2,3,4,5]
\stopchemical
```

6-3-1

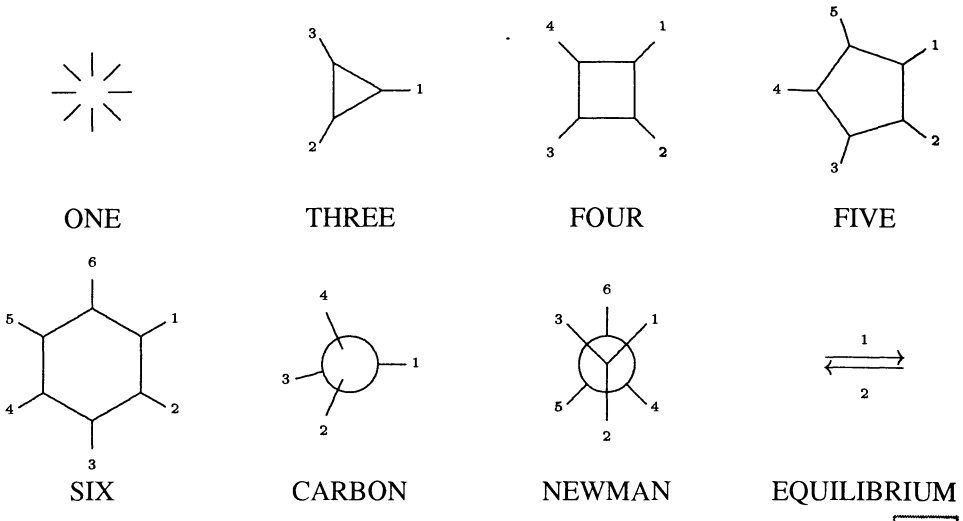
`\setupchemical{`*typesetting-data*`}`
`\startchemical{`*typesetting-data*`}`

Команда `\setupchemical` устанавливает различные общие параметры для последующего рисования химической формулы. Ее область действия ограничивается текущей группой. Инструкции по рисованию могут быть указаны в самой команде `\startchemical`, и в этом случае их действие продолжается до соответствующей команды `\stopchemical`, т. е. в пределах текущей формулы.




```
\usepackage {pictex,m-ch-en}
\setupchemical[axis=on]
\startchemical
  [width=6000,scale=small,size=medium
 \chemical[CARBON,CB1][A,B,C,D,E,F]
\stopchemical
```

6-3-2



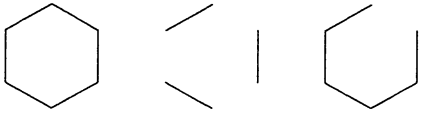
6-3-3

Рис. 6.2. Некоторые предопределенные структуры в PPCHTEX.

`\chemical[`*structure*`][`*atoms/molecule-list*`]`

Основная команда — `\chemical` — может появляться несколько раз между командами `\startchemical` и `\stopchemical`, которые ограничивают химическую формулу. Ее первый аргумент определяет структуру *structure*, которая заполняется данными, указанными во втором (необязательном) аргументе. Любой текст в *atoms/molecule-list* будет набран в математическом режиме (но прямым шрифтом). Если аргумент опущен, команда рисует скелет без атомов.

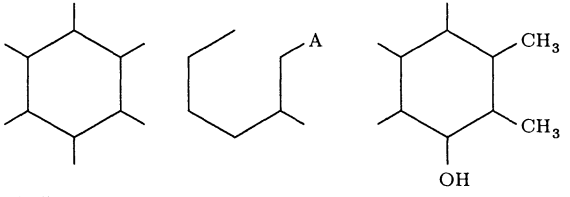
На рис. 6.2 показан перечень базовых структур, которые могут быть указаны в первом аргументе *structure*. Химические связи между атомами углерода внутри этих структур обозначаются буквой B с последующей цифрой. Это позволяет представить структуру несколькими способами, как показано в следующем примере для шестичленных структур:



```
\usepackage {pictex,m-ch-en}
\startchemical[scale=small,%
                size=medium]
 \chemical[SIX,B1,B2,B3,%
           B4,B5,B6]
 \qquad\qquad\qquad
 \chemical[SIX,B135]
 \qquad\qquad\qquad
 \chemical[SIX,B1..5]
\stopchemical
```

6-3-4

С помощью метки R к структуре могут быть добавлены субструктуры, положение которых в цикле также кодируется числом; эта метка просто рисует связи для субкомпонентов. Сама субкомпонента должна быть декларирована явным образом с помощью метки RZ, а ее «значение» (например, название атома) затем указано во втором (необязательном) аргументе. Если второй аргумент опущен или не содержит достаточно материала, никакого текста на месте субкомпоненты не помещается, и соответствующая метка RZ не оказывает никакого влияния:

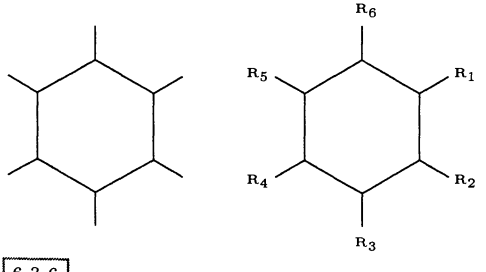


6-3-5

```
\usepackage {pictex,m-ch-en
\startchemical[scale=small,
                size=medium]
 \chemical[SIX,B1..6,R1..6]
\qquad\qquad\qquad
 \chemical[SIX,B1..5,%
           R12,RZ12][A]
\qquad\qquad\qquad\qquad
 \chemical[SIX,B1..6,%
           R1..6,RZ1..3]
          [CH_3,CH_3,OH]
\stopchemical
```

6.3.2 Определения

Достаточно просто построить библиотеку структурных элементов, которые могут использоваться затем в качестве строительных блоков более сложных структур. Например, с помощью команды `\definechemical` вы можете определить имя «`sixring`» для «неукрашенного» шестичленного кольца. Определив его так, как указано в примере ниже, вы можете затем нарисовать простое шестичленное кольцо без заместителей и добавить к нему шесть заместителей, указав их во втором аргументе:



6-3-6

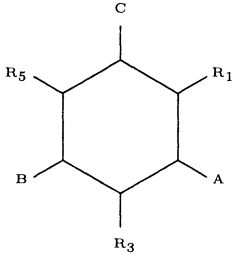
```
\usepackage {pictex,m-ch-en}
\definechemical[sixring]
   {\chemical[SIX,B,R,RZ]}
\startchemical[size=small]
 \chemical[sixring]
\qquad\qquad\qquad\qquad\qquad
 \chemical[sixring]
   [R_1,R_2,R_3,R_4,R_5,R_6]
\stopchemical
```

Можно построить и более сложные структуры. Помните, что субкомпоненты во втором аргументе перечисляются в соответствии с последовательностью,

Таблица 6.3. Идентификаторы связей в ppchtex.

Насыщенные связи	B	bond	SB	single bond
			-SB	left single bond
			+SB	right single bond
Ненасыщенные связи	EB	extra bond	DB	double bond
			TB	triple bond
Специальные связи	S	shortcut	C	circle
Связи к заместителям	R	radical	SR	single radical
	-R	left radical	-SR	single left radical
	+R	right radical	+SR	single right radical
Двойные связи к заместителям	ER	extra radical	DR	double radical
Атомы и молекулы (радикалы)	Z	atom	RZ	radical atom
			-RZ	left radical atom
			+RZ	right radical atom

в которой связи были указаны в первом аргументе. Примером этого (с немного измененным определением `sixring`) является:




```
\usepackage {pictex,m-ch-en}
\definechemical[sixring]
  {\chemical[SIX,B,R,RZ135]%
             [R_1,R_3,R_5]}
\startchemical[size=small]
  \chemical[sixring,SIX,RZ246]
           [A,B,C]
\stopchemical
```

6-3-7

6.3.3 Связи

В структурах, создаваемых с помощью пакета PPCHTEX, могут использоваться различные типы связей, список которых приведен в табл. 6.3. Пара столбцов, расположенных слева, относится к полным связям, а пара столбцов справа — к их укороченным вариантам. Связи могут быть укорочены с обеих сторон, слева (-) или справа (+). Укороченные варианты связей позволяют присоединять к ним атомы и молекулы.

За спецификаторами связей может быть указано одно или несколько чисел или их диапазон (например, B1, B135 или B1..5). Спецификатор без чисел (например, B) относится ко всем связям. Текст, который должен быть добавлен около связей, берется из второго (необязательного) аргумента `\chemical` в заданном порядке. Если таких аргументов несколько, то они нумеруются по часовой стрелке. С помощью спецификатора Z0 («Z-zero») текст может быть помещен и в центре структуры.

6.3.4 Комбинирование фрагментов

Для получения более сложных структурных формул описанные выше элементы могут сдвигаться и поворачиваться. Это осуществляется с помощью следующих спецификаторов:

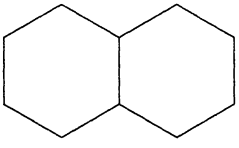
`MOV` (Move) сдвинуть идентичную структуру в направлении связи;

`ADJ` (Adjace) сдвинуть две структуры относительно друг друга в направлениях x или y так, чтобы они образовали общую сторону;

`SUB` (Substitute) сдвинуть две структуры относительно друг друга в направлениях x или y и соединить их с помощью связи;

`ROT` (Rotate) повернуть структуру.

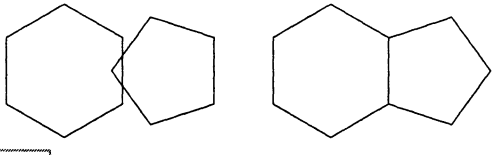
Обратите внимание, что команды могут оказывать различное действие на различные структуры. Например, угол поворота в командах `\chemical[FIVE,ROT1,B]` и `\chemical[SIX,ROT1,B]` различается. Несколько примеров использования этих команд приведено ниже.



6-3-8

```
\usepackage {pictex,m-ch-en}
\startchemical[size=small]
 \chemical[SIX,B,MOV1,B]
\stopchemical
```

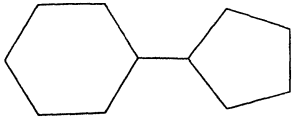
Следующий пример позволяет увидеть влияние спецификатора `ROT`:



6-3-9

```
\usepackage {pictex,m-ch-en}
\mbox{\startchemical[size=small]
 \chemical[SIX,B,ADJ1,FIVE,B]
\stopchemical\startchemical
 \chemical[SIX,B,ADJ1,FIVE,ROT3,
\stopchemical}
```

Как вы видите, связанные структуры строятся с помощью спецификатора `ADJ`. Часто одна из двух структур должна быть повернута с помощью `ROT`, чтобы получить правильное сочленение. Если между двумя структурами есть связь, то следует использовать спецификатор `SUB`. Повороты задаются шагами по 90 градусов по часовой стрелке, а перемещения, выполняемые с помощью `ADJ` и `SUB`, происходят в одном из четырех направлений.



6-3-10

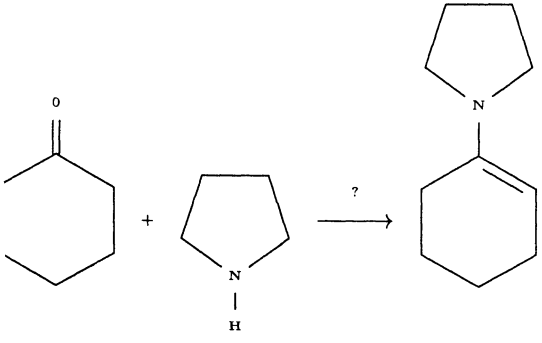
```
\usepackage {pictex,m-ch-en}
\startchemical
  \chemical[SIX,ROT2,B,R6,%
            SUB1,FIVE,B,R4]
\stopchemical
```

Приведенные выше примеры показывают, что последовательность, в которой указываются спецификаторы, должна следовать определенному порядку:

```
\chemical
  [структура и повороты,              % SIX, FIVE, ...
   связи внутри структуры,            % B, C, EB, ...
   связи вне структуры,               % R, DR, ...
   положение атомов,                  % Z
   положение субкомпонентов,          % RZ, -RZ, ...
   сочленение,                        % MOV, ADJ, ...
   структура и повороты,              % SIX, FIVE, ...
   связи внутри структуры,            % B, C, EB, ...
   ...
  ]
  [атомы, субкомпоненты]
```

6.3.5 Химические реакции

Уравнения реакций могут набираться с помощью специально созданных для этой цели символов, как показано в следующем примере:



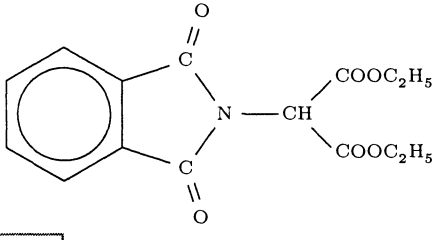

```
\usepackage {pictex,m-ch-en}
\setupchemical[size=small,
  width=fit,height=5500,
  under=1500]
\mbox{%
 \startchemical
   \chemical[SIX,B,ER6,RZ6]%
            [0]
 \stopchemical
 \startchemical
   \chemical[SPACE,PLUS,%
             SPACE]
 \stopchemical \startchemical
  \chemical[FIVE,ROT4,B125,%
   +SB3,-SB4,Z4,SR4,RZ4][N,H]
 \stopchemical
 \startchemical
    \chemical[SPACE,GIVES,%
              SPACE][?]
 \stopchemical \startchemical
  \chemical[SIX,B,EB6,R6,%
    SUB4,FIVE,ROT4,B125,%
    +SB3,-SB4,Z4][N]
 \stopchemical
}
```

6-3-11

Команда \mbox гарантирует, что структуры будут набраны в одну строку; символы GIVES и PLUS не требуют комментариев, а SPACE добавляет небольшой дополнительный пробел. Знак равновесия может быть задан с помощью EQUILIBRIUM. Кроме того, над знаками GIVES и EQUILIBRIUM можно набрать текст (в примере — вопросительный знак).

6.3.6 Дополнительные возможности

С помощью модификатора ONE текст, вводимый посредством ZO, может состоять из названия более чем одного атома. Если зарезервированного пространства не хватает, то связи 1, 2 и 8 могут быть сдвинуты с помощью команды OFF (что означает «offset»), как это показано в следующем примере, в котором смещение равно одному дополнительному символу («1»). Заметьте, что в подобных сложных структурах вращение лучше всего делать в конце. Команда CRZ (что значит «centered radical atom») используется, чтобы выровнять атом или молекулу со связью.




```
\usepackage {pictex,m-ch-en}
\startchemical[width=fit]
\chemical
  [SIX,B,C,ADJ1,FIVE,ROT3,SB34,
   +SB2,-SB5,Z345,DR35,SR4,CRZ35,
   SUB1,ONE,OFF1,SB258,ZO,Z28]
  [C,N,C,O,O,CH,COOC_2H_5,COOC_2H_5]
\stopchemical
```

6-3-12

Команда \chemical имеет еще две другие формы:

\chemical{*formula*}
\chemical{*formula*}{*text*}

Эти формы позволяют набирать формулы в виде абзаца, например:

$$2H_2 + O_2 \longrightarrow 2H_2O$$

```
\usepackage {pictex,m-ch-en}
\[ \chemical{2H_2}{}
   \chemical{PLUS}{}
   \chemical{O_2}{}
   \chemical{GIVES}{}
   \chemical{2H_2O}{} \]
```

6-3-13

Такие формулы также могут быть вставлены и внутрь абзаца; в этом случае выбирается меньший размер кегля. Заметим, что та же самая формула

могла бы быть набрана и с помощью одной из двух более коротких команд `\chemical`, в которых все указывается в первом аргументе:

```
\[\chemical{2H_2,PLUS,O_2,GIVES,2H_2O}{}\]
\[\chemical{2H_2,+,O_2,->,2H_2O}{}\]
```

Для добавления текста, поясняющего формулу, в командах `\chemical` может быть использован второй аргумент *text*. Например:

$$\underset{\text{hydrogen}}{2H_2} + \underset{\text{oxygen}}{O_2} \xrightarrow{\text{violent}} 2H_2O$$

```
\usepackage {pictex,m-ch-en}
\[ \chemical{2H_2}{hydrogen}
   \chemical{PLUS}{}
   \chemical{O_2}{oxygen}
   \chemical{GIVES}{violent}
   \chemical{2H_2O}{} \]
```

6-3-14

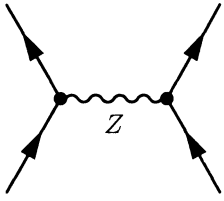
Рассматриваемый пакет также поддерживает и другие функции, такие, как закрашивание частей формулы.

6.4 Построение фейнмановских диаграмм

Фейнмановские диаграммы используются во многих областях физики чтобы организовать вычисления сумм большого числа вкладов элементарных процессов в различные физические величины. За эти годы было разработано несколько пакетов для рисования этих диаграмм. Пакет feynman Майкла Левина (Levine, 1990) является надстройкой над стандартным LaTeX'овским окружением `picture`. Это делает его полностью мобильным, но получаемые рисунки с неизбежностью оказываются несовершенными, а некоторые сложные графы часто вообще невозможно нарисовать. Пакет axodraw (автор — Йос Вермасерен) (Vermaseren, 1994) использует команды `\special` для прямого доступа к PostScript'овским примитивам для рисования диаграмм; этот подход очень гибок и создает визуально более привлекательные рисунки. Feynarts — пакет из Mathematica, в котором есть процедуры для расчета и рисования фейнмановских диаграмм (Eck and Küblbeck, 1991). В этом разделе, однако, мы подробнее рассмотрим развитую систему FeynMF Торстена Охля (1995, 1996), которая сама полностью разбирает формальную структуру фейнмановских графов, освобождая пользователя от необходимости вручную указывать положение низкоуровневых графических примитивов.

6.4.1 Использование FeynMF

Целью системы FeynMF является создание такого пользовательского интерфейса, при котором не требовалось бы указывать положение графов на уровне




```
\usepackage {feynmp}
\begin{fmfgraph*}(100,70)
  \fmfleft{em,ep}
  \fmf{fermion}{em,Zee,ep}
  \fmf{photon,label=$Z$}{Zee,Zff}
  \fmf{fermion}{fb,Zff,f}
  \fmfright{fb,f}
  \fmfdot{Zee,Zff}
\end{fmfgraph*}
```

6-4-1

Рис. 6.3. Простой пример использования пакета feynmf.

точек и кривых. Основными принципами, заложенными при разработке пакета, являются:

- *Простота и краткость* для диаграмм общего типа. Например, показанная на рис. 6.3 диаграмма рождения Z-частицы (широко используемая в физике высоких энергий) описывается всего восемью командами LaTeX'а. При этом положение вершин рассчитывается автоматически.
- *Выразительность* и *расширяемость* для сложных диаграмм.
- *Мобильность*: для построения диаграмм требуется только LaTeX и METAFONT (или METAPOST).
- *Интеграция* с LaTeX'ом: все метки и надписи (включая сложные математические выражения) делаются средствами LaTeX'а.

Существуют две версии пакета FeynMF: feynmf, которая для рисования использует METAFONT, и feynmp, которая использует METAPOST и позволяет также работать с цветом на диаграммах. Из-за этого программы немного отличаются на уровне LaTeX'а. Как и в гл. 3, мы будем писать META, если объяснения годятся для обеих программ.

Система работает путем записи кода METAFONT'а или METAPOST'а во внешний файл; при этом последовательность действий такова:

1. запускаем LaTeX; он записывает код METAFONT'а или METAPOST'а для всех рисунков в файл;
2. запускаем METAFONT или METAPOST; они создают либо набор PostScript'овских файлов (для METAPOST'а), либо шрифт с набором «глифов», по одному для каждого рисунка. Кроме этого, создается дополнительный (aux) файл с информацией о метках;
3. запускаем LaTeX снова; на этот раз он подключает рисунки или шрифт и информацию о метках.

Если код FeynMF в файле LaTeX'а не изменяется, то нет необходимости запускать METAFONT или METAPOST каждый раз, когда вы обрабатываете LaTeX'овский файл снова.

Выбор между METAFONT'ом или METAPOST'ом делается тогда, когда в преамбуле документа вы загружаете либо пакет feynmf, либо пакет feynmp. Команды, описывающие ваши диаграммы, помещаются внутри окружения `fmffile`, в котором в качестве аргумента указывается имя файла для промежуточного кода METAFONT'а или METAPOST'а. Таким образом, схематично LaTeX'овский файл должен выглядеть примерно так:

```
\documentclass{article}
\usepackage{feynmf}
\begin{document}
\begin{fmffile}{fmpict}
   .... diagram commands ....
\end{fmffile}
\end{document}
```

Поскольку мы выбрали пакет feynmf, то будет создан METAFONT'овский файл `fmpict.mf`, в котором каждый «глиф» будет отвечать одной диаграмме.

Во избежание конфликта имен между каждый раз переписываемыми файлами TeX'а и файлами METAFONT'а или METAPOST'а, обычно также имеющими расширение `log`, файл LaTeX'а *не может* иметь то же имя, что и файл FeynMF, указанный в качестве аргумента в окружении `fmffile` (`fmpict` в приведенном выше примере).

Когда файл, подобный приведенному выше, обрабатывается с помощью LaTeX'а в первый раз, выводится сообщение примерно следующего вида:

```
This is TeX, Version 3.14159 (C version 6.1)
(feynmfex.tex
LaTeX2e <1995/06/01> patch level 3
(/usr/local/lib/texmf/tex/latex/base/article.cls
Document Class: article 1995/06/26 v1.3g Standard LaTeX document class
(/usr/local/lib/texmf/tex/latex/base/size10.clo))
(/usr/local/lib/texmf/tex/latex/feynmf/feynmf.sty
Package: 'feynmf' v1.0 (rev. 1.12) <1995/05/06> (ohl))
No file feynmfex.aux.
feynmf: File fmpict.tfm not found:
feynmf: Process fmpict.mf with METAFONT and then reprocess this file.
feynmf: Label file fmpict.t1 not found:
feynmf: Process fmpict.mf with METAFONT and then reprocess this file.
[1] (feynmfex.aux) )
Output written on feynmfex.dvi (1 page, 388 bytes).
Transcript written on feynmfex.log.
```

Полученный при этом файл `fmpict.mf` должен быть далее обработан с помощью программы METAFONT, чтобы получить метрический файл TeX'овского шрифта `fmpict.tfm`, а также gf-файл в *общем формате*. Последний файл после этого преобразуется программой gftopk в упакованный растровый формат (pk-файл) с тем, чтобы он мог быть использован dvi-драйвером (см. гл. 11):

```
> virmf "\mode=localfont; \input fmpict"
This is METAFONT, Version
2.71 (C version 6.1) (fmpict.mf
(/usr/local/lib/texmf/fonts/feynmf/src/feynmf.mf)
:1:\fmfL(19.98138,14.59717,b){$Z$}% [1] )
Font metrics written on fmpict.tfm.
Output written on fmpict.300gf (1 character, 2184 bytes).
Transcript written on fmpict.log.
> gftopk fmpict.300gf
```

Когда для генерации рисунков вместо METAFONT'а используется METAPOST, система FeynMF записывает файл METAPOST'а с именем `fmpict.mp`. Когда программа METAPOST обрабатывает этот файл, для каждой из диаграмм во входном файле создаются PostScript'овские рисунки с именами `fmpict.`*n*, где *n* — порядковый номер рисунка. Они автоматически включаются в результирующий файл с помощью стандартного пакета graphics при последующем прогоне входного файла программой LaTeX.

6.4.2 Рисование диаграмм в пакете FeynMF

Вы можете использовать FeynMF в двух различных режимах в зависимости от того, как вы собираетесь решать конкретную задачу:

- *vertex mode (рисование вершин)*, в котором положение элементов автоматически рассчитывается из математического описания графа (включая положение вершин и дуг). При этом физические координаты обычно не вводятся;
- *immediate mode (непосредственный режим)*, в котором вы можете полностью контролировать все физические координаты, кодируя положение всех элементов в METAFONT.

Режим vertex mode и алгоритмическое конструирование диаграмм

Все, что необходимо, в этом режиме задается на уровне LaTeX'а, и никаких знаний METAFONT от пользователя не требуется. Для наибольшей гибкости FeynMF принимает *математическое* описание графа и автоматически создает из него соответствующую фейнмановскую диаграмму.

FeynMF имеет команды для размещения внешних вершин вдоль сторон диаграммы. Для расчета оптимальных положений *внутренних* вершин FeynMF минимизирует взвешенную сумму квадратов расстояний между ними с помощью METAFONT:

$$L(v_1, \ldots, v_n) = \frac{1}{2} \sum_{i,j} t_{ij} (v_i - v_j)^2 ,$$

где i, j обегают все комбинации вершин. Элементы матрицы «натяжений» t_{ij} по умолчанию считаются равными 1, однако пользователь может указать другие значения, чтобы немного изменить внешний вид диаграммы. Параметры «натяжений» можно рассматривать как резиновые полоски разной жесткости, которые позволяют вам притягивать или расталкивать соседние вершины (см. рис. 6.4).

Практика показывает, что наиболее эффективным способом построения фейнмановских диаграмм является комбинация поэтапного конструирования подграфов и, если необходимо, подстройки «натяжений». Часто установки по умолчанию сразу дают вполне удовлетворительный результат и требуется только небольшая подстройка «натяжений» какой-нибудь одной дуги или петли.

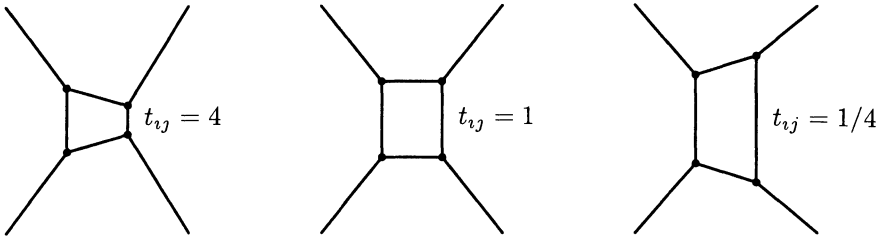


Рис. 6.4. Влияние изменения параметра «натяжения» для одной линии.

При построении диаграмм физики используют разнообразные линии, вершины и стили штриховки, и система FeynMF предоставляет средства для построения наиболее распространенных из них (см. табл. 6.4 и 6.5).

`\fmfleft{`v_1`,...}`	`\fmfleftn{`v`}{`n`}`
`\fmfright{`v_1`,...}`	`\fmfrightn{`v`}{`n`}`
`\fmftop{`v_1`,...}`	`\fmftopn{`v`}{`n`}`
`\fmfbottom{`v_1`,...}`	`\fmfbottomn{`v`}{`n`}`
`\fmfsurround{`v_1`,...}`	`\fmfsurround{`v`}{`n`}`

Это — основные команды FeynMF в vertex mode; они помещают набор внешних вершин $v_1,\ldots$ слева, справа, сверху, снизу или вокруг диаграммы. Группа команд в правом столбце (с суффиксом n) располагает все вершины с номерами v от 1 до n автоматически, и нет необходимости перечислять их явным образом.

`\fmfcurved` `\fmfstraight`

По умолчанию внешние вершины располагаются на гладкой кривой. Если указать команду `\fmfstraight`, то они будут располагаться на сторонах прямоугольника, ограничивающего диаграмму. То есть, команды `\fmfcurved` и `\fmfstraight` позволяют выбрать одну из двух альтернатив.

`\fmf{`*lsty*`}{`v_1`,...}`	`\fmfn{`*lsty*`}{`v`}{`n`}`
`\fmfcyclen{`*lsty*`}{`v`}{`n`}`	`\fmfrcyclen{`*lsty*`}{`v`}{`n`}`

Команда `\fmf` соединяет набор вершин $v_1,\ldots$ линиями со стилем *lsty* (см. табл. 6.5). Стиль линии можно дополнительно уточнить, указав несколько опций (см. табл. 6.6). Например,

```
\fmf{fermion,tension=.5}{vw,vn,ve,vs,vw}
```

соединяет указанные внутренние вершины линией «`fermion`», используя значение параметра «натяжения», равного одной второй. Остальные команды `\fmfn`,

Таблица 6.4. Вершины и стили штриховки в FeynMF.

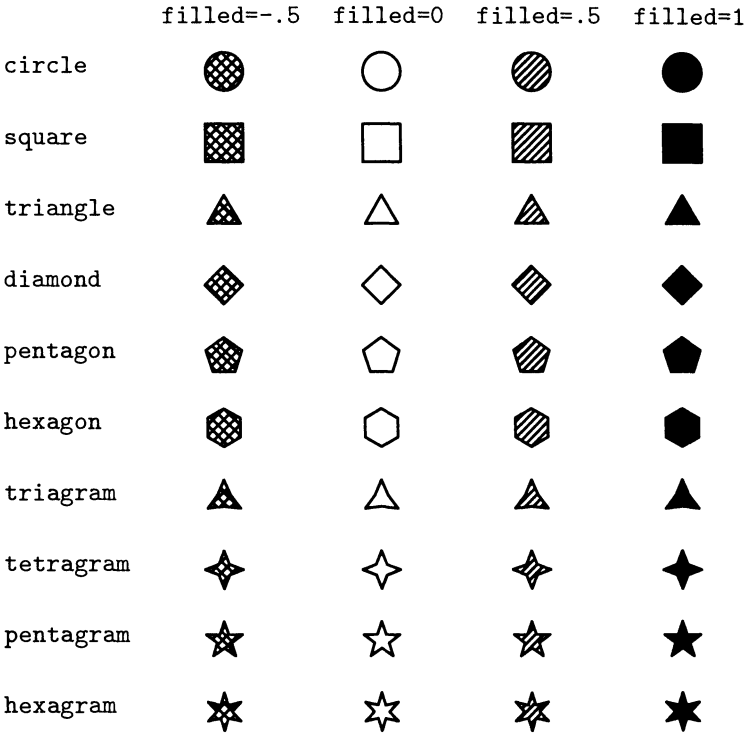
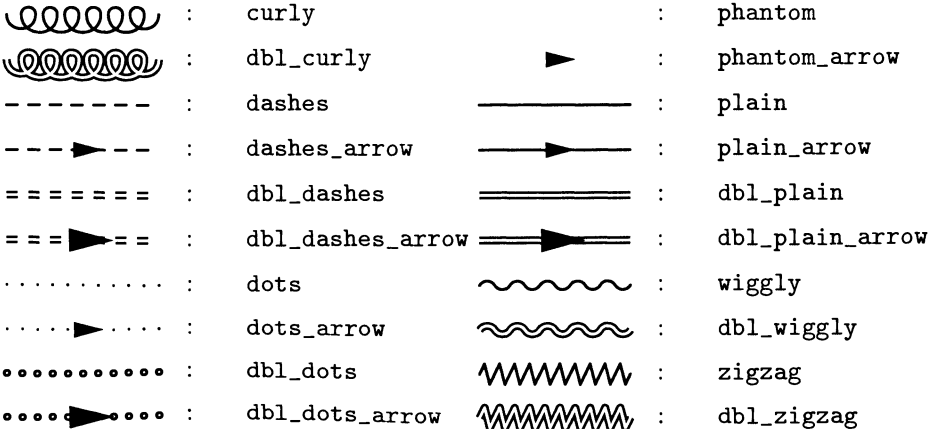


Таблица 6.5. Стили линий в FeynMF.



`\fmfcyclen` и `\fmfrcyclen` соединяют вершины $v_1, \ldots, v_n$ обычным способом, соответственно циклически или циклически в обратном порядке.

`\fmfpen{`*wgt*`}`

Эта команда устанавливает толщину (вес) линии равной *wgt*. Предопределенными размерами являются `thin` и `thick`. Чтобы изменить ширину индивидуальной дуги, используйте в качестве `width` параметры, перечисленные в табл. 6.6.

`\fmfv{`*vopt*`}{`v_1`,...}` `\fmfvn{`*vopt*`}{`v`}{`n`}`

Команда `\fmfv` задает набор внутренних вершин v_1,... со стилем оформления, определенным опциями *vopt* (`\fmfvn` делает то же самое для n вершин v_1 .. v_n). В табл. 6.4 показаны некоторые из допустимых форм вершин и стилей их штриховки, а в табл. 6.7 — возможные значения опций, относящихся к вершинам[1]

`\fmfdot{`v_1`,...}` `\fmfdotn{`v`}{`n`}`

Это — специальный случай команды `\fmfv`, в котором вершины изображаются точками. Например, две следующие команды эквивалентны:

```
\fmfdotn{v}{4}
\fmfv{decor.shape=circle,decor.filled=full,
      decor.size=2thick}{v1,v2,v3,v4}
```

`\fmfblob{`*dia*`}{`v_1`,...}` `\fmfblobn{`*dia*`}{`v`}{`n`}`

Это сокращение было введено Торстеном Охлем для рисования вершин в виде заштрихованных кругов («blob»). Две приведенные ниже команды дают один и тот же результат:

```
\fmfv{decor.shape=circle,decor.filled=shaded,
      decor.size=5mm}{vblob}
\fmfblob{5mm}{vblob}
```

`\fmfpoly{`*vopt*`}{`v_1`,...}` `\fmpolyn{`*vopt*`}{`v`}{`n`}`

Вершины сложной формы широко используются в физике твердого тела. Они могут быть нарисованы с помощью многоугольников. Команда `\fmfpoly` располагает вершины v_1,... на многоугольнике; форму вершин и другие свойства

[1] Как следует из последней таблицы, с помощью обсуждаемых команд может быть задан и текст, который будет набран рядом с вершиной. — *Прим. перев.*

Таблица 6.6. Опции FeynMF для рисования линий.

Опция	*Объяснение*
`label`	TEX'овский текст, используемый для метки
`label.side`	поместить текст «`left`» (слева) или «`right`» (справа)
`label.dist`	поместить текст на указанном расстоянии
`left`	нарисовать полуокружность слева
`right`	нарисовать полуокружность справа
`straight`	нарисовать прямую линию (используется по умолчанию)
`tag`	метка для однозначного определения дуги (если нужно)
`tension`	нарисовать дугу компактнее (> 1) или свободнее (< 1)
`width`	толщина линии
`foreground`	цвет основных линий (только для METAPOST'а)
`background`	цвет фона (только для METAPOST'а)

при этом можно изменять с помощью опций *vopt* (`\fmpolyn` делает то же для вершин $v_1, \ldots, v_n$). Допустимые опции команды перечислены в табл. 6.8.

`\fmffreeze`

Ряд команд позволяет вносить коррективы в алгоритм автоматического расчета положения элементов в FeynMF. Среди них, пожалуй, наиболее важной является команда `\fmffreeze`, которая рассчитывает диаграмму вплоть до данной точки и «замораживает» ее, так что дуги, добавляемые после, уже не влияют на положение точек. Этот важный прием использования «каркасов» при конструировании диаграмм подробно описан в руководстве (Ohl, 1996); пример его применения показан на рис. 6.6.

Использование программы METAPOST позволяет добавить цвет в ваши диаграммы, указав спецификации `foreground` и `background` в опциях рисования линий и вершин (см. табл. 6.6 и 6.7). Предопределенными цветами являются `white`, `black`, `red`, `green` и `blue` (белый, черный, красный, зеленый, синий); другие цвета могут быть указаны как тройки RGB (red, green, blue). Например, `foreground=(1,,0,,1)`[1] и `foreground=red+blue` эквивалентны. Для дуг цвет фона используется только для внутренней части между двойными линиями. Например, красная линия глюона между вершинами `in` и `out` рисуется с помощью команды:

```
\fmf{gluon,fore=red}{in,out}
```

Заметим, что ключевые слова в опциях можно сокращать до их кратчайшей однозначно определяемой формы, как, например, `fore` вместо `foreground` в предыдущем примере. Это работает и для каждой из отделенных точкой компонент названия опции: так, `l.d` интерпретируется как `label.dist`.

[1] Обратите внимание на использование двух запятых «`,,`», необходимых для того, чтобы однозначно отличать запятую как разделитель параметров в команде от запятых, передающих значения для конкретного параметра.

Таблица 6.7. Опции FeynMF для рисования вершин.

Опция	*Объяснение*
`decoration.shape`	форма вершины
`decoration.size`	размер вершины
`decoration.filled`	штриховка вершины
`decoration.angle`	повернуть вершину
`label`	TEX'овский текст, располагаемый рядом с вершиной
`label.angle`	поместить текст в направлении под углом относительно вершины
`label.dist`	поместить текст на заданном расстоянии
`foreground`	цвет основных линий (только для METAPOST'a)
`background`	цвет фона (только для METAPOST'a)

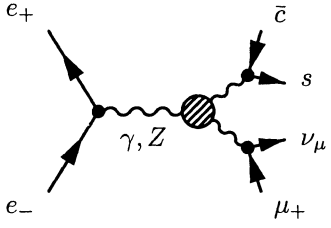
Таблица 6.8. Опции FeynMF для многоугольников.

Опция	*Объяснение*
`empty`	рисуются только контуры
`filled`	тип штриховки
`full`	закрасить внутреннюю область
`hatched`	заштриховать внутреннюю область
`label`	TEX'овский текст, помещаемый рядом с многоугольником
`label.angle`	поместить текст в направлении под углом относительно вершины
`label.dist`	поместить текст на заданном расстоянии
`phantom`	ничего не рисуется
`pull`	«втянуть» стороны между вершинами (<0) или «раздуть» их (>0)
`shade`	затенить внутреннюю область
`smooth`	закруглить углы
`tension`	«натяжение», используемое для углов
`foreground`	цвет основных линий (только для METAPOST'a)
`background`	цвет фона (только для METAPOST'a)

FeynMF умеет рассчитывать оптимальные положения надписей с помощью METAFONT'a. Поскольку METAFONT может писать только в свой `log`-файл, информация о позиционировании, необходимая для набора надписи с помощью LATEX'a, записывается в этот файл, который после читается и сканируется программой LATEX. По умолчанию все надписи помещаются рядом с дугой или вершиной, с которой они связаны. Разумеется, возможно и явное задание положения надписи пользователем; как это делается, описано в руководстве.

Чтобы дать почувствовать, как строить фейнмановские диаграммы в vertex mode, рассмотрим диаграмму, показанную на рис. 6.5 на следующей странице. Окружение `fmfgraph` содержит описание простой фейнмановской диаграммы. По аналогии с LATEX'овским стандартным окружением `picture`, параметры указывают ширину и высоту диаграммы в единицах `\unitlength`.

```
1  \usepackage {feynmp}
2  \begin{fmfgraph*}(100,60)
3   \fmfleftn{i}{2} \fmfrightn{o}{4}
4   \fmflabel{$e_-$}{i1}\fmflabel{$e_+$}{i2}
5   \fmflabel{$\mu_+$}{o1}
6   \fmflabel{$\nu_{\mu}$}{o2}
7   \fmflabel{$s$}{o3}
8   \fmflabel{$\bar c$}{o4}
9   \fmf{fermion}{i1,v1,i2}
10  \fmf{boson,label=$\gamma,,Z$}{v1,v2}
11  \fmf{boson}{v3,v2,v4}
12  \fmf{fermion}{o1,v3,o2}
13  \fmf{fermion}{o4,v4,o3}
14  \fmfdot{v1,v3,v4}\fmfblob{.12w}{v2}
15  \end{fmfgraph*}
```



6-4-2

Рис. 6.5. Рисование фейнмановской диаграммы в режиме vertex mode.

Это окружение не разрешает добавлять надписи к диаграмме. Чтобы их добавить (например, пометить внутреннюю волнистую линию и внешние вершины на нашем рисунке), надо использовать версию окружения со звездочкой `fmfgraph*`.

В строке 3 на рис. 6.5 декларируются две входящие слева частицы (с помощью команды `\fmfleftn`) и четыре выходящие направо частицы (с помощью команды `\fmfrightn`), а в строках 4–8 им приписываются обозначения. Внутренние вершины пронумерованы от `v1` до `v4` слева направо. В строке 9 входящие фермионы `i1` и `i2` соединяются с первой внутренней вершиной `v1`. Затем в строке 10 эта вершина соединяется бозонной линией с левой стороной центра («blob») (внутренней вершиной `v2`) и добавляется надпись. В строке 11 рисуется бозонная линия между внутренними вершинами `v2`, `v3` и `v4`, а в строках 12–13 эти две последние внутренние вершины соединяются с выходящими фермионными линиями. Наконец, в строке 14 вершины `v1`, `v3` и `v4` изображаются точками, а в вершине `v2` помещается кружок с диаметром, равным `.12w` (где `w` — полная ширина диаграммы), что составляет 0.12×50 мм или 6 мм.

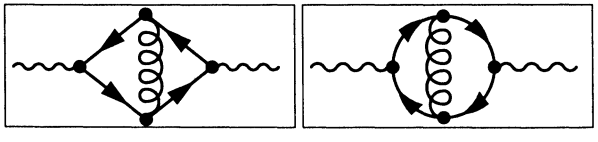
Режим immediate mode

Режим рисования вершин (vertex mode) в FeynMF, который обсуждался в предыдущем разделе, оперирует с абстрактными вершинами, а результат зависит от того, как они соединены. Во многих случаях такой «автоматический»

режим достаточен для того, чтобы получить желаемую диаграмму. Однако, за небольшими исключениями, этот режим позволяет работать только с прямыми линиями. Следовательно, если вам нужны изогнутые линии, вам следует использовать *непосредственный* (*immediate*) режим FeynMF. Этот режим также дает возможность более точного контроля за позиционированием элементов диаграммы, поскольку он контролирует как координаты вершин, так и соединяющие их дуги (*пути* (*paths*) в Knuth (1986c)).

Рассмотрим диаграммы с петлями, показанные на рис. 6.6. Левая диаграмма была нарисована в режиме vertex mode, а правая — в immediate mode. Видно, что последняя имеет более приятный вид.

```
\usepackage {feynmp}
\fbox{\begin{fmfgraph}(100,40)
 \fmfleft{w}\fmfright{e}
 \fmf{boson}{w,vw}\fmf{boson}{ve,e}
 \fmf{fermion,tension=.5}{vw,vn,ve,vs,vw}
 \fmf{gluon}{vn,vs}
 \fmffixed{(0,h)}{vn,vs}
 \fmfdot{vw,vn,ve,vs}
\end{fmfgraph}}
\fbox{\begin{fmfgraph}(100,40)
 \fmfleft{w}\fmfright{e}
 \fmf{boson}{w,vw}\fmf{boson}{ve,e}
 \fmf{phantom,left,tension=.4}{vw,ve,vw}
 \fmfdot{vw,ve}
 \fmffreeze
 \fmfipath{pn,ps}\fmfipair{vn,vs}
 \fmfiequ{pn}{vpath (__vw, __ve)}
 \fmfiequ{ps}{vpath (__ve, __vw)}
 \fmfiequ{vn}{point .5length(pn) of pn}
 \fmfiequ{vs}{point .5length(ps) of ps}
 \fmfi{fermion}{subpath (0,.5)*length(pn) of pn}
 \fmfi{fermion}{subpath (.5,1)*length(pn) of pn}
 \fmfi{fermion}{subpath (0,.5)*length(ps) of ps}
 \fmfi{fermion}{subpath (.5,1)*length(ps) of ps}
 \fmfi{gluon}{vn--vs}
 \fmfiv{dec.sh=circle,dec.siz=2thick}{vn}
 \fmfiv{dec.sh=circle,dec.siz=2thick}{vs}
\end{fmfgraph}}
```



6-4-3

Рис. 6.6. Диаграмма, нарисованная в режиме vertex mode, и ее улучшенная версия, полученная с помощью команд в immediate mode.

Для ясности вершины на обеих петлевых диаграммах обозначены, если двигаться по часовой стрелке, как `vw`, `vn`, `ve` и `vs` (что значит west — запад, north — север, east — восток и south — юг). Для первой диаграммы на строке 5 для контроля фермионной петли используется параметр натяжения `tension`. Команда `\fmffixed` в строке 7 фиксирует расстояние между указанными вершинами в списке. Здесь расстояние между верхней и нижней вершинами принимается равным высоте диаграммы `h`, т. е. 20 «единицам» (это ограничение используется на стадии обработки METAFONT'ом при расчете положения диаграммы). Без этой команды петля сжалась бы (коллапсировала).

Рассмотрим теперь «улучшенную» диаграмму. Строки 11–12, отвечающие «внешним» линиям, аналогичны строкам на предыдущей диаграмме. Команда `\fmffreeze` (строка 15) обеспечивает, что уже построенная часть диаграммы останется неизменной (т. е. на нее не будут влиять последующие команды FeynMF). Начиная со строки 16 и далее мы используем команды режима *immediate* FeynMF, каждая из которых начинается с четырех букв `fmfi`[1]. Команды `\fmfipath` и `\fmfipair` декларируют, соответственно, путь и пару координат для METAFONT. Строки 17–20 являются командами присваивания (*arg1*=*arg2*). Обратите внимание на команды `vpath`, которые берут из METAFONT путь между двумя вершинами (после `\fmffreeze`). Кроме того, заметьте, что перед именами вершин нужно указывать двойное подчеркивание (например, `ve` становится `__ve`). Команды `\fmfi` в строках 21–25 рисуют линию со стилем, заданным первым аргументом, вдоль пути, заданного вторым аргументом. Строка 25 также содержит оператор METAFONT `--`, который формирует прямую линию (для глюона). Наконец, в строках 26–27 с помощью команды `\fmfiv` рисуются вершины в координатах METAFONT, указанных во втором аргументе.

6.4.3 Расширение FeynMF

Иногда бывает нужно выйти за рамки предопределенных функций FeynMF и использовать непосредственно команды METAFONT. Это можно сделать либо добавляя команды из файла METAFONT, либо с помощью команды `\fmfcmd` системы FeynMF:

`\fmfcmd{`*MFcmds*`}`

Команда `\fmfcmd` записывает команды METAFONT *MFcmds* прямо в выходной файл. Этот способ может оказаться полезным, например, для определения новых стилей линий. Макро METAFONT `style_def` используется для регистрации нового стиля в FeynMF и для определения макро, который будет вызываться каждый раз, когда появляется ссылка на новый стиль (например, в первом аргументе команды `\fmf`). Подобные функции называются *трансформерами* (*transformers*)

[1] Для детального понимания этих команд вам следует иметь представление о конструкциях METAFONT, например о том, как они соединяют вершины с помощью кривых Безье (см. Knuth (1986c)).

так как они берут путь META в качестве своего аргумента и возвращают трансформированный (приукрашенный) путь. Этот способ уже был использован для того, чтобы получить различные стили линий в табл. 6.5 на с. 277. Например, чтобы преобразовать линию в волнистую линию и добавить к ней стрелку, вы можете использовать предопределенный стиль `wiggly`, к которому затем добавляется стрелка:

```
\fmfcmd{%
 style_def charged_boson expr p =
 draw (wiggly p);
 fill (arrow p)
 enddef;}
```

В общем, все команды META, связанные с путями, оказываются доступными. Для работы с цветом (с помощью METAPOST'а) вам следует явно использовать работающие с цветом функции feynmp, такие, как `cdraw`, `cfill` и т. п.

6.5 Набор временны́х диаграмм

Йенс Лайлих и Людвиг Май написали пакет timing для набора временны́х диаграмм в цифровой технике. Они разработали METAFONT'овский алфавит символов и использовали METAFONT'овский механизм лигатур для набора логических переходов. На рис. 6.7 показан пример использования пакета timing.

6.5.1 Команды в окружении `timing`

Команды, описанные в этом разделе, могут использоваться только внутри окружения `timing`. Оно является расширением LaTeX'овского окружения `picture`, и поэтому все команды, используемые в `picture`, также оказываются доступными. Выбранной единицей длины `\unitlength` является `1sp`.

`\begin{timing}[`*symbol-type*`]{`*label-width*`}` ... `\end{timing}`

Необязательный аргумент *symbol-type* указывает, какой из четырех вариантов «шрифтов» для временны́х диаграмм следует использовать. Значением этого аргумента могут быть `1`, `1s`, `2` или `2s`, где цифра обозначает ширину (около одного или двух мм, соответственно), а буква `s` указывает, что для соединения уровней сигналов используются наклонные линии вместо вертикальных. По умолчанию используется шрифт типа 2. В табл. 6.9 приведены все состояния сигналов, указаны кодирующие их буквы и даны примеры во всех четырех вариантах шрифтов. Обязательный аргумент *label-width* задает величину наиболее длинной метки, которая помещается рядом с линией сигналов (см. рис. 6.7).

```
\usepackage {timing}
\begin{timing}[2s]{1.4cm}
\tnote{0.5}{4}{$\mathrm{T}_1$}
\tnote{0.5}{12}{$\mathrm{T}_2$}\tnote{0.5}{20}{$\mathrm{T}_i$}
\tnote{0.5}{28}{$\mathrm{T}_1$}\tnote{0.5}{36}{$\mathrm{T}_2$}
\tnote{0.5}{44}{$\mathrm{T}_i$}\tnote{0.5}{52}{$\mathrm{T}_1$}
%% Clock              ....1111....2222....iiii....1111....2222....iiii....1111
\tin{1}{CLK}   \til{1}{HHHHLLLLLHHHHLLLLHHHHLLLLHHHHLLLLHHHHLLLLHHHHLLLLHHHHLLLL}
%% Adresses line      ....1111....2222....iiii....1111....2222....iiii....1111
\tin{2}{ADDR}  \til{2}{VVVVXVVVVVVVVVVVVVVVXVVVVVVVXVVVVVVVVVVVVVVVXVVVVVVXVVVV}
\tnote{1.85}{10}{Valid}\tnote{1.85}{22}{Invalid}%
\tnote{1.85}{34}{Valid}\tnote{1.85}{46}{Invalid}
%% Adresses status    ....1111....2222....iiii....1111....2222....iiii....1111
\tin{3}{ADS\#} \til{3}{HHHHLLLLLLLLLHHHHHHHHHHHHHHHHLLLLLLLLHHHHHHHHHHHHHHHLLLL}
%% Write/Read         ....1111....2222....iiii....1111....2222....iiii....1111
\tin{4}{W/R\#} \til{4}{HHHHLLLLLLLLLLLLLLLLFFFFFFFFHHHHHHHHHHHHHHHHFFFFFFFLLLL}
%% Burst ready        ....1111....2222....iiii....1111....2222....iiii....1111
\tin{5}{BRDY\#}\til{5}{UUUUUUUUUUUUZZZZZZZZUUUUUUUUUUUUUUUUZZZZZZZZUUUUUUUUUUU}
%% Data lines         ....1111....2222....iiii....1111....2222....iiii....1111
\tin{6}{DATA}  \til{6}{ZZZZZZZZZZZZVVVVVVVVZZZZZZZZZZZZZZZZVVVVVVVVZZZZZZZZZZZZ}
\tnote{5.85}{14}{To CPU}\tnote{5.85}{37}{From CPU}
\sline{0.6}{0}{6.}\sline{0.6}{8}{6.}\sline{0.3}{16}{5.5}\sline{0.6}{24}{1.5}
\sline{2.1}{24}{6.}\sline{0.6}{32}{6.}\sline{0.3}{40}{5.5}\sline{0.6}{48}{1.5}
\sline{2.1}{48}{6.}\sline{0.6}{56}{6.}
\end{timing}
```

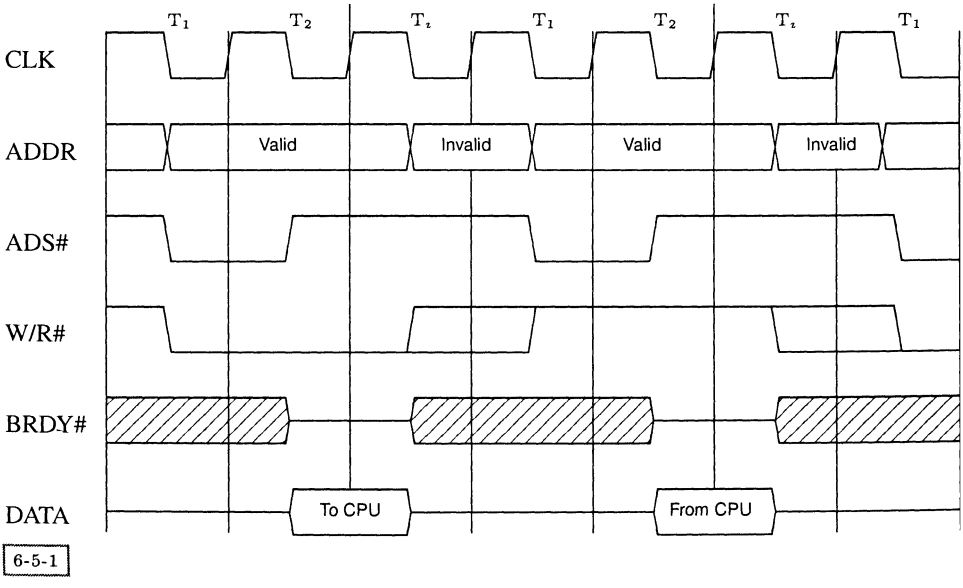
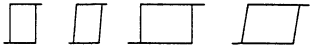
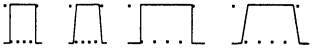
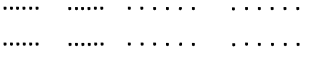
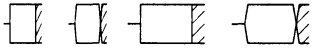
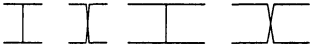
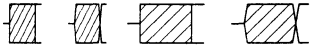
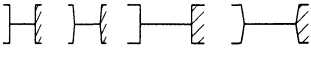
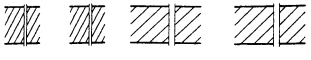


Рис. 6.7. Временна́я диаграмма цикла чтения из памяти с последующей записью в нее.

6.9. Кодирование состояния сигналов и их представление во всех вариантах в окружения `timing`.

Состояние сигнала	*Код*	Варианты шрифта 1	1s	2	2s
Низкий уровень	`HLLLLH` →				
Высокий уровень	`LHHHHL` →				
Плавающее состояние	`LFFFFH` →				
Низкий уровень с метками	`hllllh` →				
Высокий уровень с метками	`lhhhhl` →				
Пустая линия с метками	`.....` →				
Достоверные данные	`ZVVVVU` →				
Изменение состояния на шине	`VVVXVV` →				
Недостоверные данные	`ZUUUUV` →				
«Третье состояние»	`VZZZZU` →				
Верхняя линия с временны́ми метками	`TtttTt` →				
Верхняя линия без временны́х меток	`TtTtTt` →				
Нижняя линия с временны́ми метками	`BbbbBb` →				
Нижняя линия без временны́х меток	`BbBbBb` →				
Знак разрыва	`UUU-UU` →				

`\til{`*y-pos*`}{`*symbols*`}`

Линия сигналов на временно́й диаграмме набирается с помощью команды `\til`; *y-pos* указывает положение линии на диаграмме по вертикали. В большинстве случаев вам следует использовать последовательно идущие целочисленные значения этого аргумента. Второй аргумент *symbols* содержит комбинацию букв (см. табл. 6.9), которые кодируют состояния сигнала. Из-за способа, с помощью которого реализованы шрифты символов, лучше всего записывать по крайней мере две идентичные буквы для представления каждого состояния. В противном случае механизм лигатур, с помощью которого рисуются переходы между состояниями, может работать некорректно.

Аргумент *symbols* может также содержать одну из следующих команд:

`\timingcounter{`*separation*`}{`*Start-value*`}{`*End-value*`}{`*Interval*`}`
`\conttimingcounter{`*separation*`}{`*Start-value*`}{`*End-value*`}{`*Interval*`}`

С их помощью набирается шкала, содержащая отсчеты времени. Вторая форма, `\conttimingcounter`, используется для построения диаграмм с разрывом на оси времени, которая оставляет для этого необходимое пространство.

`\tin{`*y-pos*`}{`*text*`}`

Команда `\tin` вставляет надпись рядом с линией сигналов. Если использовать то же самое значение *y-pos*, как и в соответствующей команде `\til`, текст надписи *text* правильно центрируется слева от вертикальной линии.

`\tnote{`*y-pos*`}{`*x-pos*`}{`*text*`}`

В отличие от предыдущей, команда `\tnote` позволяет расположить аннотацию в любом месте на линии сигналов. Как и в предыдущей команде, значение *y-pos* должно соответствовать значению, указанному в команде `\til`, хотя, возможно, вы захотите немного подправить его, чтобы сдвинуть надпись *text* по вертикали. Параметр *x-pos* указывает начальную позицию по горизонтали; при этом за единицу длины принимается ширина символов, заданная в аргументе *symbols* команды `\til`. Например, значение `5` означает, что положение надписи будет идти за `LHHHL`.

`\rarw{`*y-pos*`}{`*x-pos*`}{`*length*`}{`*text*`}`
`\larw{`*y-pos*`}{`*x-pos*`}{`*length*`}{`*text*`}`

Эти две команды рисуют горизонтальные стрелки: `\rarw` — стрелку, направленную направо, а `\larw` — налево. Чтобы расположить эти стрелки над линиями сигналов, надо задать значение *y-pos* немного меньше значения, указанного для линии (например, `0.6`).

`\sline{`*y-pos*`}{`*x-pos*`}{`*y2-pos*`}`

С помощью команды `\sline` рисуется вертикальная линия, которая начинается в точке (*x-pos*,*y-pos*) и заканчивается в точке (*x-pos*,*y2-pos*). Ширину этой линии можно менять с помощью LaTeX'овской декларации `\linethickness`.

6.5.2 Настройка

Вид полученной диаграммы может быть улучшен путем подгонки значения переменной

`\timescalefactor`

которая управляет расстоянием между линиями (по умолчанию ее значение равно 2, что означает, что между двумя линиями сигналов, лежащими на расстоянии одного *y-pos*, вставляется одна пустая строка), и значения переменной

`\timadjust`

которая подгоняет положение вертикальных линий (по умолчанию ее значение равно `0pt`). Последняя команда может помочь в случае, когда драйвер принтера не позиционирует вертикальные линии точно в середине переходов между состояниями. Обе эти переменные переопределяются с помощью команды `\renewcommand`.

6.6 Диаграммы в электронике и оптике

Пакет circ, созданный Себастьяном Таннертом и Андреасом Тилле (Tannert and Tille, 1996), может использоваться для набора электронных схем и диаграмм в оптике. Этот пакет предоставляет удобный интерфейс для рисования диаграмм, содержащих не только резисторы, конденсаторы и транзисторы, но также линзы, зеркала и т. п. Поскольку «символы» закодированы на METAFONT'е, полученные рисунки могут быть распечатаны или показаны на любом устройстве.

Принцип рисования в пакете circ напоминает принцип черепашьей графики (turtle system): все элементы и линии рисуются по отношению к «текущей» точке, которая передвигается автоматически, хотя при необходимости ее положение и направление движения могут задаваться вручную. В пакете имеются команды для *рисования (draw)*, *подгонки (justify)*, *связывания (link)* и *позиционирования (position)* элементов и линий соединения абсолютно или относительно друг друга.

Пакет состоит из несколько частей, которые могут вызываться по отдельности с помощью опций в команде `\usepackage`:

`basic` основные элементы, такие, как резисторы, конденсаторы, выключатели, диоды, транзисторы (см. табл. 6.10);

`box` «черный ящик», осциллографы, генераторы, усилители, преобразователи сигналов;
`gate` операционные усилители и логические элементы, такие, как И-НЕ;
`ic` интегральные схемы, такие, как триггеры (см. табл. 6.11);
`optics` оптические элементы, такие, как линзы, зеркала и т. п. (см. табл. 6.12).

6.6.1 Основные команды для рисования диаграмм

Каждая электрическая схема включается в окружение `circuit`:

`\begin{circuit}{`*magstep*`}` ... `\end{circuit}`

Аргумент *magstep* является целым числом (в пределах от 0 до 4), который указывает размер элементов. В табл. 6.10–6.12 приведены некоторые команды и изображения соответствующих им элементов схем, которые можно вызывать внутри окружения `circuit`.

Общий синтаксис команд рисования имеет вид:

`\symbolname` *number*␣*label*␣*specs*␣*dir*

При вызове пакета с опцией `basic` имеется более 60 команд рисования, из которых можно выбирать. Аргумент *number* является дополнительным идентификатором и обеспечивает уникальность комбинации `\symbolname` и *number* на одной диаграмме; это необходимо для того, чтобы различные элементы схемы могли быть однозначно соединены друг с другом. Направление, в котором рисуются элементы, определяется параметром *dir* (`h` для горизонтального, `v` для вертикального, `l` для левого и т. п.). Рядом с каждым элементом может быть помещена надпись, задаваемая с помощью параметра *label* (обратите внимание на положение элементов и надписей по отношению к текущей точке в табл. 6.10–6.12). Необязательный параметр *spec* необходим для элементов с несколькими выводами (переменных резисторов, транзисторов и т. д.) и указывает вывод, отвечающий текущей точке рисования.

Кроме надписи, создаваемой рядом с элементом с помощью метки *label*, элемент также автоматически помечается с помощью его аббревиатуры с индексом *number* (например, R_1). Можно отменить формирование надписи, указав перед командой префикс `\nv`, а формирование аббревиатуры — указав соответственно префикс `\ln`. Кроме того, с помощью префикса `\cc` можно поменять местами положение обеих меток.

`\.`*number* `\junction` *number*

Точки соединения создаются с помощью команд `\.` или `\junction`. Единственным параметром здесь является *number*, который помечает эту точку для последующих ссылок.

Таблица 6.10. Некоторые основные элементы электронных схем.

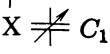
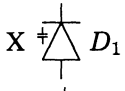
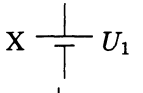
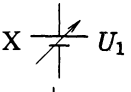
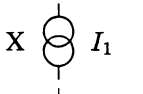
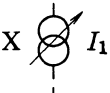
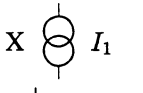
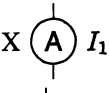
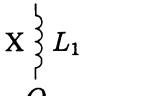
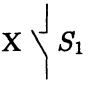
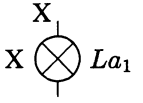
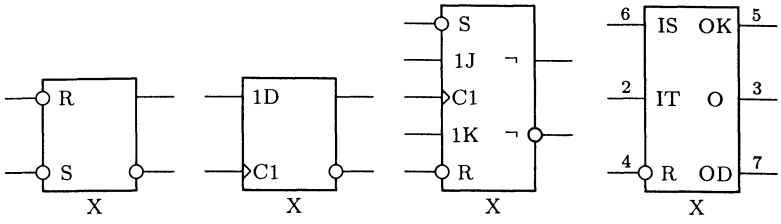
(переменный) резистор	\R		\Rvar	
(переменный) конденсатор	\C		\Cvar	
электролитический конденсатор	\Cel			
(зенеровский) диод	\D		\ZD	
светодиод	\LED			
варикап	\Dcap			
(регулируемый) источник напряжения	\U		\Uvar	
(регулируемый) источник тока	\I		\IvaR	
вольтметр, амперметр	\V		\A	
катушка индуктивности, выключатель	\L		\S	
кварцевый резонатор	\Q			
лампа накаливания	\La			
npn- pnp-транзистор	\npn		\pnp	
полевой транзистор с каналом n- p-типа	\nfet		\pfet	
полевой транзистор с изолированным затвором и каналом n- p-типа	\nvmos		\pvmos	

Таблица 6.11. Некоторые элементы интегральных схем.



\NRFSS	\DFF	\JKMSFF	\fff
RS-триггер	D-триггер	JK-триггер	таймер NE 555

6-6-2

\-␣*len*␣*dir*	\wire␣*len*␣*dir*

Простые линии соединения между элементами рисуются с помощью команд \- или \wire. Первый аргумент команды *len* указывает длину линии в масштабе 2.5 мм на один шаг, а второй аргумент *dir* — направление (`l` налево, `r` направо, `u` вверх, `d` вниз). Таким образом, команда \-␣8␣u рисует линию соединения длиной 8 единиц (2 см) в направлении вверх. Варианты \dashed, \bundle и \wwire рисуют штриховую линию, «пучок» проводов и пару линий соответственно.

\htopin␣*pinref*␣ \vtopin␣*pinref*␣

С помощью команд \htopin и \vtopin точка в текущей позиции соединяется линией, идущей в горизонтальном или вертикальном направлении, с точкой, x- или y-координата которой соответствует положению вывода, указываемого символическим идентификатором *pinref*. Например, если резистор R2 был ранее определен с помощью команды \R, то последовательность команд

```
\vtopin R2r  \htopin R2r
```

нарисует линию, начинающуюся в текущей точке и идущую вверх до y-координаты резистора R2, а затем по горизонтали до правого вывода этого резистора.

\at x␣y␣ \from x␣y␣
\moverel␣x␣y␣ \shift␣x␣y␣

Текущее положение точки может быть изменено с помощью нескольких команд. Первые две указанные выше команды устанавливают точку в положение с абсолютными координатами (x, y) в координатной системе circ (с шагом в 2.5 мм). В общем случае такое абсолютное позиционирование нежелательно, поскольку все диаграммы неявно рисуются относительно точки $(0, 0)$, и введение абсолютных координат нарушает логическую структуру. Лучше исполь-

Таблица 6.12. Некоторые элементы оптических схем.

(половина) тонкой линзы	\SLens		\HSLens	
(половина) толстой линзы	\BLens		\HBLens	
(половина) вогнутой линзы	\VLens		\HVLens	
поляризатор, экран с отверстием	\Polar		\Pinhole	
делитель пучка	\BSplit			
камера	\Cam			
экраны (взгляд со стороны)	\ScrL		\ScrR	
экраны (взгляд сквозь)	\ScrTL		\ScrTR	
лазер, фотоумножитель	\Laser		\PM	

6-6-3

зовать команды \moverel (или \shift), которые осуществляют относительное перемещение текущей точки на x единиц вправо и на y единиц вверх.

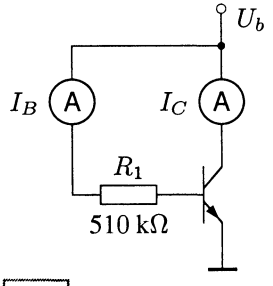
\atpin␣*pinref*␣ \frompin␣{*pinref*}␣

Вы можете перемещать текущую точку с помощью команд \atpin или \frompin. После выполнения этих команд рисование продолжается с указанной точки. Например, с помощью команды \atpin␣L4d␣ текущая точка устанавливается на нижний вывод катушки индуктивности, помеченный меткой L4.

Описанные выше команды демонстрируют лишь некоторые из возможностей, описанных в Tannert and Tille (1996). В пакете circ также имеются команды для центрирования объектов по горизонтали или вертикали по отношению к другим объектам, команды для добавления текста в текущей точке и несколько других приемов, большая часть из которых проиллюстрирована в примерах, приведенных ниже.

6.6.2 Примеры

Наш первый пример заимствован из справочного руководства к пакету circ, в котором подробно описано, как генерируется этот рисунок. На этом рисунке показана схема для измерения коэффициента усиления npn-транзистора по току.



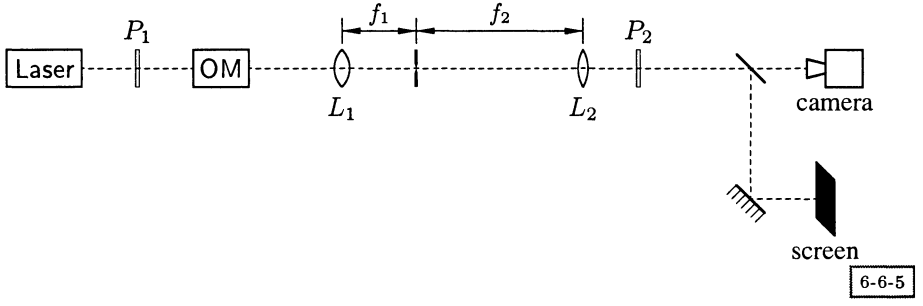
6-6-4

```
\usepackage [basic]{circ}
\begin{circuit}{0}
\npn1 {?} B l     % транзистор в центре
\frompin npn1C    % рисовать начинаем с коллектора
\- 1 u            % небольшая линия вверх
\nl\A1 {$I_C$} v  % измеритель тока коллектора
\atpin npn1B      % следующую линию начинаем с базы
\- 1 l            % небольшая линия налево
\<\R1 {510 k$\Omega$} h  % резистор в цепи базы
\- 1 l            % еще одна небольшая линия налево
\centerto A1      % выравниваем по положению первого
                  % амперметра
\nl\A2 {$I_B$} v  % и рисуем второй амперметр
\frompin A2d      % подсоединяем второй амперметр
\vtopin R1l       % к резистору
\frompin A1u
\- 1 u \.1        % определяем точку соединения
\frompin A2u      % соединяем второй амперметр
                  % с точкой~1
\vtopin .1
\htopin .1
\- 1 u \cc\connection1
      {$U_b$} u   % соединяем с источником питания
\frompin npn1E    % соединяем эмиттер с землей
\- 1 d  \GND1     % сам вывод земли
\end{circuit}
```

Второй пример также взят из руководства. Это — экспериментальная оптическая установка с лазером, модулятором, линзами, зеркалами, камерой и экраном:

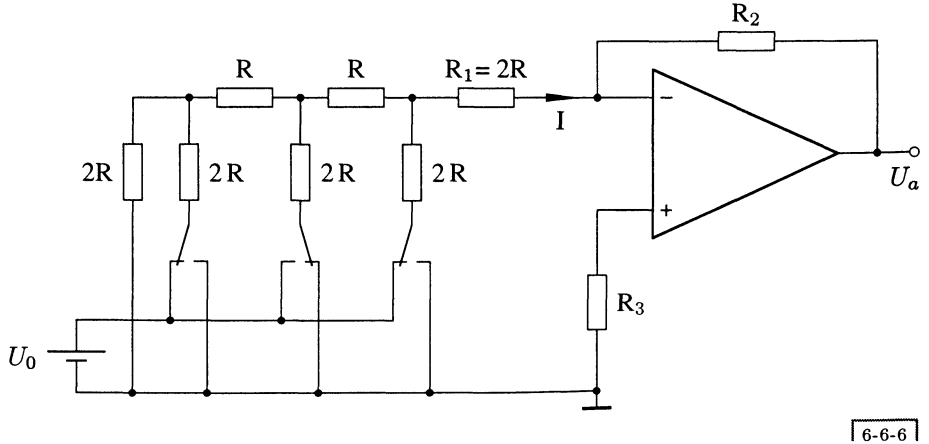
```
\usepackage [basic,optics]{circ}
\begin{circuit}{0}
   \nl\Laser1 {} h                  % лазер
   \oa 2 r                          % оптическая ось
   \Polar1 {} h                     % поляризатор
   \oa 2 r
   \nl\OM1 P1 {} {} {} {} h         % оптический модулятор
   \atpin OM1P3
   \oa 4 r
   \cc\BLens1 {} h                  % первая линза
   \oa 2 r
   \nl\Pinhole1 {} h                % экран с отверстием
   \oa 7 r
   \cc\SLens2 {} h                  % вторая линза
   \oa 1 r
   \Polar2 {} h                     % анализатор
   \oa 3 r
```

```
   \oa 3 r
   \nl\BSplit1 {} + d                  % делитель пучка
   \atpin BSplit1+                     % продолжаем рисовать направо
   \oa 2 r
   \nl\Cam1 {camera} l                 % камера
   \atpin BLens1: \shift 0 2 \P3       % пометить середину первой линзы
   \atpin Pinhole1: \shift 0 2 \P4     % пометить середину экрана
   \atpin SLens2: \shift 0 2 \P5       % пометить середину второй линзы
   \Dtext{\small $f_1$} from P3 to P4 % f_1
   \Dtext{\small $f_2$} from P4 to P5 % f_2
   \atpin BSplit1.                     % вторая часть пучка вниз
   \oa 5 d
   \Mirror1 {} * R                     % зеркало
   \oa 2 r
   \nl\ScrL1 {screen} h                % белый экран
\end{circuit}
```

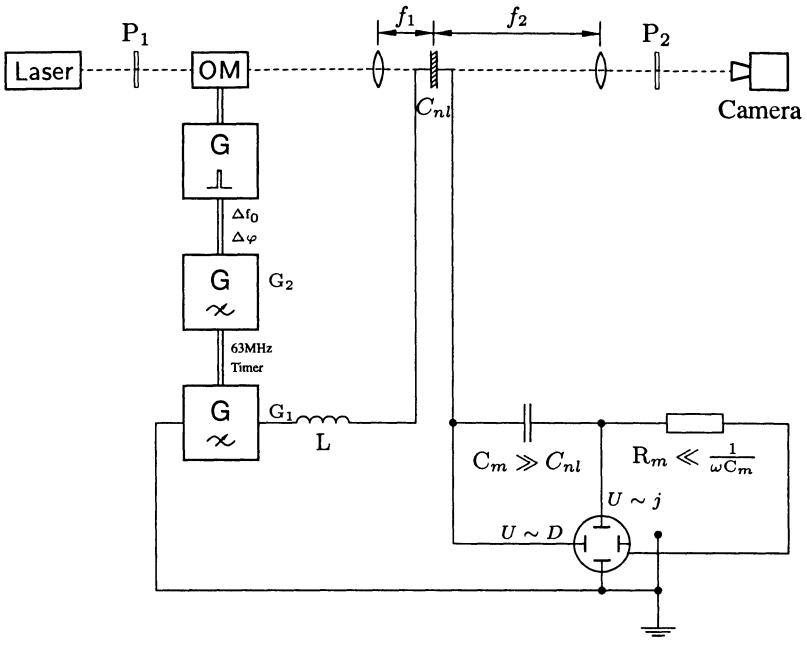


6-6-5

Ниже приведены без кода два более сложных примера, любезно предоставленных нам авторами пакета. На первом изображена схема с операционным усилителем, а на втором — вариант разобранной нами выше схемы. В этой экспериментальной оптической установке измеряются характеристики образца с использованием импульсного и опорного генераторов, LC-цепи, осциллографа, а также поляризатора, анализатора и камеры.



6-6-6



6-6-7

6.7 Использование макропроцессора m4 для рисования электронных диаграмм

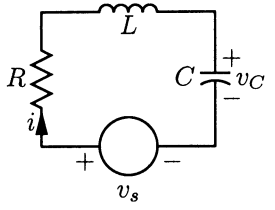
В основу пакета circuit_macros, написанного Двайтом Аплевичем, положен еще один способ подготовки рисунков, который совершенно независим от TEX'a и METAFONT'a. Этот подход использует набор макро для макропроцессора m4, используемого в системе Unix. Эти макро генерируют код на языке pic, который затем обрабатывается программой-интерпретатором pic (gpic) и создает входной файл для TEX'a в виде tpic \specials, которые описаны в приложении А.2. Автор также разработал специальный интерпретатор tpic — программу dpic, которая может генерировать выходной файл для mfpic и PSTricks.[1]

6.7.1 Основные принципы

В пакете имеются несколько библиотек m4, каждая из которых содержит набор базовых макро, позволяющих конструировать сложные электронные схемы и блок-схемы. При рисовании электрических и электронных схем используются обозначения согласно стандарту IEEE Standard 315 1975 г.

[1]Программа dpic не входит в состав пакета, но вы можете получить ее со страницы автора http://ece/uwaterloo/ca/~aplevich/dpic/.

Пакет позволяет сравнительно легко строить достаточно сложные диаграммы (дистрибутив содержит тексты отлаженных сложных примеров, помещенных в конце Maclenan and Burns (1991)). Здесь мы просто покажем, как строится простая схема. Код на языке m4 нашего первого примера выглядит так:




```
.PS
cct_init
define('dimen_',0.6)
loopwid = 0.9; loopht = 0.7
  source(left_ loopwid); llabel(-,v_s,+)
  resistor(up_ loopht); llabel(,R,); b_current(i)
  inductor(right_ loopwid,L); rlabel(,L,)
  capacitor(down_ loopht,C); llabel(+,v_C,-); rlabel(,C,)
.PE
\usebox{\graph}
```

6-7-1

Каждый рисунок заключается в макро `.PS` и `.PE`, которые являются обычными ограничителями в программе pic. Первая макрокоманда `cct_init` инициализирует некоторые внутренние переменные в пакете рисования электрических схем (глобальные переменные, такие, как ширина линии, размер страницы, масштаб). По умолчанию ширина рисунка считается равной половине дюйма (12.7 мм). На следующих двух строках устанавливается ширина основных элементов (`dimen_`) и, поскольку мы хотим нарисовать замкнутую схему, мы указываем ее ширину (`loopwid`) и высоту (`loopht`). После этого мы рисуем источник тока и, пометив его знаками плюс, минус и меткой v_s, начинаем движение влево. Рисуем резистор (двигаясь вверх), помечаем его R и добавляем стрелку для обозначения направления тока в точке i. После этого добавляем катушку индуктивности в точке L и замыкаем цепь конденсатором в точке C, у которого указываем напряжение V_C и полярность зарядов на пластинах. Обратите внимание на то, что для размещения текста слева и справа от элементов схемы были использованы команды `llabel` и `rlabel`.

Для того чтобы использовать этот рисунок в LaTeX'е, нам надо сделать еще несколько преобразований. Во-первых, нам надо перевести инструкции с языка m4 в код pic, запустив m4 и использовав библиотеку `libcct.m4`, содержащую определения основных используемых элементов. Полученный файл затем пропускается через интерпретатор языка pic (например, через gpic), чтобы создать конструкции `\special` для tpic. Последние могут уже восприниматься TeX'ом и окончательно интерпретируются программой, читающей файлы dvi (например, dvips). Последовательность команд, выполняемая для этого на компьютере с операционной системой Unix, имеет примерно следующий вид (она зависит от того, где установлены файлы m4):

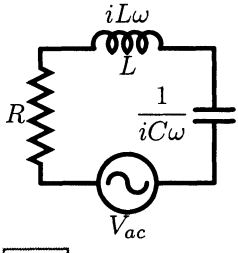
```
m4 /usr/local/lib/m4/libcct.m4 cirexa.m4 > cirexa.pic
gpic -t cirexa.pic > cirexa.tex
```

На выходе мы получаем TeX'овский файл `cirexa.tex`, содержащий только код tpic для рассматриваемого примера. Для дальнейшей его обработки нам надо включить его в файл LaTeX'а с помощью команды `\input`. При этом рисунок

будет помещен в бокс под именем \graph, так что мы должны добавить его вызов \usebox{\graph} в том месте документа, где мы хотим, чтобы рисунок появился.

6.7.2 Модификация диаграмм

Чтобы показать, насколько гибким является подход, использующий circuit_macros, немного модифицируем наш пример и покажем, как схема ведет себя на переменном токе.

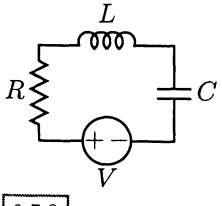



```
.PS
cct_init
linethick=1.6
define('dimen_',0.6)
loopwid = 0.9; loopht = 0.7
  source(left_ loopwid,AC); llabel(,V_{ac},)
  resistor(up_ loopht,5); llabel(,R,)
  inductor(right_ loopwid,W); rlabel(,L,); llabel(,iL\omega,)
  capacitor(down_ loopht,); llabel(,C,)
                    rlabel(,\displaystyle\frac{1}{iC\omega},)
.PE
\usebox{\graph}
```

6-7-2

После того как мы дали указание использовать толстые линии, нарисуем источник переменного тока. Сделаем резистор немного крупнее и укажем комплексные значения для импедансов катушки индуктивности и конденсатора. Обратите внимание на то, как с помощью команд llabel и rlabel текст размещается с обеих сторон элемента. Поскольку текст в метках набирается в математическом режиме, вы можете спокойно использовать математические символы и другие команды, специфичные для математического режима (такие, как \displaystyle для увеличения размеров букв в числителе и знаменателе выражения, показанного рядом с конденсатором).

Некоторые авторы предпочитают рисовать свои электрические схемы, используя сетку. Мы можем написать макро для m4 под названием grid, который имеет два параметра $1 и $2, определяющие x- и y-координаты каждого элемента, который надо нарисовать:



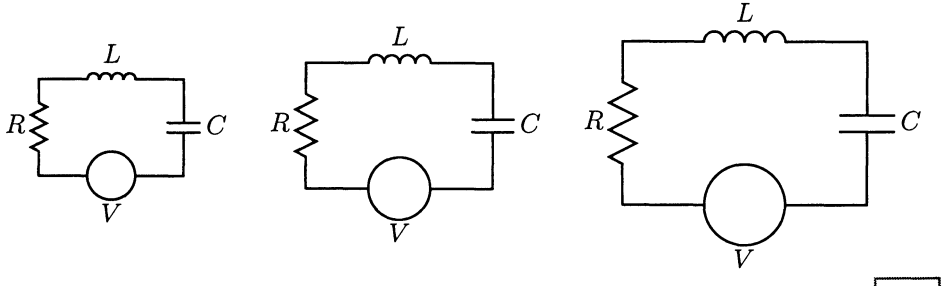

```
.PS
cct_init

gridsize = 0.1
define('grid',`(gridsize*`$1',gridsize*`$2')')

  source(left_ from grid(7,0) to grid(0,0),V); llabel(,V,)
  resistor(up_ from grid(0,0) to grid(0,5),4); llabel(,R,)
  inductor(right_ from grid(0,5) to grid(7,5),W); llabel(,L,)
  capacitor(down_ from grid(7,5) to grid(7,0)); llabel(,C,)
.PE
\usebox{\graph}
```

6-7-3

В качестве последнего примера (другие достаточно сложные примеры можно найти в дистрибутиве пакета) рассмотрим программирование созда-

ния контуров на языке pic. Приведенный ниже пример также показывает, как масштабировать рисунок, выражая величину смещения и размеры элементов в единицах длины `dimen_` (которая определяет размер тела двухвыводных элементов типа резистора, конденсатора и т. п.) и длины `elen_` (длина элемента по умолчанию). Обе эти длины определяются параметром `linewid`, величина которого по умолчанию равна 0.5 дюйма. Как показывает пример ниже, вариация последнего параметра отражается в изменении обеих величин, `dimen_` (по умолчанию равной `linewid`) и `elen_` (по умолчанию равной 1.5 `dimen_`):

```
.PS
cct_init

define('loop','[
  source(left_ elen_); llabel(,V,)
  resistor(up_ dimen_); llabel(,R,)
  inductor(right_ elen_); llabel(,L,)
  capacitor(down_ dimen_); llabel(,C,)
  ]')

for linewid = 0.5 to 1 by  *1.3 do { loop; move right }
.PE
\usebox{\graph}
```



6-7-4

Подготовка партитур

Задача подготовки партитур чрезвычайно сложна, поскольку нотная запись может содержать огромную информацию о структуре и исполнении музыкального произведения. При чтении партитуры музыканты должны получать информацию, которая необходима им для исполнения, например информация о высоте и длине нот, ритме, выразительности и т. д. В зависимости от инструмента нотная запись может располагаться более чем на одном нотном стане (например, на трех или более в случае органа), поэтому объем данных, обрабатываемых одновременно, огромен. Конечно, в этой связи предъявляются большие требования к способностям музыканта, особенно когда идет чтение с листа. Качество шрифта имеет для игры большое значение: шрифт может помочь музыканту, четко выявив структуру произведения, или сделать исполнение произведения с листа практически невозможным.

Высокое качество набора нотных изданий требует наметанного глаза и большого опыта. До недавнего времени эта работа выполнялась высококлассными наборщиками нот, которым за день, по словам Элен Ванске (Wanske, 1990), удавалось набрать не более одной-двух страниц. Как и при наборе текста, критерием высокого качества является общий вид страницы, особенно распределение черного и белого. Безусловно, существуют правила набора особых нагромождений нот, но они не так важны, как общий эстетический подход. В последние годы было разработано несколько компьютерных систем для набора партитур, но все же пока они не могут заменить опытного наборщика нот. Все, чем эти системы способствуют высококачественному набору, — это создание черновика, который можно использовать для дальнейшего его совершенствования. Они вносят свой вклад в создание высококачественных партитур только в том случае, если принятие эстетического решения остается за *опытным* пользователем.

В этой главе будут рассмотрены система MusiXTEX и некоторые программы предварительной обработки. MusiXTEX — это система TEX'овских макро, представляющая собой мощный инструмент для набора партитур. Стоит подчеркнуть, что система MusiXTEX не принимает решений — они остаются за наборщиком. Поэтому этой системой достаточно сложно пользоваться, и для облегчения взаимодействия с ней были разработаны программы предварительной обработки.

7.1 Использование TEX'а для набора партитур — общий обзор

Первые попытки использовать TEX для подготовки партитур предпринимались Андреа Штайнбахом и Анжеликой Шофер (Schofer and Steinbach, 1987), а позднее Франсуа Жальбером, который разработал MuTEX (Jalbert, 1989). Однако лишь Даниель Топен совершил настоящий прорыв, создав пакет MusicTEX (Taupin, 1992, 1993a, 1995a,b).

Главной целью Топена при разработке пакета MusicTEX был набор сложной полифонической оркестровой или инструментальной музыки. Сначала он думал, что может расширить MuTEX до нескольких нотных станов, но вскоре решил написать набор совершенно новых макро, только лишь приняв MuTEX'овский код на METAFONT'е за отправную точку. Однако полученная система (MusicTEX) по-прежнему была неадекватна, так как ее однократного прохода было недостаточно для расчета оптимального расстояния между нотами. MusiXTEX, новая система, разработанная Даниелем Топеном совместно с Россом Митчеллом и Андреасом Эглером, взяла свое начало от MusicTEX. Она совместима с ранним пакетом, но в ней использован трехкратный проход для оптимизации лиг и расстояния между нотными знаками[1]. Поскольку принципы и команды для написания исходного текста во многом идентичны, различия в нотной записи, созданной пакетами MusicTEX и MusiXTEX, обычно незначительны для рядового пользователя и бóльшая часть предлагаемого далее описания может быть отнесена к обеим системам.

Помимо команд для манипулирования большим количеством инструментов, для различных ключей, нот, аккордов, перекладин, лиг, украшений и т. д., в MusiXTEX входят несколько библиотек-расширений с командами, которые реже встречаются или которые требуются лишь для особых видов музыки. Библиотеки-расширения охватывают такие темы, как:

- 128-е ноты;
- особые знаки для хоровой музыки;
- Григорианский распев;
- гитарные табулятуры;
- струнные инструменты.

Система MusiXTEX не предназначена для того, чтобы переводить стандартную нотную запись в TEX, и не делает попыток учитывать эстетические соображения. Система набирает станы, ноты, аккорды, перекладины, лиги и украшения, дословно следуя указаниям наборщика (т. е. вашим). В руках специалиста MusiXTEX становится очень гибким и мощным инструментом, помогающим достичь очень хороших результатов.

Из-за того что MusiXTEX достаточно сложен, было разработано несколько программ предварительной обработки для облегчения взаимодействия с ним.

[1] Андреас Эглер сейчас работает над собственной системой, которая будет *несовместима* с MusicTEX'ом, но приобретет много дополнительных возможностей (например, знаки легато с точками, лиги и другие элементы, создаваемые средствами PostScript'а).

Они, конечно, относятся только к части возможностей MusiXTEX'a. У каждой из них есть свои приоритеты и язык ввода. Программа midi2tex (разд. 7.5), например, превращает midi-файлы в код MusicTEX'a и, следовательно, представляет интерес для тех, кто предпочитает записи нот игру на музыкальных инструментах.

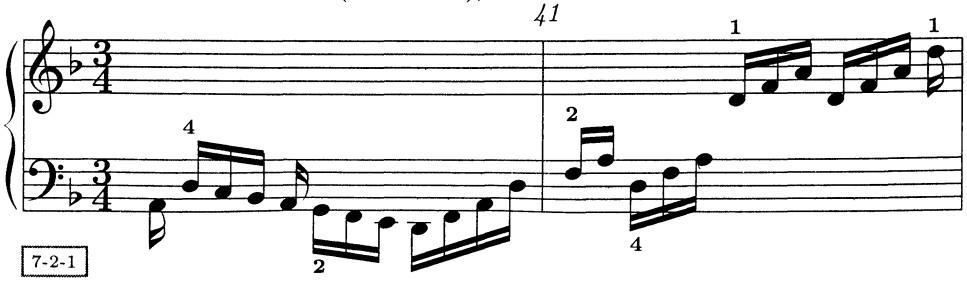
Существуют, по крайней мере, три программы предварительной обработки, имеющие текстовый ввод. Программа abc2mtex (разд. 7.3) использует в качестве языка ввода язык музыкальной записи abc в формате ASCII. Этот язык, изначально разработанный для записи мелодий, интуитивно очевиден при чтении и написании. В особенности он подходит для тех случаев, когда нужен только один стан, но может справиться и с записью гитарных аккордов над станом. В MPP (разд. 7.4) использован собственный достаточно простой для чтения язык ввода. Возможно, вы отдадите предпочтение MPP, если записываете музыку на нескольких нотоносцах, поскольку это даст вам возможность разместить их в разных файлах. Программа PMX, созданная Доном Саймонсом (Simons, 1995), поддерживает еще один текстовый язык ввода со встроенной возможностью записывать оцифрованный бас.

В следующем разделе мы опишем основные концепции написания ввода для MusiXTEX'a. Поскольку MusiXTEX — очень большая система, мы ограничимся лишь некоторыми ее чертами. Для получения подробной информации следует обращаться к полному комплекту документации (Taupin et al., 1996). В последних трех разделах данной главы будет рассказано, как пользоваться abc2mtex, MPP и midi2tex. Если вы собираетесь использовать только одну из программ предварительной обработки, то можете пропустить следующий раздел при первом прочтении этой книги, так как он не требуется для понимания программ предварительной обработки. Позднее вам следует прочесть, как минимум, разд. 7.2.5 для того, чтобы научиться пользоваться MusiXTEX'ом, ведь он потребуется вам в любом случае.

7.2 Использование MusiXTEX'a

7.2.1 Структура исходного MusiXTEX'овского текста

Партитура музыкального произведения, например такты 40–41 прелюдии Иоганна Себастьяна Баха (BWV 926),



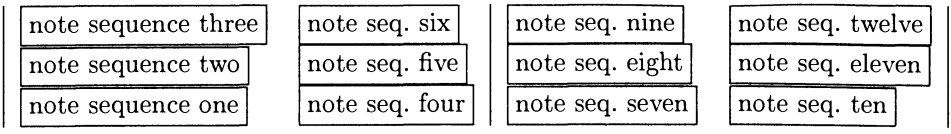


Рис. 7.1. Порядок рассмотрения стана MusiXTEX'ом.

содержит важную информацию, которую можно извлечь, зная только контекст:

- высоту ноты: это нельзя определить только по положению ноты по отношению к линейкам нотного стана — она зависит еще и от ключа и знаков альтерации, относящихся к ноте. Ключ обычно стоит в начале линеек, но может появиться в любом другом месте нотоносца. Учитывая знаки альтерации, вы должны различать случайные знаки альтерации, относящиеся только к данному такту, и знаки, относящиеся к большей части произведения; последние стоят в начале такта или в одном из предыдущих тактов.
- момент, в который должна звучать нота: это обычно зависит от последовательности других нот и пауз в данном такте, но может быть неочевидна, если голос переходит с нотоносца на нотоносец, как в приведенном выше примере.

Входные данные MusiXTEX'а указывают на специфику структуры и информацию, заложенную в партитуре. Но партитура произведения с несколькими нотоносцами имеет две плоскости, ось x, представляющую время и поделенную на такты, и ось y, используемую для нотных станов. По мнению Топена,

> «музыкант читает или обдумывает несколько последовательно идущих нот (обычно длинную долю или группу логически связанных нот), после этого он переходит к [...] следующему инструменту или голосу и в конце концов собирает все воедино для того, чтобы создать музыкальный отрывок продолжительностью в несколько секунд. Затем он переходит к следующему такту или части партитуры».

Исходя из этой теории, Топен воплотил синтаксис ввода, описывающий маленькие отрывки партитуры, начиная с некоторых нот на нижнем стане и затем продвигаясь вверх (см. рис. 7.1). Естественность этого синтаксиса является предметом обсуждения в литературе, а некоторые программы предварительной обработки MusiXTEX'а обрабатывают стан в обратном порядке, т. е. сверху вниз.

Ключевая команда в MusiXTEX'е выглядит так:

```
\notes ... & ... & ... \enotes
```

Она описывает один столбец последовательности нот. Знак `&` отделяет последовательность нот, которые должны быть набраны на одном стане и соответствуют разным голосам или инструментам, начиная снизу. Несколько станов

разделены знаком |, например, партитура пьесы для скрипки с фортепиано будет записана таким образом:

```
\notes ... | ... & ... \enotes
```

для каждой колонки *групп нот*, рассматриваемых как логическое целое, не просто аккорды из нескольких нот, которые надо играть вместе, но также небольшие последовательности нот, составляющих музыкальную фразу. Поэтому основная часть исходного текста MusiXTEX'a задается командами типа `\notes...\enotes`. За одной или несколькими из этих команд идет команда `\bar`, означающая конец такта. MusiXTEX имеет несколько вариантов форм, например `\Notes`, `\NOtes` (см. табл. 7.2 на с. 307), которые отличаются только распределением расстояний между отдельными нотами.

7.2.2 Написание нот

В MusiXTEX'е есть много команд для набора элементов партитуры, являющихся основополагающими для работы с этой системой. (В табл. 7.1 представлены наиболее часто используемые знаки.) Чтобы показать принцип работы, в этом разделе будут объяснены некоторые основные команды для написания нот.

Определение высоты ноты

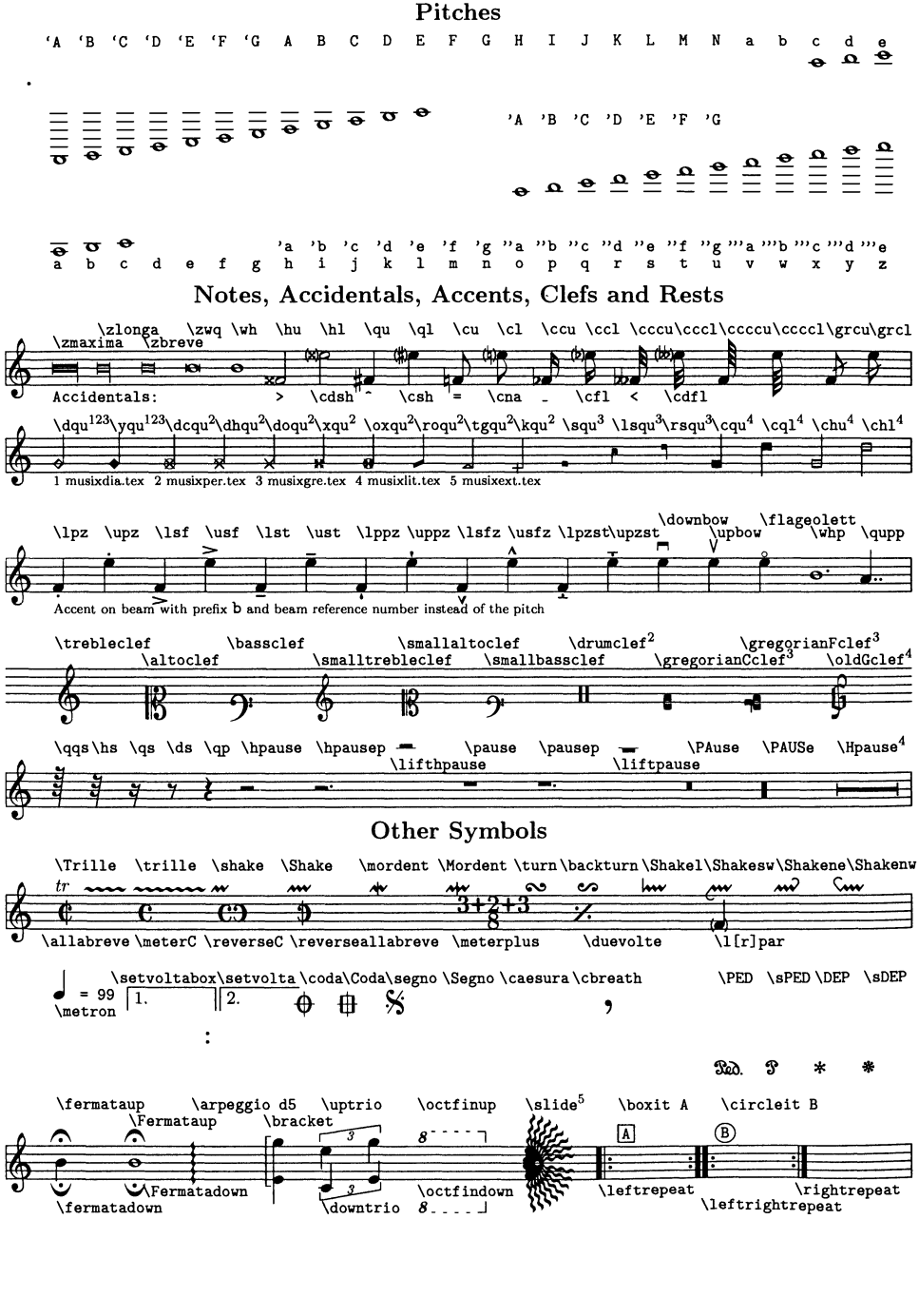
Высота нот обозначается буквами (см. табл. 7.1 ниже). Важно отметить, что это определение не зависит от ключа. Это — отклонение от принципа MusiXTEX'a, состоящего в том, что представление информации о музыкальном произведении обычно очень похоже на то, как она представлена в самой партитуре. Это сделано для того, чтобы можно было изменить ключ, относящийся, скажем, только к нескольким тактам, не меняя все ноты, попадающие в сферу его влияния. Положение нот на стане набираемой партитуры автоматически корректируется относительно нового ключа.

Случайные знаки альтерации должны быть определены отдельно, и к ним должен быть такой же подход, как к самой партитуре. Для того чтобы поставить бемоль или диез перед определенной нотой, скажем нотой `c`, вы пишете `\fla c`, или `\sh c`, или просто `_c`, или `^c` соответственно. Для определения знака при ключе, относящегося ко всему произведению или его части, используется команда `\generalsignature`, в которой в качестве аргумента стоит цифра с плюсом в случае бемолей и с минусом — в случае диезов.

Нотные символы

Для того чтобы построить ноту, определение высоты звука надо дополнить командой, указывающей на длительность ноты и направление ее штиля. Наи-

Таблица 7.1. Обзор MusiXTEX'овских команд (подготовлено Даниелем Топеном) (pitches — высота; notes — длительность, accidentals — альтерация, accents — акценты, clefs — ключи, rests — паузы; other symbols — другие символы).



более часто использующиеся команды для записи нот с указанием единичного пробела после ноты выглядят так:

`\wh` p : целая нота на высоте p;

`\hu` p : половинная нота на высоте p со штилем вверх;

`\hl` p : половинная нота на высоте p со штилем вниз;

`\ha` p : половинная нота на высоте p; направление штиля выбирается автоматически в соответствии с обычаем в изображении мелодий: если нота располагается ниже третьей линейки стана, штиль направлен вверх, в прочих случаях он направлен вниз;

`\qu` p : четвертная нота на высоте p со штилем вверх;

`\ql` p : четвертная нота на высоте p со штилем вниз;

`\qa` p : четвертная нота на высоте p с автоматически выбранным направлением штиля;

`\cu` p : одна восьмая на высоте p со штилем вверх (буква `c` в этой и следующих командах идет от французского «croche», что означает восьмая доля; `cc`, `ccc` означает «double croche» «triple croche» и т. д.);

`\cl` p : одна восьмая на высоте p со штилем вниз;

`\ca` p : одна восьмая на высоте p с автоматически выбранным направлением штиля;

`\ccu` p : одна шестнадцатая на высоте p со штилем вверх;

`\ccl` p : одна шестнадцатая на высоте p со штилем вниз;

`\cca` p : одна шестнадцатая на высоте p с автоматически выбранным направлением штиля;

`\cccu` p : одна 32-я на высоте p со штилем вверх;

`\cccl` p : одна 32-я на высоте p со штилем вниз;

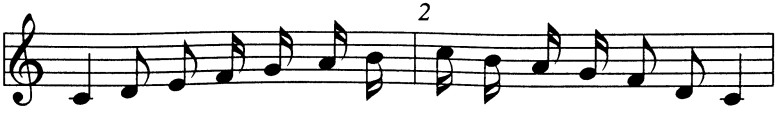
`\ccccu` p : одна 64-я на высоте p со штилем вверх;

`\ccccl` p : одна 64-я на высоте p со штилем вниз.

С помощью этих команд можно набрать первый простой пример:

```
\usepackage {musixtex}
\begin{music}
\startextract
\Notes\qu c\cu d\cu e\ccu f\ccu g \ccu h\ccl i\enotes\bar
\Notes\cca j\cca i\cca h\cca g\ca f\ca d\qa c\enotes
\endextract
\end{music}
```

7-2-2

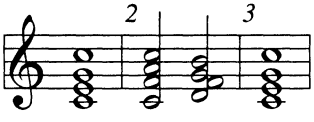


Поскольку мы используем LaTeX, все должно быть размещено внутри окружения `music` (в формате plain TeX обработка выполняется другими средствами). Кодировка нот начинается с команды `\startextract` и заканчивается `\endextract`. Если вы пишете целую пьесу, то используйте вместо этого `\startpiece` и `\endpiece`.

Для получения четверти с точкой на высоте c вы можете написать `\qu{.c}` или воспользоваться командой `\pt` и написать `\pt␣c\qu␣c`. Для пауз используются следующие команды: `\pause` для целой паузы, `\hpause` для половинной паузы, `\qp` для четвертинной паузы и `\ds` для восьмой паузы.

Однако приведенных выше команд для написания нот недостаточно. Для того чтобы набирать аккорды, вам потребуются команды, которые *не* добавляют после себя пространства. И еще, чтобы набирать аккорды с очень маленькими интервалами, вы должны уметь размещать тело ноты справа или слева от штиля. Для этого есть набор беспробельных команд, подобных командам, приведенным выше, начинающихся с `\z...`, и ряд команд для смещения тела ноты вправо или влево, начинающихся, соответственно, с `\r...` или `\l...`.

При наборе аккордов используйте беспробельные команды для всех нот в аккорде, кроме последней. Если последняя нота со штилем, то у аккорда пишется один штиль, который объединяет все ноты, звучащие одновременно:




```
\usepackage {musixtex}
\begin{music}
\startextract
\Notes\zw c\zw e\zw g\wh j\enotes\bar
\Notes\zh c\zh f\zh h\hu j\zh d\rh f%
   \zh g\hu i\enotes\bar
\Notes\zw c\zw e\zw g\wh j\enotes
\endextract
\end{music}
```

7-2-3

7.2.3 Расстояние между нотами

Размещение нот элегантным образом — задача сложная. В колонке нот расстояние между нотами не обязательно постоянно, поскольку в идеале оно зависит от длительности самой короткой из тех нот, что играются одновременно. Правда, легко найти исключения из правила, так как расстояние зависит не только от данных нот, но и от контекста, по крайней мере в том же такте.

Вот почему в MusiXTeX'е используется многоступенчатый подход к определению расстояния. Первый шаг — определение расстояния в исходном тексте. Для этого используется последовательность `\notes ... \enotes`. У этой команды есть разновидности; все они действуют одинаково, но помещают разный промежуток (`\noteskip`) между последовательно идущими нотами (см. табл. 7.2). В каждом из этих вариантов все набираемые расстояния изначально одинаковые. С помощью команды `\sk` вы «можете вставить» дополнительное пространство шириной `\noteskip` между нотами, идущими одна за другой.

Таблица 7.2. Варианты форм команды \notes.

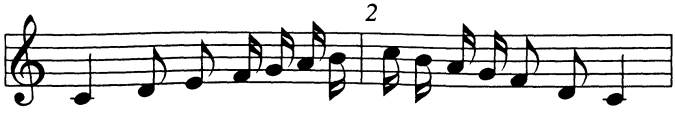
Ноты	Расстояние	Рекомендации по применению
\znotes ... & ... & ... \enotes	0 \elemskip	специальная
\notes ... & ... & ... \enotes	2.0 \elemskip	16-е
\notesp ... & ... & ... \enotes	2.5 \elemskip	16-е с точкой
\Notes ... & ... & ... \enotes	3.0 \elemskip	8-е
\Notesp ... & ... & ... \enotes	3.5 \elemskip	8-е с точкой
\NOtes ... & ... & ... \enotes	4.0 \elemskip	четверти
\NOtesp ... & ... & ... \enotes	4.5 \elemskip	четверти с точкой
\NOTes ... & ... & ... \enotes	5.0 \elemskip	половинные
\NOTesp ... & ... & ... \enotes	5.5 \elemskip	половинные с точкой
\NOTEs ... & ... & ... \enotes	6.0 \elemskip	целые

По умолчанию размер \noteskip увеличивается линейно от \notes до \NOTEs. Это может быть изменено командой \geometricskipscale на геометрическую прогрессию, для которой размер в \Notes в $\sqrt{2}$ раз больше по сравнению с \notes, и т. д. На самом деле, и основная единица расстояния (\elemskip), и расстояние между отдельными нотами (\noteskip) могут свободно регулироваться, как это объяснено в Taupin et al. (1996).

Второй шаг осуществляется программой musixflx, которая берет величину пробела \elemskip и вычисляет для каждой линейки новую величину, используемую при наборе. Это делается для того, чтобы тактовая черта проходила через все линейки и чтобы линейка не прерывалась в середине такта. В разд. 7.2.5 этот вопрос будет рассмотрен более подробно.

Чтобы проиллюстрировать, как работают команды, определяющие расстояние, приведем уже знакомый нам пример, но уже с правильными промежутками между знаками:

```
\usepackage {musixtex}
\begin{music}
\startextract
\NOtes\qu c\enotes  \Notes\cu d\cu e\enotes
\notes\ccu f\ccu g\ccu h\ccl i\enotes  \bar
\notes\cca j\cca i\cca h\cca g\enotes
\Notes\ca f\ca d\enotes  \NOtes\qa c\enotes
\endextract
\end{music}
```



7-2-4

7.2.4 Сравнительно полный пример

Чтобы представить еще несколько понятий MusiXTEX'a, покажем первые четыре такта фортепианного произведения «Schweinehirtenlied» Бела Бартока. Начало кода содержит некую информацию заголовка; если такая информация отсутствует, как в предыдущих примерах, то используются значения по умолчанию.

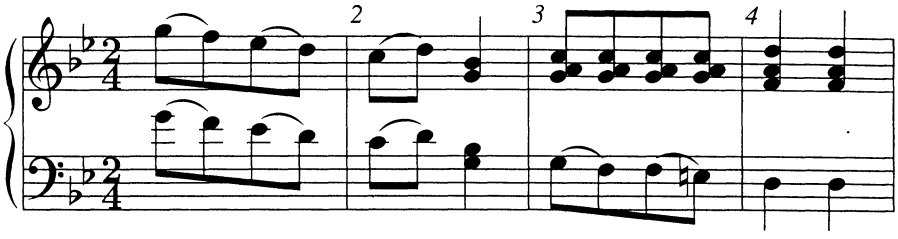
Заголовок, вероятно, должен содержать команду `\instrumentnumber`, которой задается число исполняющих произведение инструментов. Инструменты нумеруются снизу вверх. Инструмент, партия которого занимает более одного нотоносца, отмечается командой `\setstaffs`.

Доступны два размера шрифта: 20pt на нотоносец или 16pt на нотоносец; для последнего в заголовке добавляется `\smallmusicsize`.

Традиционно пользователи MusiXTEX'a по возможности опускают фигурные скобки по обе стороны аргументов команд, например они пишут `\qb0g` вместо `\qb{0}{g}`. Для упрощения понимания нашего примера оставим фигурные скобки вокруг аргументов, обозначающих высоту звуков в командах. Поскольку это произведение предназначено для фортепиано, нотоносцы принадлежат одному инструменту, поэтому для отделения групп нот внутри команд `\notes` использован не `&`, а `|`:

```
\usepackage {musixtex}
\begin{music}
\instrumentnumber{1}
\setstaffs{1}{2}
\generalmeter{\meterfrac24}
\setclef{1}{\bass}
\setclef{2}{\treble}
\generalsignature{-2}
\startextract
\Notes
  \isluru0g\ibl0e{-2}\qb0{g}\tslur0f\qb0{f}%
  \isluru0e\qb0{e}\tslur0{d}\tbl0\qb0{d}%
 |\isluru0n\ibl0m{-2}\qb0{n}\tslur0m\qb0{m}%
  \isluru0l\qb0{l}\tbl0\tslur0k\qb0{k}%
\enotes\bar
\Notes
  \isluru0c\ibl0c0\qb0{c}\tslur0d\tbl0\qb0{d}%
 |\isluru0j\ibl0j0\qb0{j}\tslur0k\tbl0\qb0{k}%
\enotes
\NOtes \zq{N}\ql{b}|\zq{g}\qu{i}\enotes\bar
\Notes
  \isluru0N\ibl0N{-2}\qb0{N}\tslur0M\qb0{M}%
  \isluru0M\qb0{M}\tslur0L\tbl0\qb0{=L}%
 |\ibu0j0\zqb0{g}\rq{h}\qb0{j}\zqb0{g}\rq{h}\qb0{j}%
  \zqb0{g}\rq{h}\qb0{j}\tbu0\zqb0{g}\rq{h}\qb0{j}%
\enotes\bar
```

```
\NOtes \ql{K}\ql{K}|\zq{f}\zq{h}\qu{k}\zq{f}\zq{h}\qu{k}\enotes
\endextract
\end{music}
```



7-2-5

Текст программы для этой партитуры выглядит достаточно сложно. Это объясняется наличием лиг и перекладин. Для каждой перекладины или лиги добавляется две команды, одна в начале (которая должна стоять перед первой нотой, к которой она относится) и одна в конце (которая должна стоять перед последней нотой, к которой она относится).

Для лиг самая распространенная открывающая команда — `\isluru` *np*, если лига стоит над нотами, и `\islurd` *np*, если под ними. Два параметра этой команды определяют номер лиги[1] и высоту звука, к которому относится начало лиги. Соответствующей замыкающей командой является `\tslur`, аргументами которой служат номер лиги и высота звука, на котором эта лига заканчивается.

Часто используемые открывающие команды для перекладин — `\ibu` для перекладин над нотами и `\ibl` для перекладин под ними. Для создания двойных, тройных и четверных «ребер» просто увеличьте число символов `\b` в команде, например `\ibbu` для двойной перекладины над нотами, и т. д. Существуют команды для повторяющихся рисунков перекладин и полуавтоматически создаваемые перекладины, у которых вычисляется угол наклона. Есть возможность даже печатать перекладины поверх тактовой черты. У команд для создания перекладин есть три параметра: номер, высота звука, от которого начинается соединяющая черта, и наклон — целое число в диапазоне $[-9, 9]$, которое задает угол наклона от -45% до 45%. Перекладина ставится тремя линейками выше или ниже ноты, к которой она относится. Для обозначения конца соединительной черты, соответственно, используются команды `\tbu` или `\tbl`, в которых в качестве аргумента указывается только номер перекладины.

Для создания перекладины недостаточно только открывающей и закрывающей команд, так как к ней должны быть присоединены ноты, которые она охватывает. Для этого используется команда `\qb`, действующая аналогично `\qu` или `\ql`, но имеющая дополнительный параметр, описывающий номер перекладины, к которой присоединен штиль.

[1] Обычно номер лиги равен 0, если в такте не больше одной параллельно исполняемой лиги.

7.2.5 Запуск MusiXTEX'а

Как было сказано ранее, MusiXTEX возник на основе MusicTEX'а для упрощения создания «красивых партитур». MusicTEX — однопроходная система, тогда как MusiXTEX — трехпроходная система.

На первом проходе через LaTeX или plain TeX[1] прогоняется файл, содержащий исходный MusiXTEX'овский текст. В результате получается файл с расширением `mx1`, который содержит всю необходимую информацию об этом музыкальном произведении, например расстояние между нотами и т. д. На втором проходе внешняя программа `musixflx` вычисляет оптимальное расстояние между нотами. Эта программа считывает `mx1`-файл и записывает новый файл с расширением `mx2`, который содержит соответствующие настройки размера `\elemskip` для каждой линейки нотоносца. На третьем проходе файл с исходным текстом опять обрабатывается (LA)TEX'ом. На этот раз MusiXTEX находит соответствующий `mx2`-файл и использует содержащуюся в нем информацию.

На первом проходе MusiXTEX может найти относящийся к предыдущему запуску старый `mx2`-файл, который содержит неправильную информацию. В таком случае получаемый `mx1`-файл тоже неверен, и, следовательно, результат третьего прохода тоже будет плохим. Поэтому после внесения изменений в исходный файл или при получении сообщения о непонятной ошибке или неожиданного результата удалите все вспомогательные файлы и начните заново.

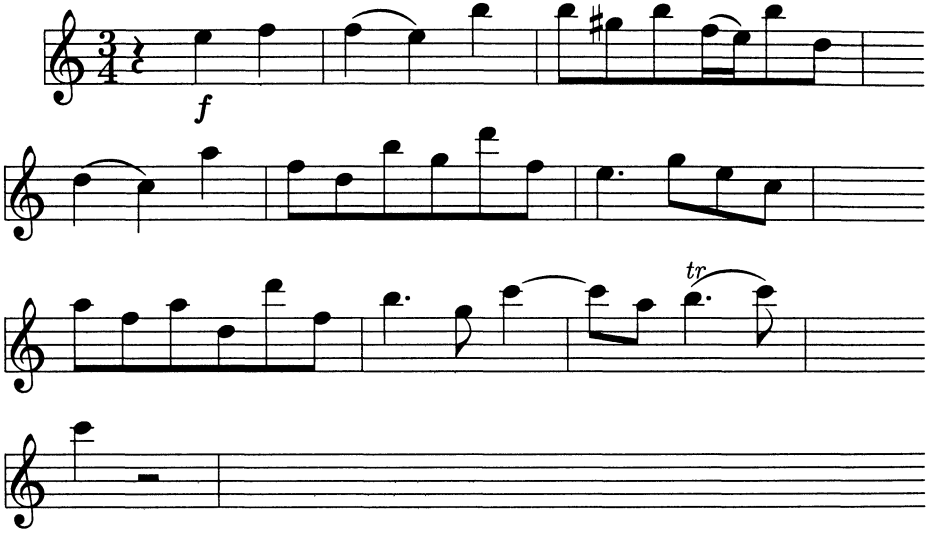
В качестве небольшого примера полной последовательности проанализируйте следующий исходный MusiXTEX'овский текст:

```
\documentclass{article}
\usepackage {musixtex}
\pagestyle{empty}
\begin{document} \begin{music}
\instrumentnumber{1}  \generalmeter{\meterfrac34}  \nobarnumbers
\startpiece
\NOtes\qp\zcharnote N\f\ql{lm}\enotes\bar                                    %1
\NOtes\isluru0m\ql{m}\tslur0l\ql{l}\ql{p}\enotes\bar                         %2
\Notes\ibl0l0\qb0{p^np}\enotes
  \notes\nbbl0\isluru0m\qb0{m}\tbl0\tslur0l\qb0{l}\enotes
  \Notes\qb0p\tbl0\qb0k\enotes\bar                                           %3
\NOtes\isluru0k\ql{k}\tslur0j\ql{j}\ql{o}\enotes\bar                         %4
\Notes\ibl0l0\qb0{mkpnr}\tbl0\qb0m\enotes\bar                                %5
\NOtesp\qlp{l}\enotes\Notes\ibl0n{-4}\qb0{nl}\tbl0\qb0j\enotes\bar           %6
\Notes\ibl0l0\qb0{omokr}\tbl0\qb0m\enotes\bar                                %7
\NOtesp\qlp{p}\enotes\Notes\cl{n}\enotes\NOtes\itenu0r\ql{q}\enotes\bar%8
\Notes\ibl0q{-3}\tten0\qb0q\tbl0\qb0o\enotes
  \NOtesp\Uptext{\it tr}\isluru0p\qlp{p}\enotes
```

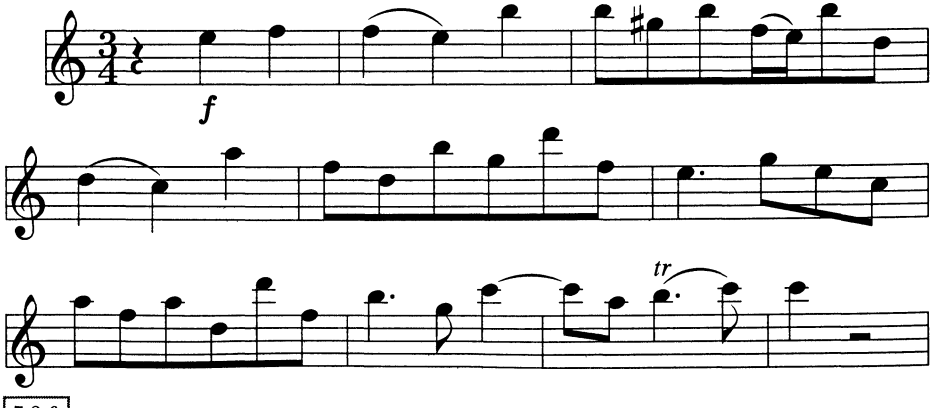
[1] Какой вариант TeX'а использовать, зависит от содержания файла. В примерах этого раздела предполагается использование LaTeX'овского интерфейса; это разумный выбор, если музыка будет перемешана с текстом. Однако если надо создать только партитуру, то дополнительные функциональные возможности LaTeX'а вряд ли понадобятся, поэтому большинство программ предварительной обработки порождают файлы, которыми можно манипулировать с помощью TeX'а.

```
  \Notes\tslur0q\cl{q}\enotes\bar                                    %9
\NOtes\ql{q}\enotes \NOTes\hpause\enotes
\endpiece
\end{music} \end{document}
```

Его первая обработка LaTeX'ом дает результат, при котором на всех линейках располагаются равные (и, следовательно, неправильные) промежутки между знаками:



Если после этого полученный `mx1`-файл будет обработан программой musixflx, а затем еще раз обработан LaTeX'ом, то в результате мы получим шедевр (первые такты Соната Сеста для блок-флейты и баса Франческо Мариа Верачини):



7-2-6

7.3 abc2mtex: записать мелодию — это легко

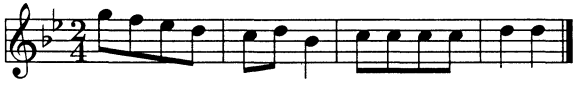
Если вы хотите записывать мелодии, возможно, с несколькими аккордами в качестве аккомпанемента, вам поможет MusiXTEX'овская программа предварительной обработки abc2mtex, написанная Крисом Уолшо (Walshaw, 1996). Эта система конвертирует созданный Уолшо музыкальный нотный язык abc в MusiXTEX'овский текст. С программой abc2mtex вам совсем не надо заботиться о расположении нот. Вы просто вводите музыкальную информацию в нотации abc, которая проста и удобна. Потом программа сама использует внутренние установки по умолчанию[1] для создания MusiXTEX'овского кода.

Музыкальная нотация abc была разработана в понятном для компьютера ASCII-формате, который люди могут легко прочесть и даже сыграть. Изначально она была предназначена для записи народных и традиционных мелодий Западной Европы, которые могут быть записаны на одном нотоносце. Этот вид записи нашел широкое распространение с момента своего появления в 1991 г. В Интернете появилось несколько коллекций мелодий, записанных на языке abc. Около дюжины программ используют abc, например, для конвертации abc в MIDI-файлы и обратно, или для воспроизведения музыки непосредственно через колонки компьютера.

В этой книге описана версия abc2mtex 1.6. Над программой все еще идет активная работа и читателю рекомендуется пользоваться самой последней версией, поскольку некоторые возможности, описанные далее, отсутствовали в предыдущих версиях.

7.3.1 Написание исходного текста для abc2mtex

Для того чтобы посмотреть, как строится исходный текст для abc2mtex, проанализируйте следующий пример:



```
M:2/4
L:1/8
K:Gm
gfed | cd B2 | cccc | d2 d2 |]
```

7-3-1

Исходный текст abc2mtex для мелодии состоит из двух частей: заголовка и тела. Заголовок содержит информационные поля, каждое из которых начинается с заглавной буквы, обозначающей тип информации, и следующего за ней двоеточия. Тело состоит из самого музыкального произведения. В тело можно вставлять дополнительные информационные поля, с помощью которых может меняться информация в заголовке (например, ключ, размер или темп).

[1] Эти установки хранятся в файле `header.tex` и при желании могут быть изменены.

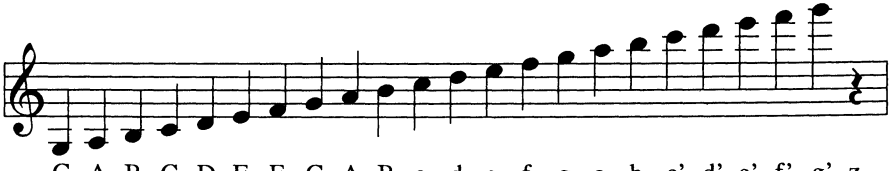
Всего существует 26 информационных полей, большинство из которых являются необязательными[1] Вот лишь наиболее общие из них:

- Музыкальная информация:
  - `K:` ключ. Вы можете использовать мажорную (например, `K:E`) и минорную (`K:g`) тональности, миксолидийский (`K:AMix`) и дорийский (`K:EDor`) лады;
  - `M:` размер, например, `M:3/4`, `M:C` или `M:C|`.
- Техническая информация:
  - `E:` значение `\elemskip` в MusiXTEX'е, т. е. величина, определяющая расстояние между нотами в конечном тексте;
  - `X:` номер мелодии в файле; обязателен, если в файле содержится больше одной мелодии.
- Общая информация, например `T:` название (обязательно), `C:` композитор, `D:` дискография, `H:` история, `O:` происхождение, `S:` источник.

Порядок полей не так важен, однако `T:` должно стоять первым (или вторым, если есть поле `X:`), а за ним должно следовать `K:`, так как оно обозначает начало тела (и, значит, тоже является обязательным).

Возвратившись к вышеприведенному примеру, вы легко можете найти *тактовые черты*, кодируемые символом `|`. *Двойные черты* вводятся с помощью `||`, `[|` или `|]`, а *повторы* получаются с помощью `:|`, `::` или `|:`. Первый и второй повторы получаются благодаря использованию `[1` и `[2`.

Для каждой *ноты* вы пишете букву, обозначающую ее в музыке[2] Заглавные буквы используются для низких нот, строчные буквы — для более высоких нот. Правая кавычка (`'`) справа от строчной буквы означает следующую (более высокую) октаву, а запятая у заглавных букв означает предыдущую (более низкую) октаву. Это тоже хорошо знакомо музыкантам. Для обозначения паузы используется буква `z`.



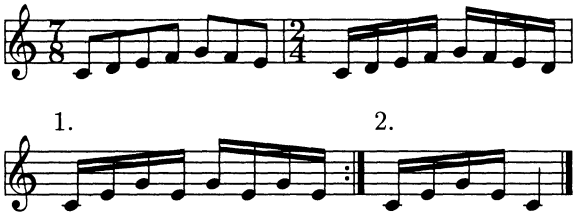
G, A, B, C D E F G A B c d e f g a b c' d' e' f' g' z

7-3-2

[1] Их полный перечень можно найти в Walshaw (1996).

[2] Здесь используется широко распространенная в Западной Европе система буквенного обозначения высоты звуков. Следует заметить, что звук, обозначаемый буквой `b`, в действительности соответствует *си бемоль*, а для обозначения звука *си* традиционно используется буква `h`. Это соглашение *нарушается* в программах MusiXTEX и abc2mtex. — *Прим. ред.*

Как же указывается *длительность нот*? Существует принятое по умолчанию значение длительности ноты (1/16 или 1/8), которое зависит от указанного размера. Одна шестнадцатая используется, если размер менее 0.75, а одна восьмая — если больше, поэтому в случае размера 4/4 принятая по умолчанию длительность ноты составляет одну восьмую. Эта принятая по умолчанию длительность ноты может быть изменена в информационном поле `L:`. Следующий пример иллюстрирует автоматическое изменение принятой по умолчанию длительности, когда размер меняется с 7/8 на 2/4:

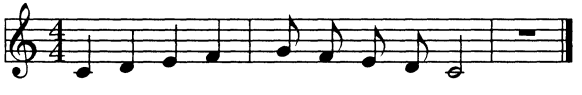



```
M:7/8
K:C
CDEF GFE  | \
M:2/4
CDEF GFED |[1 CEGE GEGE :|
          [2 CEGE C4 |]
```

7-3-3

По умолчанию каждая строка исходного текста порождает одну строку партитуры. Этого можно избежать, если завершать строку символом \, как в предыдущем примере.

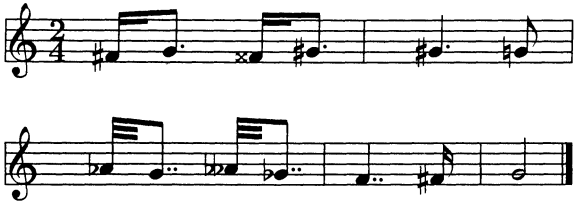
Ноты или паузы разной длительности получаются благодаря множителю, который ставится после обозначающей ноту буквы. Например, если размер равен 4/4, то длительность `G1` (или `G1`) равна 1/8, длительность `G2` равна 1/4, а длительность `G/2` равна 1/16.



```
M:4/4
L:1/4
K:C
C D E F |
G/2 F/2 E/2 D/2 C2| z4 |]
```

7-3-4

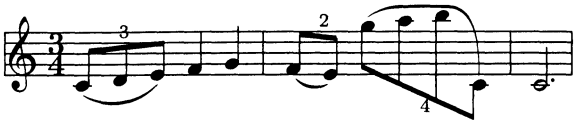
Знаки альтерации получаются с помощью знаков ^ (диез), = (бекар) или _ (бемоль), а дубль-диез и дубль-бемоль — с помощью ^^ и __, которые ставятся перед нотой. Есть также и легко читаемые соглашения для обозначения ритма с точками:



```
M:2/4
L:1/8
K:C
^F<G ^^F<^G | ^G2>=G2 |
_A<<G __A<<_G | F2>>^F2 |
G4 |]
```

7-3-5

и для дуолей, триолей и тому подобных элементов:



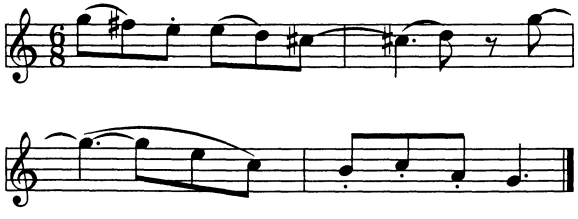

```
M:3/4
K:C
(3CDE  F2 G2 | (2FE
(4gabC | C6 |
```

7-3-6

Как уже говорилось, когда вы пишете исходный текст для abc2mtex'а, вы не заботитесь о внешнем виде макета, поскольку программа пользуется собственными принятыми по умолчанию настройками. Однако вы можете использовать некоторые средства для оказания локального воздействия на его внешний вид, как то:

- Информационное поле `E:` может быть использовано для задания или изменения MusiXTEX'овского параметра `\elemskip`.
- Между нотами появляется *перекладина*, если буквы исходного текста набраны без пробелов. Высота и наклон перекладины устанавливаются автоматически и не могут быть изменены.
- *Лиги* получаются, если ноты, которые она связывает, заключены в скобки (без пробела после открывающей и перед закрывающей скобками). *Связки* между двумя нотами (слигованные ноты) возникают при использовании знака `-`; он действует даже при переходе через тактовую черту.

В следующем примере показаны перекладины, лиги и связки, а также знаки стаккато. Последние появляются, если перед нотой поставить точку:

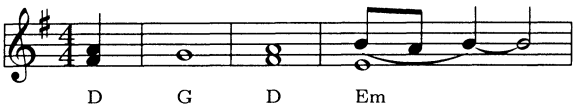


```
M:6/8
K:C
(g^f).e (ed)^c-|
(^c3 d) z g-|
(g3-gec) | .B.c.A G3 |]
```

7-3-7

Аккорды кодируются с помощью квадратных скобок. Они могут быть сгруппированы перекладинами, но между ними не должно быть пробелов

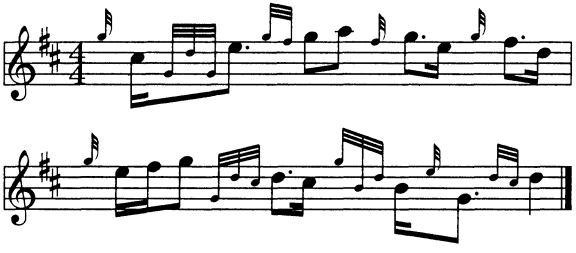
Гитарные аккорды могут быть указаны под нотоносцем мелодии; их кодируют, помещая между двойными кавычками:




```
M:4/4
L:1/4
K:G
"D"[FA] |"G"G4 |
"D"[F4A4] | \
"Em"[(B/2E4]A/2 B)- B2|
```

7-3-8

Помимо этого, можно набрать *форшлаги*, заключив ноты в фигурные скобки. Обычно эти ноты не имеют длительности, поэтому выражения типа {c3} или {c>d} не имеют смысла. Следующий пример, взятый из мелодии «Athol Brose», показывает сложные мелизмы волынки во всей их красе:



```
M:4/4
K:D
{g}c<{GdG}e {gf}ga
   {f}g>e {g}f>d| \
{g}e/f/g {Gdc}d>c
   {gBd}B<{e}G {dc}d2|]
```

7-3-9

Программа abc2mtex может быть использована и для произведений, написанных больше чем для одного голоса, хотя, по мнению автора, «это не следует рассматривать как поддерживаемое свойство».

7.3.2 Работа с abc2mtex

Имеются версии abc2mtex для Unix и DOS. Для DOS exe-файл прилагается в дистрибутиве, а пользователям Unix придется компилировать его из исходного кода на C.

Программа может быть вызвана без аргументов, и в этом случае она интерактивно задает вопросы. Ее можно вызвать и другим способом:

abc2mtex *options tunefile*:*selection*

В этом случае одна или несколько мелодий или музыкальных произведений, содержащихся в *tunefile*.abc, конвертируется в MusiXTEX'овский текст и записывается в файл music.tex. Потом этот файл может быть обработан MusiXTEX'ом, как это было объяснено в разд. 7.2.5, для получения окончательно напечатанной партитуры. Параметр *selection* позволяет указать только те мелодии, которые вам нужны; знак - означает все мелодии. Вы можете указать

номера мелодий через запятую или диапазонами; например, если вам нужны пьеса 1, пьесы с 3 по 5 и с 7 и далее, то *selection* будет `1,3-5,7-`. В качестве *options*:

`-x` создать выход для MusiXTEX'а; без этой опции создается выход для MusicTEX'а;

`-o` *name* имя выходного файла, если оно отличается от `music.tex`;

`-t` *number* переносит пьесу в другой ключ в процессе ее обработки;

`-i` создает перечень всех музыкальных произведений в файле.

7.4 MPP — MusiXTEX'овская программа предварительной обработки

Ян Ньивенхёйзен разработал MusiXTEX'овскую программу предварительной обработки MPP (Nieuwenhuizen and Nienhuys, 1996) для облегчения работы с MusiXTEX'ом. Он охарактеризовал ее таким образом:

> MPP — очень простая программа по сравнению с MusiXTEX'ом, но потенциально она может сделать MusiXTEX намного более *приемлемым*, если не сказать *используемым*. Ее задача состоит не в том, чтобы охватить все возможности MusiXTEX'а, а в том, чтобы существенно облегчить набор простых музыкальных произведений (или частей).

Концепции этой программы во многом близки концепциям abc, и результирующий код тоже похож. Одно большое отличие заключается в способе кодирования нескольких нотоносцев. В MusiXTEX'е музыкальное произведение делится на очень мелкие куски, часто меньше одного такта. Эти куски последовательно кодируются, начиная с нот на нижнем нотоносце и потом переходя к следующим повыше. Именно поэтому очень сложно читать код. Для MPP используется другой метод манипулирования двумя измерениями партитуры: для каждого стана пишется свой файл, а потом MPP соединяет их для того, чтобы получить MusiXTEX'овский исходный текст.

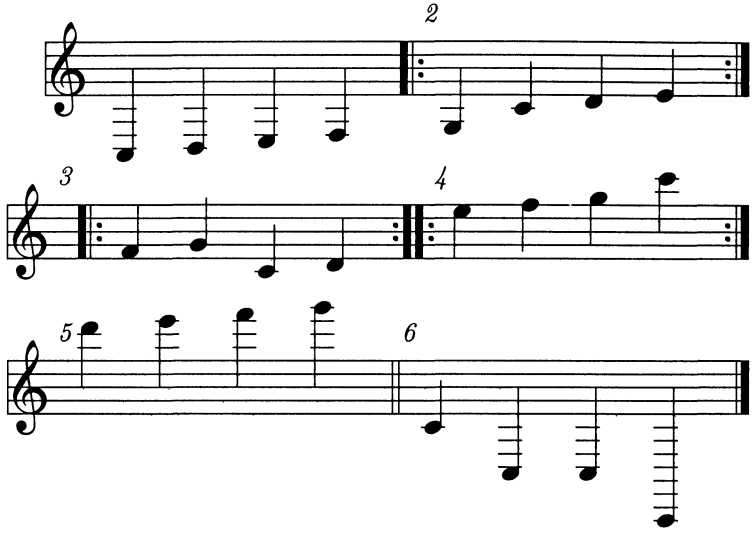
7.4.1 Написание исходного текста MPP

Для набора одной ноты MPP надо задать четыре параметра: *октаву*, *название ноты*, *длительность* и *характеристики*. Из них обязательным является только *название ноты*. Вся эта информация кодируется в заданном порядке без пробелов; пробел разделяет две ноты.

Ноты обозначаются обычными буквами: строчные буквы используются для верхних октав, а прописные — для нижних октав. Апостроф слева направо (`‘`) перед буквой переносит ноту на одну октаву ниже, а апостроф справа налево (`’`) повышает ее на одну октаву. Для создания разных ключей используется команда `\clef`, за которой следуют `\treble`, `\alto`, `\tenor` или `\bass`. В

следующем примере показаны некоторые варианты высот и то, как создаются разные виды *тактовых черт.*

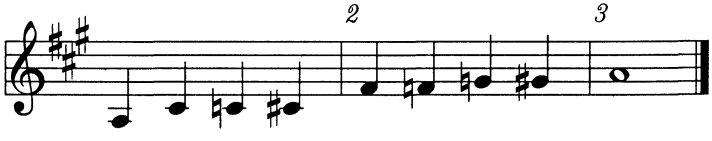
```
C D E F |: G c d e :||: f g c d :|: 'e 'f 'g ''c :|
''d ''e ''f ''g || c `c C `C |||
```



7-4-1

В отличие от abc2mtex, случайные знаки альтерации обозначаются дополнительными *названиями нот*, например `gis` или `bes`, а не символом перед основной нотой.[1] Когда вы записываете музыку с помощью MPP, вы задаете тональность произведения командой `\key`. Для отдельной ноты вам надо лишь написать правильное название ноты, а MPP сам расставит соответствующие случайные знаки альтерации (в соответствии с выбранной вами тональностью):

```
\key\A
A cis c cis | fis f g gis | a1 |||
```



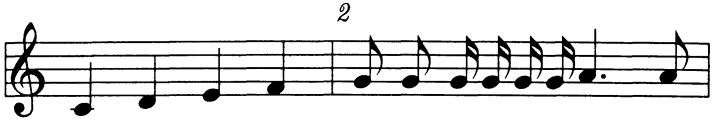
7-4-2

В *длительности* половинки, четверти, восьмые, 16-е, 32-е и 64-е задаются соответствующими номерами после названия нот. Для нот с точками точка

[1] Названия случайных знаков альтерации взяты из немецко-голландской системы обозначений: суффикс **«is»** обозначает ноту с диезом, например **«cis»** значит «c♯» (*до диез*), а суффикс **«es»** обозначает ноту с бемолем, например **«ges»** — это «g♭» (*соль бемоль*).

указывается после этого номера. Принимаемая по умолчанию длительность ноты может задаваться командой \duration, которая потом используется для всех нот, длительность которых не указана. Если \duration не указан, то принятая по умолчанию длительность ноты равна одной четверти.

```
\duration8   | c4 d4 e4 f4 |  g g g16 g16 g16 g16 a4. a |.
```



7-4-3

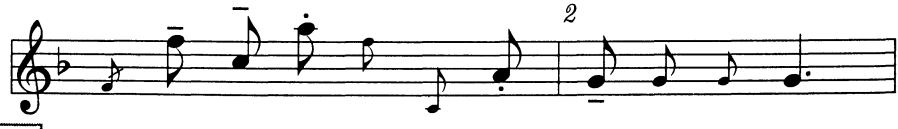
Такое представление длительности нот имеет свое ограничение, так как вы не можете представлять дуоли, триоли, и т. д. естественным образом. Однако, как будет показано ниже, их можно описать, используя механизм группировки нот.

Прочая информация, такая, как детали набора и акценты или, в более общем виде, нижние или верхние индексы, может быть добавлена в конце кода ноты. Нижние и верхние индексы добавляются способом, уже знакомым пользователям LaTeX'а, с помощью _ и ^. В параметре нижнего или верхнего индекса может содержаться часть текста или *макро*; в настоящее время поддерживаются \tr (трель), \upbow и \downbow.

Для манипуляций с набором нот MPP предлагает помимо всего прочего следующие команды:

- \stemdown, \stemup, \stemno: для определения вида штиля;
- \headleft, \headright: для определения, с какой стороны от штиля поместить ноту;
- \grace: для создания форшлага;
- \normal, \small, \tiny: для установления размера следующих нот.

```
\key\F
\duration8
f\grace 'f^{-} 'c\stemup^{-} 'a^{.} \tiny 'f  c \normal a_{.} |
 g_{-} \small g \tiny g \normal g4. |.
```

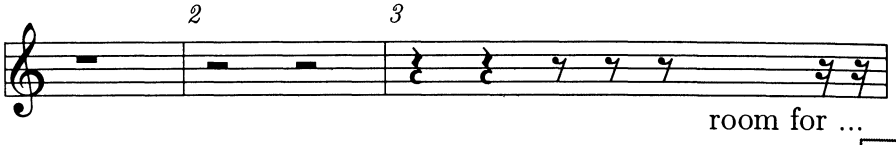


7-4-4

Иногда у вас может возникнуть желание оставить немного свободного места на нотном стане для примечания. Этого можно добиться, вставив название знака s (что означает *script*) в ваш исходный текст. Со знаком s обращаются так же, как и с нотой, поскольку у него есть длительность (которая определяет

порождаемое им расстояние) и могут быть верхние и нижние индексы. Для введения *паузы* наберите «ноту», обозначенную `r`.

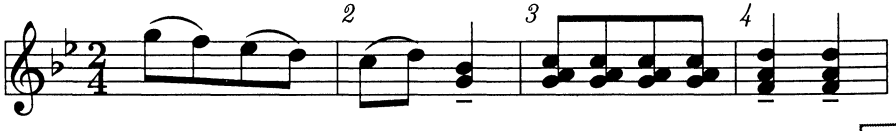
```
r1 | r2 r2 | r r r8 r8 r8 s1_{room for ...} r16 r16 |.
```



7-4-5

Перекладина получается путем заключения всех нот, объединяемых ею, в квадратные скобки, например `[c8 d8 e8 f8]`. Для *аккордов* используются фигурные скобки, а для *лиг* — обычные. Открывающая скобка для лиги пишется после первой ноты, а закрывающая — перед последней нотой, например, для того, чтобы создать лигу, включающую ноты `c d e f`, надо написать `c( d e )f`. Обычно выбор направления штилей внутри такта оставляется на усмотрение программы, но его можно задать в явном виде командой `[^...]` или `[v...]`. Аналогичная ситуация с лигами, положение которых может быть скорректировано добавлением к одной или обеим скобкам знаков `+` или `-`.

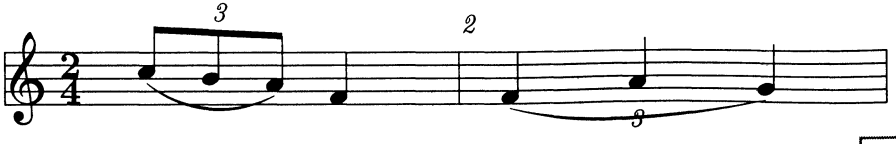
```
\key\Bes
\duration8
\meter{2/4}

['g( )'f 'es( )'d ] | ['c( )'d ] {g4_{-} bes4} | [{g a\headright 'c}
{g a\headright 'c} {g a\headright 'c} {g a\headright 'c}] | {f4_{-} a4
'd4} {f4_{-} a4 'd4} |.
```



7-4-6

Для создания дуолей, триолей, и т. д. с перекладинами поставьте ноты, принадлежащие к x-олю в скобки и добавьте `/x` после закрывающей скобки. x-оли без соединительных «ребер» написать сложно, что и иллюстрирует следующий пример:

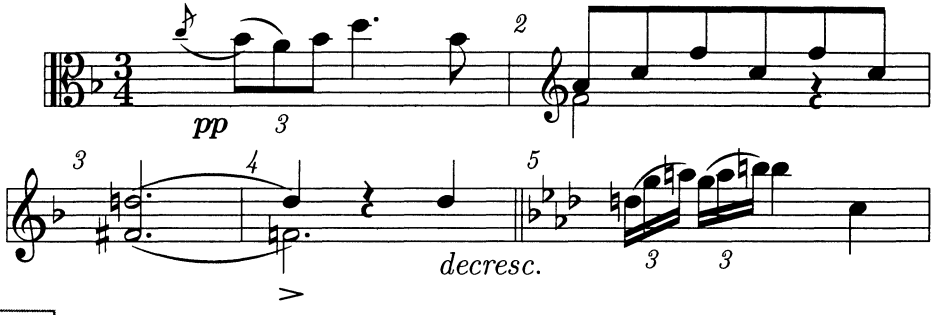
```
\meter{2/4}  \duration{8}
 ['c( b )a]/3  f4 |
 f4/3( a4/3_{{\xpletstyle 3}} ) g4/3  |.
```



7-4-7

У MPP есть еще несколько других черт; в дистрибутив добавлен ряд иллюстрирующих файлов, которые стоит изучить. Ниже приводится последний пример, демонстрирующий его дополнительные функциональные возможности.

```
\meter{3/4}
\key\F          \clef\alto      \duration8
   'c\grace\stemup(   [) bes_{pp}( ) a bes ]/3 'd4. bes |
\clef\treble
    [^ {  a f2 } 'c 'f 'c { 'f r4 } 'c ] |
    { fis2.(v ='d2.\stemup(^ } |
    { ) 'd4  ) =f2._> } { r4 s4_{decr} } 'd4\stemup \key\As ||
\duration{16}
    [ 'd( 'g )'a ]/3  [ 'g( 'a )'b ]/3  'b4 'c4 |.
```



7-4-8

7.4.2 Работа с MPP

Для работы с MPP нужно сначала создать один или несколько файлов с расширением mpp. Благодаря прогону этих файлов через программу mpp возникает единственный файл с расширением tex, который в дальнейшем может быть обработан MusiXTEX'ом.

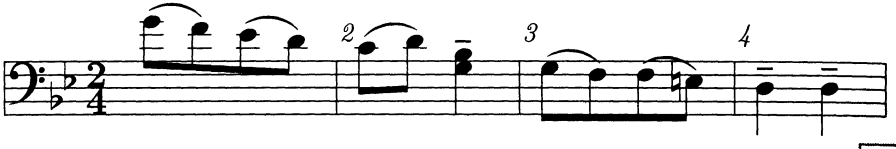
Передаваемым программе mpp аргументом является перечень разделенных пробелами файлов, каждый из которых содержит по одному нотоносцу произведения. Порядок файлов определяет порядок нотных станов в конечном тексте. В отличие от MusiXTEX'а, нотоносцы нумеруются сверху вниз. Если два нотных стана надо объединить, например, для фортепиано, поставьте запятую между названиями соответствующих файлов в командной строке. Таким образом, команда

```
mpp violin1 violin2 pianoR , pianoL
```

объединяет файлы `violin1.mpp`, `violin2.mpp`, `pianoR.mpp` и `pianoL.mpp` в один TEX'овский файл под названием `violin1.tex`, имя которого создается из имени первого вводимого файла. Впоследствии этот файл может быть обработан системой MusiXTEX'а, как объяснялось в разд. 7.2.5.

Наш сравнительно сложный пример из разд. 7.2.4 намного проще записать с помощью MPP. В примере 7-4-6 уже был показан код для верхнего стана. Код нижнего стана имеет следующий вид:

```
%file pianoL.mpp
\key\Bes  \duration8  \meter{2/4}  \clef\bass
['G( )'F 'Es( )'D ] | ['C( )'D ] {G4 Bes4^{-}} |
[G( )F F( )E ] | D4^{-} D4^{-} |.
```

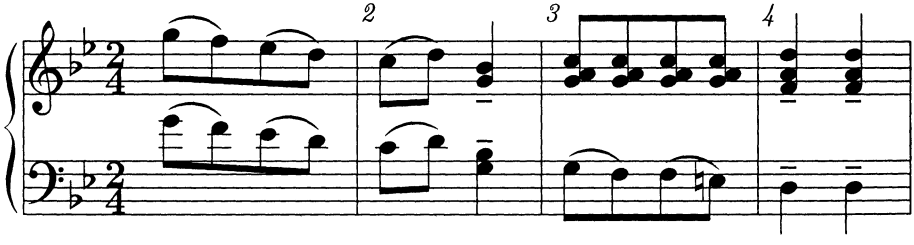


7-4-9

Поэтому если вы выполните команду

```
mpp pianoR , pianoL
```

то получите такой результат:



7-4-10

Возможно, результат кодирования текста этого примера непосредственно в MusiXTEX'е выглядит несколько красивее (см. пример 7-2-5). Расстояния между нотами и перекладинами в первом такте здесь слишком малы, а перекладины с наклоном выглядят лучше. Для получения более высокого качества вы можете просто отредактировать MusiXTEX'овский код, созданный MPP. Это займет значительно меньше времени, чем кодирование всего произведения непосредственно в MusiXTEX'е.

7.5 midi2tex

Х. Дж. П. Куикенс создал другой тип программы предварительной обработки, midi2tex (Kuykens and Verbruggen, 1992), который превращает MIDI-файлы типа 1 в MusicTEX'овский формат. Версии для MusiXTEX'а не существует, но поскольку MusiXTEX обеспечивает режим обратной совместимости, midi2tex тоже может быть использован в сочетании с MusiXTEX'ом, если вы учтете несколько моментов, описанных в Taupin et al. (1996).

MIDI (музыкальный инструментальный цифровой интерфейс) — это международный стандарт подключения электронных музыкальных инструментов к компьютерам. Существует много программ-секвенсоров[1], позволяющих записывать и редактировать MIDI-данные, и их можно использовать для создания MIDI-файла, который вам нужен для набора музыкального произведения с помощью midi2tex.

MIDI-данные состоят из ряда сообщений длиной по два-три байта. Каждое сообщение содержит информацию о команде, например *note on* для обозначения начала ноты или *note off* для обозначения ее конца. В стандарте MIDI определены номера дорожек от 1 до 16, а команды имеют аргументы, указывающие, к примеру, какую ноту взять или снять. MIDI-данные в большой степени ориентированы на клавиатуру, на которой клавиши пронумерованы от 1 до 128. Поэтому для MIDI нет разницы между соль диез и ля бемоль (*gis* и *as*).

7.5.1 Работа с midi2tex

Программа midi2tex существует для систем DOS и Atari. Она вызывается следующей командой:

```
midi2tex inputfile outputfile -option1 -option2 ...
```

Предполагается, что входной файл — это MIDI-файл типа 1. При использовании файлов типа 0 могут возникнуть проблемы; если используется такой файл, то midi2tex выдает предупреждение о том, что результат может быть непредсказуемым, тем не менее Куикенс предлагает пользователям поэкспериментировать с ним.

Одной из проблем с MIDI-файлами, созданными программируемой записывающей клавиатурой или иным MIDI-совместимым инструментом, является то, что расчет времени может быть не очень точным. Для получения хорошей партитуры с верными началами и длительностями нот нужен «чистый» MIDI-файл. Стоит приложить усилия для получения как можно более хорошего MIDI-файла для того, чтобы избежать большого числа ошибок в MusicTEX'овском файле.

Секвенсоры часто предоставляют функции для решения этой задачи. Если это не так, то опция -q программы midi2tex, за которой следует число, может быть использована для *квантирования* результата. Это число задает дительность самой короткой ноты (или паузы) в музыкальной пьесе. Если, например, вы пишете -q8, то в получаемой партитуре не будет нот короче одной восьмой. Могут быть использованы следующие значения: 1 (целая нота), 2 (половинная), 4 (четверть), 8, 16, 32, и 64.

[1] Секвенсорами называют программы, позволяющие зависывать, редактировать и затем воспроизводить MIDI-сообщения, представленные в виде треков (в MIDI-формате типа 1). — *Прим. ред.*

Для отладки имеется опция `-d`. За ней должны следовать либо `SCREEN`, `FILE`, либо `PRINTER`, которые указывают, куда направляется информация.

Большая часть других опций midi2tex позволяет вам определить, сколько в конечном тексте имеется станов, как по ним распределены MIDI-данные, какая тональность или ключ использованы и т. д.

Есть группа опций, которые указывают, как в конечном тексте должны появляться *дорожки* MIDI-файлов. Если не указана ни одна из них, то для каждой дорожки создается свой стан. Дорожка 1 по умолчанию опускается, поскольку в MIDI-файлах типа 1 она обычно не используется.

`-o`*tracklist* Меняет порядок дорожек в *tracklist* (перечень дорожек, разделенных запятой). Не забудьте включить все дорожки в перечень, иначе midi2tex остановится из-за ошибки.

`-s`*tracklist* Пропускает дорожки, указанные в *tracklist*.

`-i`*tracklist* Приписывает дорожки, указанные в *tracklist*, одному инструменту (например, правой или левой руке клавиатуры).

Другая группа опций позволяет уточнить *ключи*, используемые на нотоносцах; по умолчанию используется скрипичный ключ:

`-b`*tracklist* Устанавливает басовый ключ для всех дорожек в *tracklist*.

`-a`*key tracklist* Задает альтовый ключ для всех дорожек в *tracklist*, причем *key* — это `1`, `2`, `3` или `4`, обозначающие одно из положений альтового ключа.

`-k`*number* Задает музыкальную *тональность*; *number* должно быть от `-8` до `8` и определяет число диезов (положительные числа) или бемолей (отрицательные числа). Бемоли и диезы вставляются в соответствии с указанным числом.

Помимо этого, к MusicTEX'овскому файлу вы можете добавить текст:

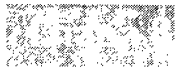
`-t`*tracklist* Добавляет к конечному файлу текст, который берется из отдельного файла с тем же именем, что и MIDI-файл, но с расширением `txt`. В *tracklist* указываются дорожки, в которые будет вставляться текст. Содержимое текстового файла разделяется знаками `|` для того, чтобы различить такты. Текст больше чем для одной дорожки разделяется `/`.

И наконец, есть целый ряд опций для воздействия на MusicTEX'овский набор, в том числе:

`-f` Наклон всех перекладин задает равным 0.

`-e`*dimension* Устанавливает MusicTEX'овский параметр `\elemskip`.

`-m`*number* Задает размер набираемых нот; *number* может быть 16 (16pt) или 20 (20pt).



Играя в игры

Настольные и карточные игры имеют давнюю историю. Тысячи книг на сотнях языков посвящены шахматам, го, картам и т. д. Для объяснения их правил или записи ходов партий в этих книгах почти всегда используются диаграммы. В данной главе мы рассмотрим некоторые примеры, демонстрирующие, как подобные графические объекты можно создать с помощью LaTeX'а.

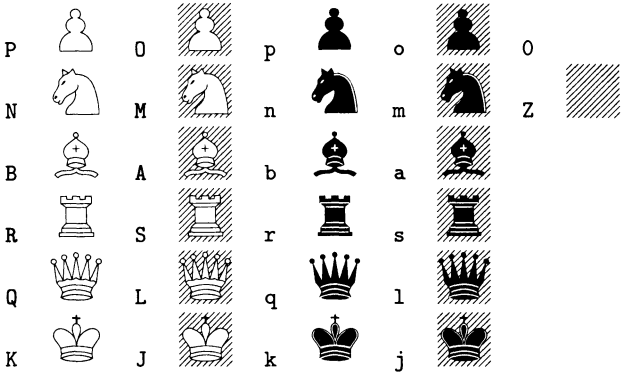
Большинство относящихся к играм пакетов связано либо с предоставлением возможности использования специальных шрифтов для набора необходимых символов, либо специальных макро для создания хороших примеров хода игры. Например, глубоко проработанная шахматная система записи, с которой мы начнем, вполне сочетается с такой алгоритмической системой типографского набора, как LaTeX. Относящиеся к шахматам пакеты отслеживают сделанные ходы и позволяют выводить информацию в разной форме.

Затем мы перейдем к таким довольно близким играм, как китайские шахматы, го и нарды. После этого мы обратимся к карточным играм, для одной из которых — классической игры бридж — существует специальный пакет. Глава завершается в таинственном мире составления кроссвордов. Хотя составление кроссвордов не является игрой, возникающие при этом проблемы набора аналогичны, и применяющие LaTeX составители кроссвордов будут рады прибегнуть к помощи двух нетривиальных пакетов.

8.1 Шахматы

Набор шахматных партий с помощью TeX'а начался в 1988 г., когда Вольфганг Аппельт разработал набор предназначенных для этой цели макрокоманд (Appelt, 1988). Этот первый вариант не содержал специального шрифта с изображениями шахматных фигур, поэтому Аппельт обозначал их такими символами, как Ω. Чуть позже Цальман Рубинштейн (Rubinstein, 1989) создал с помощью METAFONT'а первый простой шахматный шрифт. Почти четыре года спустя голландский шахматист-энтузиаст Пит Тютеларс разработал

Таблица 8.1. Расположение шахматных фигур в шрифтах chess.



шахматный шрифт, символы которого имели вполне классический вид, и соответствующий LaTeX'овский пакет (Tutelaers, 1992), превративший LaTeX в профессиональную систему набора для шахматных журналов.

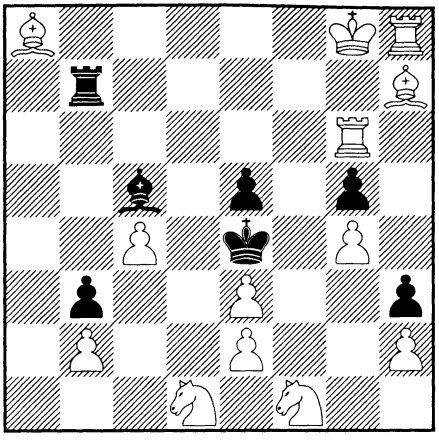
8.1.1 METAFONT'овский шахматный шрифт

Созданный Питом Тютеларсом шахматный шрифт содержит 26 символов: каждая фигура (конь, слон, ладья, ферзь, король) и пешка представлена четырьмя символами, которые соответствуют ее белому и черному варианту, стоящему на обоих типах полей (белом и черном). Кроме того, существуют два символа, изображающих пустые белые и черные поля.

Рамка вокруг доски не является частью шрифта, а добавляется создающими шахматную диаграмму макро (обсуждаемыми в следующем разделе) в виде горизонтальных и вертикальных линеек. В табл. 8.1 показаны все символы, входящие в шрифт `chess20` (сторона поля в этом шрифте равна 20 пунктам). В дистрибутив также входят шрифты размера 10 (`chess10`) и 30 (`chess30`) пунктов, а в случае необходимости можно легко создать другие размеры. Дополнительный шрифт `chessf10` содержит только изображения фигур (короля, ферзя, ладьи, слона, коня) и применяется для сокращенной записи ходов, например строка `|25. N5*g3|` приводит к появлению 25. ♘5×g3.

8.1.2 chess — пакет для шахматного набора

Пит Тютеларс (Tutelaers, 1992) создал пакет chess для облегчения набора шахматных диаграмм и записи партий, частично опираясь на идеи, опубликованные Вольфгангом Аппельтом четырьмя годами ранее. Этот пакет предоставляет команды для описания ходов, отражаемых потом в алгебраической записи партии. Во внутреннем представлении сохраняется информация о текущих позициях всех фигур, и в любой момент с помощью единственной команды можно показать текущее положение фигур на доске.



8-1-1

```
\usepackage {chess}
\board{B* * *KR}
      {*r* * *B}
      { * * *R*}
      {* b p p }
      { *P*k*P*}
      {*p* P *p}
      { P *P* P}
      {* *N*N* }
\[ \showboard \]
```

Рис. 8.1. Шахматная задача: белые делают ход, не дав мат черным.

Указанные особенности наиболее полезны, если записывается ход шахматной партии. В случае шахматных задач автоматический учет сделанных ходов обычно не требуется. Однако для этой цели в пакете chess предусмотрена команда `\board`, восемь аргументов которой обозначают восемь рядов полей шахматной доски. Для получения изображения внутри всех аргументов используется простая нотация: пустые белые поля обозначаются пробелом, пустые черные — символом `*`, а фигуры и пешки задаются аббревиатурами **K**ing (король), **Q**ueen (ферзь), **R**ook (ладья), **B**ishop (слон), k**N**ight (конь) и **P**awn (пешка) независимо от того, на поле какого цвета они расположены (прописными буквами обозначаются белые фигуры и пешки, строчными — черные). На рис. 8.1 показан пример использования этой команды. Решение этой задачи помещено на с. 341.

Запись шахматных партий

Игра начинается либо командой `\newgame`, либо окружением `position`. Последний вариант дает вам возможность задать расположение фигур, отличное от начального. Например, позиция после 25-го хода в партии Фишер–Таль (см. рис. 8.2 на с. 329) определяется следующим образом:

```
\begin{position}
  \White(Kh1,Qe6,Re1,a2,b2,c2,g2,h2)
  \Black(Kf8,Qb8,Rd7,Rg8,Be7,a6,b4,h7)
  \Whitetrue   \setcounter{movecount}{25}
\end{position}
```

Команда `\Whitetrue` указывает, что следующий ход делают белые, а `\Whitefalse` указывает, что следующий ход за черными. Кроме того, счетчик

ходов установлен равным 25, определяя тем самым, что следующим будет 26-й ход.

Позиции, определенные окружением `position` или командой `\newgame`, автоматически обновляются с помощью команд `\ply` и `\move`. Если ход белых не сопровождается комментариями, следует использовать команду `\move`, в противном случае ход может быть разбит на две части (полуход) с помощью команды `\ply`.

`\move` *WhiteMove* ␣ *BlackMove*
`\ply` *Move*

Аргументы описывают поля, на которых фигуры стояли до и после выполнения хода (или перемещение короля в случае рокировки). Для обозначения полей используется принятая в Европе алгебраическая нотация, т. е. положение по вертикали обозначается *буквой* из диапазона `a...h`, а положение по горизонтали — *цифрой* из диапазона `1...8`. Таким образом, например, ход белого коня с поля `g1` на поле `f3` записывается как `g1f3` (без пробелов).

Пешка, достигшая последней горизонтали, превращается в ферзя, ладью, слона или коня путем добавления аббревиатуры `Q`, `R`, `B` или `N`. Если новая фигура явно не описана, по умолчанию ею считается ферзь. Ходы могут снабжаться комментариями в виде символов «`!`» в случае сильных ходов, «`??`» в случае неожиданно слабых ходов и т. д. Вдобавок к автоматическому выбору новой фигуры макро умеют обрабатывать взятия пешками «на проходе».

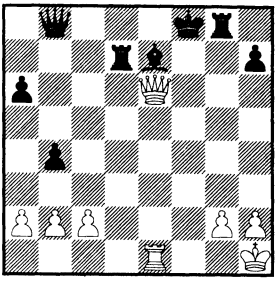
Эта сокращенная форма записи во внутреннем представлении превращается в длинную алгебраическую запись. Так, например, указание хода `g1f3` будет преобразовано к виду ♘g1–f3 в том случае, если поле `f3` является пустым, или к виду ♘g1×f3 в случае взятия, а ход `e1c1` (рокировка) превратится в 0–0–0. На рис. 8.2 в качестве примера приведен отрывок снабженной комментариями записи партии между Фишером и Талем во время Турнира претендентов 1959 г. Комментарии принадлежат Фишеру (краткую алгебраическую запись всей партии см. на рис. 8.3 на с. 331).

			`\usepackage {chess}`
1.	♘g1–f3	d7–d6	`\newgame`
2.	d2–d4	♗c8–g4	`\move g1f3 d7d6`
3.	c2–c4	♘g8–f6	`\move d2d4 c8g4`
4.	♘b1–c3		`\move c2c4 g8f6`
4.	...	♗g4×f3	`\ply b1c3`
5.	g2×f3	♘b8–d7	`\ply g4f3`
6.	e2–e3	e7–e5	`\move g2f3 b8d7`
7.	d4×e5	d6×e5	`\move e2e3 e7e5`
8.	♕d1–c2	c7–c6	`\move d4e5 d6e5`
9.	♗c1–d2	♕d8–c7	`\move d1c2 c7c6`
10.	0–0–0	0–0–0	`\move c1d2 d8c7`
			`\move e1c1 e8c8`

8-1-2

```
\begin{position}
\White(Kh1,Qe6,Re1,a2,b2,c2,g2,h2)
\Black(Kf8,Qb8,Rd7,Rg8,Be7,a6,b4,h7)
\Whitetrue \setcounter{movecount}{25}
\end{position}
\begin{center}
\centerline{Diagram: Fisher--Tal
 after 25.~\ldots, {\Fig K}f8!}
\[\showboard\]
\end{center}

\ply  e6d7
Not |26. Rf1+, Kg7; 27. Rf7+, Kh8;
and if 28. Q*d7, Rd8; 29. Qg4, Qe5|
wins.

\ply       b8d6
\move d7b7 g8g6
Within a handful of moves the game
has changed its complexion. Now it
is White who must fight for a draw!

\ply  c2c3!
Black's extra piece means less with
each pawn that's exchanged.

\ply       a6a5
On |28.: b*c3; 29. Qc8+, Bd8;
30. Q*c3|=.

\ply  b7c8+
On the wrong track. Right is
|29. c*b4!, Q*b4 (if 29.: a*b4;
30. a3!, b*a3; 31. b*a3, Q*a3 draws);
30. Qf3+, Kg7; 31. Qe2| draws, since
Black can't possibly build up a
winning K-side attack and his own
king is too exposed.

\ply       f8g7
\move c8c4 e7d8
\move c3b4 a5b4
On |31.: Q*b4; 32. Qe2| White should
draw with best play.
\[\showboard\]
```

Diagram: Fisher–Tal after 25. ..., ♔f8!



26. ♕e6×d7

Not 26. ♖f1+, ♔g7; 27. ♖f7+, ♔h8; and if 28. ♕×d7, ♖d8; 29. ♕g4, ♕e5 wins.

26. ... ♕b8–d6

27. ♕d7–b7 ♖g8–g6

Within a handful of moves the game has changed its complexion. Now it is White who must fight for a draw!

28. c2–c3!

Black's extra piece means less with each pawn that's exchanged.

28. ... a6–a5

On 28. ..., b×c3; 29. ♕c8+, ♗d8; 30. ♕×c3=.

29. ♕b7–c8+

On the wrong track. Right is 29. c×b4!, ♕×b4 (if 29. ..., a×b4; 30. a3!, b×a3; 31. b×a3, ♕×a3 draws); 30. ♕f3+, ♔g7; 31. ♕e2 draws, since Black can't possibly build up a winning K-side attack and his own king is too exposed.

29. ... ♔f8–g7

30. ♕c8–c4 ♗e7–d8

31. c3×b4 a5×b4

On 31. ..., ♕×b4; 32. ♕e2 White should draw with best play.

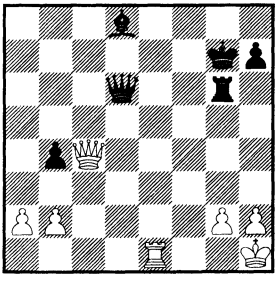


Рис. 8.2. Пример LaTeX'овских команд и полученного результата при наборе шахматной партии.

Турнирный стиль

Ради экономии места в турнирных бланках используется сокращенная алгебраическая запись, при которой всюду, где это возможно, опускается описание поля, *откуда* переместилась фигура. Если расшифровка хода неоднозначна (поскольку на одно и то же поле могли попасть несколько разных фигур), добавляется буква или цифра, обозначающая положение вертикали или горизонтали исходного поля. Сокращенная и полная алгебраические записи также используются при анализе партий, когда комментарии выходят за пределы основной строки, т. е. при описании реальной партии или ее возможных продолжений. Для облегчения набора ходов при сокращенной алгебраической записи существует компактная форма записи `|...|`. В тех случаях, когда нельзя воспользоваться символом `|`, его можно вызвать внутри окружения `chess`. Ниже приведены несколько примеров полной и сокращенной алгебраической записи при анализе партии:

ввод	вывод
`|21. Nf3-e5??, 0-0!|`	21. ♘f3–e5??, 0–0!
`|21.: Ke8*f8|`	21. ..., ♔e8×f8
`|21. Nfe5, K*f8|`	21. ♘fe5, ♔×f8
`|21.: f8Q+|`	21. ..., f8♕+

При анализе партии некоторые символы приобретают специальное значение: «`*`» превращается в «×» (взятие), «`-`» — в «–», а «`:`» служит сокращением для многоточия «...,» (необходимого, когда анализ начинается с хода черных).

Такое использование символа `|` обладает одним нежелательным побочным эффектом: в данном документе этот символ больше нельзя использовать для других целей. Возможно, самым важным из таких применений является то, что он выступает в роли вертикальной линейки в синтаксисе данного окружения. Чтобы решить эту проблему, в пакете `chess` существует окружение `nochess`, внутри которого символ `|` имеет обычное значение. В случае необходимости вместо него можно применить окружение `chess`.

В турнирном стиле основные ходы обычно набираются полужирным шрифтом, а возможные продолжения — прямым. С помощью следующих команд можно изобразить некоторые общепринятые знаки, характеризующие состояние или оценки конкретных ходов:

	LaTeX	значение
⩲	`\wbetter`	у белых небольшое преимущество
⩱	`\bbetter`	у черных небольшое преимущество
±	`\wupperhand`	положение белых лучше
∓	`\bupperhand`	положение черных лучше
+−	`\wdecisive`	белые имеют решающее преимущество
−+	`\bdecisive`	черные имеют решающее преимущество
→	`\withattack`	с нападением
⩳	`\betteris`	было бы лучше
o-o	`\seppawns`	одинокие пешки

1. e4 c5 2. ♘f3 d6 3. d4 cd4 4. ♘d4 ♘f6 5. ♘c3 a6 6. ♗c4 e6 7. ♗b3 [7. 0–0 ♗e7 8. ♗b3 ♕c7 9. f4 b5 10. f5 b4 11. fe6!? (11. ♘ce2 e5 12. ♘f3 ♗b7∓) bc3 12. ef7 ♔f8 13 ♗g5 ♘g4!∓] **b5!** [7 ..., ♗e7? 8. f4 0–0 9. ♕f3 ♕c7 10. f5! (10. 0–0? b5 11. f5 b4! 12. ♘a4 e5 13. ♘e2 ♗b7∓) e5 11. ♘de2 b5 12. a3 ♗b7 13. g4±→] **8. f4!? b4! 9. ♘a4 ♘e4 10. 0–0 g6?** [⌓10 ..., ♗b7] **11. f5! gf5** [11 ..., ef5? 12. ♗d5 ♖a7 13. ♘f5! gf5 14. ♕d4] **12. ♘f5! ♖g8** [⌓12 ..., d5 13. ♘h6 ♗h6 14. ♗h6; 12 ..., ef5? 13. ♕d5 ♖a7 14. ♕d4] **13. ♗d5! ♖a7** [13 ..., ed5 14. ♕d5 ♗f5 15. ♖f5 ♖a7 16. ♕e4 ♖e7 17. ♕b4 ♖e2 18. ♗g5! ♖g5 19. ♖g5 ♕g5 20. ♖b8+−PANOV] **14. ♗e4?** [⌓14. ♗e3! ♘c5 15. ♕h5! ♖g6 (15 ..., ♘a4 16 ♗a7 ed5 17 ♖ae1) 16. ♖ae1! KEVITZ] **ef5 15. ♗f5 ♖e7! 16. ♗c8 ♕c8 17. ♗f4?** [⌓17. c3! ♕c6 18. ♖f2; 17. ♕d6? ♖g2 18. ♔g2 ♖e2 19. ♔f3 ♗d6 20. ♔e2 ♕c2−+] **♖g ♕c6! 18. ♕f3 ♕a4** [18 ..., ♕f3 19. ♖f3 ♖e2 20. ♖f2 ♖f2 21. ♔f2±o·o] **19. ♗d6 ♕c6! 20. ♗b8 ♕b6** [20 ..., ♕f3? 21. ♖f3 ♗g7 22. c3+−] **21. ♔h1 ♕b8 22. ♕c6** [22. ♖ae1 ♔d8! 23. ♖d1 ♔c7! (23 ..., ♔c8? 24. ♕c6+−) 24. ♕f4 (24. ♖d4 ♕b7!) ♔b7 25. ♖d6 ♕c7 26. ♕b4 ♔c8 27. ♖a6 ♕b7! 28. ♕b7 ♔b7 29. ♖af6 ♖g7=; 22 ..., 6? 23. ♕f7 ♔d7 24. ♖d1! ♖d6 25. ♖d6 ♔d6 26. ♖f6!+−] **♖d7 23. ♖ae1** [23. ♖ad1 ♗d6 24. ♖f7 (24. ♖f6 ♖g6 25. ♖dd6 ♕d6!) ♕c7; 23. ♖f7 ♕d6] **♗e7** [23 ..., ♔d8? 24. ♖f7! ♗e7 25. ♖fe7 ♖e7 26. ♖d1+−] **24. ♖f7 ♔f7 25. ♕e6 ♔f8!** [25 ..., ♔g7? 26. ♕d7+−] **26. ♕d7** [26. ♖f1? ♔g7 27. ♖f7 ♔h8 (28. ♕d7 ♖d8 29. ♕g4 ♕e5−+)] **♕d6 27. ♕b7 ♖g6 28. c3! a5** [28 ..., bc3 29. ♕c8 ♗d8 30. ♕c3=] **29. ♕c8** [⌓29. cb4! ♕b4 (29 ..., ab4 30. a3! ba3 31. ba3 ♕a3=) 30. ♕f3 ♔g7 31. ♕e2=] **♔g7 30. ♕c4 ♗d8 31. cb4 ab4** [31 ..., ♕b4 32. ♕e2=] **32. g3?** [32. ♕e4 (32 ..., ♗c7 33. ♕e7 ♔g8 34. ♕e8 ♕f8 35. ♕e4)] **♕c6 33. ♖e4 ♕c4 34. ♖c4 ♖b6!** [34 ..., ♗e7? 35. a3!] **35. ♔g2 ♗f6 36. ♔f3 ♔e5 37. ♔e3 ♗g5 38. ♔e2 ♔d5 39. ♔d3 ♗f6 40. ♖c2? ♗e5 41. ♖e2 ♖f6 42. ♖c2 ♖f3 43. ♔e2 ♖f7 44. ♔d3 ♗d4! 45. a3** [45. b3? ♖f3 46. ♔e2 ♖f2 47. ♕d3 ♖c2 48. ♔c2 ♔e5+−]
b3 46. ♖c8 [46. ♖e2 ♖f3 47. ♔d2 ♗b2; 46. ♖d2 ♖f3 47. ♔e2 ♖f2] **♗b2 47. ♖d8 ♔c6 48. ♖b8 ♖f3 49. ♔c4 ♖c3 50. ♔b4 ♔c7 51. ♖b5 ♗a1 52. a4 b2!** [53. ♔c3 b1♕!] **0 : 1** [**Fisher**]

```
\begin{mainline}{0~:~1}{Fisher}
1.  e4  c5   2. Nf3 d6   3. d4  cd4      4. Nd4 Nf6   5. Nc3 a6 ...
13. Bd5! Ra7
   \begin{variation}
   13: ed5       14. Qd5 Bf5 15. Rf5 Ra7 16. Qe4 Re7  17. Qb4 Re2
   18. Bg5! Rg5 19. Rg5 Qg5 20. Rb8\wdecisive
   \begin{nochess} PANOV \end{nochess}
   \end{variation}
14. Be4?            ...
\end{mainline}
```

Рис. 8.3. Запись шахматного турнира.

Для иллюстрации турнирного стиля на рис. 8.3 представлена запись полной партии между Бобби Фишером и Михаилом Талем (Belgrade, 1959). Для облегчения написания программы были использованы следующие окружения, которые не входят в пакет chess:

```
\newenvironment{mainline}[2]{\bfseries
  \newcommand{\result}{#1}\newcommand{\commentator}{#2}\begin{chess}}
 {\end{chess}\finito{\result}{\commentator}}
\newenvironment{variation}{[\mdseries\ignorespaces}{]\ignorespaces}
\newcommand{\finito}[2]{{\bfseries\hfill#1\hfill[#2]\par}}
```

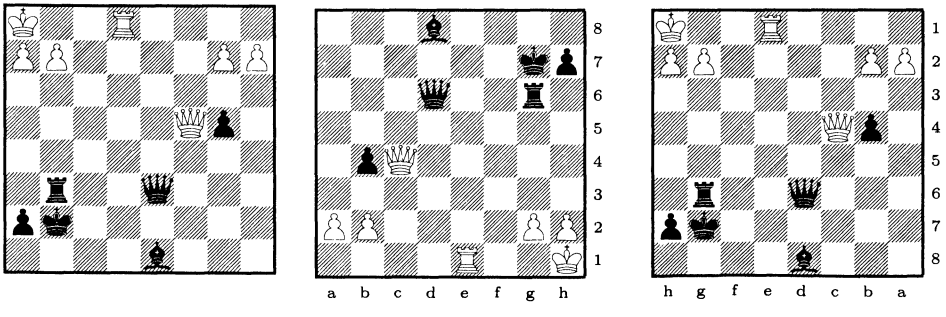
8.1.3 Расширения пакета chess

Франк Хассель разработал интересный дополнительный пакет для записи шахматных партий по переписке (Hassel, 1993). Его пакет bdfchess также предоставляет несколько дополнительных возможностей, показанных на рис. 8.4.

В случае ведения игры по переписке традиционно играются две партии одновременно, причем каждый участник играет одну из партий белыми, т. е. делает первый ход. В данном пакете это реализуется с помощью двух окружений `gameone` и `gametwo`, ведущих запись текущей позиции в каждой игре. В этих окружениях вы можете использовать команды `\postmove` и `\postply`, напоминающие `\move` и `\ply`, которые вдобавок собирают информацию о датах получения открыток и о времени, затраченном каждым игроком на обдумывание хода:

`\postmove` *WhiteArrival* ␣ *Days* ␣ *Move* ␣ *BlackMove* ␣ *Arrival* ␣ *Days*
`\postply` *Move* ␣ *Arrival* ␣ *Days*

Даты получения (*Arrival*) задаются в формате `dd.mm.yyyy`, аргумент *Days* обозначает требующееся на обдумывание ответного хода время, измеренное в це-



`\showinversboard` `\showboardwithnotation` `\showinversboardwithnotation`

Рис. 8.4. Дополнительные команды для изображения шахматных позиций.

лых днях. Окружения `gameone` и `gametwo` создают список ходов в виде полной алгебраической записи. Ходы или полуходы можно снабжать комментариями и в любой момент времени с помощью команды `\showboard` или одной из ее разновидностей можно графически изобразить текущее положение фигур.

```
\begin{gameone}
\postmove 11.09.1996 0 d2d4 g8f6 12.09.1996 2
\postmove 15.09.1996 1 c2c4 g7g6 17.09.1996 2
\postply  21.09.1996 1 b1c3
\end{gameone}
```

Date						Date		
Arrival	Leaving	$\sum$	**White**		**Black**	Arrival	Leaving	$\sum$
11.9.96	11.9.96	0	d2–d4	1.	♘g8-f6	12.9.96	14.9.96	2
15.9.96	16.9.96	1	c2–c4	2.	g7–g6	17.9.96	19.9.96	4
21.9.96	22.9.96	2	♘b1–c3	3.				

В этом пакете есть несколько команд для создания почтовой открытки на основе информации, сохраненной окружениями `gameone` и `gametwo`. Наиболее важные из них перечислены ниже.

`\sender{`*address*`}`
`\receiver{`*address*`}`
`\postcard[`*frame*`] (` *h-offset*, *v-offset* `)`
`\postcardaddress[`*split*`] (` *h-offset*, *v-offset* `)`

Адреса отправителя и получателя *address* можно указывать в произвольной форме, по мере необходимости привлекая LaTeX'овские команды и окружения. Команда `\postcard` заполняет оборотную сторону открытки, демонстрируя текущее состояние партии. Если указан необязательный аргумент *frame*, состоящий из буквы `f`, то содержимое открытки заключается в рамку. Будучи использованной, команда `\postcardaddress` создает лицевую сторону открытки, где помещаются адреса, которые указаны в командах `\sender` and `\receiver`. Необязательный аргумент *split* может содержать одну из двух или обе буквы `f` (обозначающую frame (рамка)) и `t` (которая на лицевой стороне открытки проводит вертикальную черту и добавляет слово «postcard»). Аргументы *offset* обеих команд определяют смещения образа открытки относительно листа бумаги, если в них возникает необходимость. Полный пример, демонстрирующий вводимые команды и получаемый результат (за исключением списков ходов, созданных игровыми окружениями), приведен на рис. 8.5 и 8.6.

Франк Хассель добавил шрифт `chess15` для создания диаграмм, которые лучше приспособлены для размещения на почтовых открытках. Чтобы использовать его в пакетах chess или bdfchess, необходимо после загрузки пакета добавить следующую строку:

```
\font\Chess=chess15
```

```
\begin{gameone}
  \postmove 03.09.1996 0 e2e4 d7d6 11.09.1996 0
  \postmove 12.09.1996 2 d2d4 g8f6 15.09.1996 1
  \postmove 17.09.1996 2 b1c3 g7g6 21.09.1996 1
\end{gameone}
\begin{gametwo}
  \postmove 11.09.1996 0 d2d4 g8f6 12.09.1996 2
  \postmove 15.09.1996 1 c2c4 g7g6 17.09.1996 2
  \postply  21.09.1996 1 b1c3
\end{gametwo}
\sender{\small sender:\\
        \large\textbf{TUG Office}     \\
        \TeX{} User Group             \\
        1850 Union Street, \#1637     \\
        San Francisco CA 94123        \\
        USA}
\receiver{\setlength{\parskip}{5mm} \LARGE \bfseries
          DANTE                     \vskip 2pt \hrule
          Postfach 10\,18\,40       \vskip 2pt \hrule
          69008 Heidelberg          \vskip 2pt \hrule
          Germany}
\cardmessage{The office will be closed from\\ 30.09.96 until 14.10.96}
\postcard[f](0mm,-20mm)

\vspace{10mm}

\postcardaddress[tf](-1mm,0mm)
```

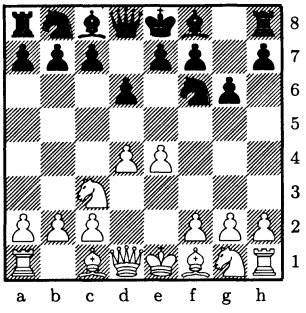
Рис. 8.5. Информация, вводимая в bdfchess для создания шахматной почтовой открытки.

Он также разработал пакет chesskey с единственной командой `\key`, параметром которой является код дебюта согласно классификации Chess-Openings Classification Scheme; его значение лежит в диапазоне от A00 до E99.

8.1.4 Взаимодействие с шахматными базами данных

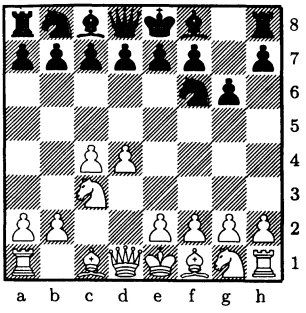
Существует несколько шахматных баз данных, которые содержат описания законченных партий, например ChessBase и NicBase[1] Чтобы получить возможность набирать партии, хранящиеся в подобных базах данных, Юрген Ламерс написал на языке Pascal набор программ, основанных на более ранней работе Карстена Хансена. Эти программы преобразуют форматы шахматных баз данных в LaTeX'овские входные файлы, которые можно использовать совместно с пакетом chess.

[1]Информацию об используемых в них форматах хранения информации можно найти в архиве CTAN, **support/chesstools**.



Game 1

No.	Your move	No.	M move
3	♘b1–c3	3	g7–g6



Game 2

No.	Your move	No.	M move
2	g7–g6	3	♘b1–c3

Received on 21.9.96
Replied on 22.9.96
My time taken 1 day(s)
My total time 2 days

Your postmark date 19.9.96
Your time taken 2 day(s)
Your total time 4 days

Yours sincerely

sender:
TUG Office
TEX User Group
1850 Union Street, #1637
San Francisco CA 94123
USA

postcard

The office will be closed from
30.09.96 until 14.10.96

DANTE
Postfach 10 18 40
69008 Heidelberg
Germany

8-1-3

Рис. 8.6. Шахматная почтовая открытка, созданная с помощью пакета bdfchess.

Например, команда для извлечения партии из базы в формате ChessBase имеет вид:

```
cb2tex fritz2 fritz2.tex
```

Она создает запись партии в «полной нотации», в которой каждый ход записывается в виде `\move e4e5 d6*e5`. Кроме того, после каждых 15 ходов в выходной файл добавляется команда `\showboard`.

Избрав другой путь, можно получить в выходном файле запись партии в «сокращенной» форме, указав что-нибудь вроде

```
nic2tex fischer3 fischer.tex -s
```

В этом случае партия, записанная в формате базы данных NicBase, выводится в LaTeX'овский файл в сокращенной форме, например как `12. g2g4, ♘c6d4`. Эти программы вместе с некоторыми частями баз данных доступны в архиве CTAN, `support/chesstools` в виде исполняемых и исходных файлов.

8.1.5 Использование шрифта cheq фирмы Adobe

Cheq — это PostScript'овский шрифт формата Type1, бесплатно предоставляемый компанией Adobe. Его подключение к системе LaTeX впервые было реализовано Адрианом Кларком с помощью им же написанного пакета cheq, в котором определялся набор таких команд, как `\CheqBKing`, для осуществления манипуляций с отдельными фигурами, но не было предусмотрено никаких средств для изображения доски. Благодаря этому пакету и разработанным Вольфгангом Аппельтом (Appelt, 1988) шахматным макро Себастиан Ратц написал файл `chessboard.tex`, демонстрирующий один из возможных способов использования этого шрифта.

В данном разделе мы избрали другой путь: приспособить данный шрифт к работе в пакетах chess и bdfchess. С одной стороны, это служит примером использования PostScript'овских средств для работы со шрифтами, обсуждаемых в гл. 10, а с другой стороны является руководством по настройке и интеграции других (коммерческих) шахматных шрифтов так, чтобы эти мощные пакеты могли их использовать.

Шрифт Cheq поставляется в виде pfa-файла вместе с соответствующими afm- и pl-файлами[1]. Для того чтобы использовать этот шрифт в LaTeX'е, мы должны прежде всего на основе pl-файла создать tfm-файл:

```
$ pltotf -verbose cheq.pl cheq.tfm
This is PLtoTF, Version 3.4 (C version 6.1)
'055 '057 '102 '104 '113 '114 '116 '120
'121 '122 '123 '124 '134 '137 '142 '144
'153 '154 '156 '160 '161 '162 '163 '164
'170 '240 '247 '250 '266 '270 '271 '272
```

[1] В архиве CTAN имена всех входящих в дистрибутив файлов написаны строчными буквами, но вы также можете встретиться с такими именами файлов, как `Cheq.pfa`.

```
'265 '302 '316 '317 '345 '346 '352 '353
'360 '361 '365 '366 '373.
```

Этого достаточно, чтобы получить возможность использовать этот шрифт в LaTeX'е, но мы еще должны сообщить драйверу PostScript'овского принтера, что этот шрифт следует загрузить в принтер, а не использовать в виде pk-файла. В случае драйвера dvips это можно выполнить, написав маленький файл (таблицу соответствия) `cheq.map`, который содержит следующую строку:

```
cheq Cheq <cheq.pfa
```

Обратите внимание, что в этой таблице первым указано имя файла в той форме, в какой его видит TeX, т. е. имя tfm-файла, а вторым идет имя, используемое в исходном тексте шрифта на PostScript'е (оно начинается с прописной буквы Cheq, поскольку именно так этот шрифт называется в файле `cheq.pfa` в команде `FontName`; подробнее см. гл. 10).

Чтобы сообщить драйверу dvips об этой таблице соответствия, мы можем или обновить его главный конфигурационный файл `config.ps`, или создать другой однострочный конфигурационный файл (`config.cheq`), состоящий из следующей строки:

```
p +cheq.map
```

Если мы выберем второй путь, то должны будем использовать драйвер dvips для обработки любого содержащего данный шрифт документа следующим способом:

```
$ dvips -Pcheq chessboard   # run dvips with the config file
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.08.25:2245' -> chessboard.ps
<tex.pro><cheq.ps><texps.pro>. [1]
```

Чтобы использовать этот шрифт с пакетом chess, мы можем попробовать после загрузки пакета ввести что-то вроде `\font \Chess=cheq at 10pt`, но вскоре обнаружится, что такой способ не работает потому, что, к сожалению, одни и те же символы в шрифтах `cheq` и `chess10` находятся в разных позициях![1] Чтобы достичь желаемого результата, мы должны изменить раскладку PostScript'овского шрифта, т. е. поместить нужные символы на нужные места. Это не так сложно, как может показаться. Ниже описаны шаги, которые необходимо предпринять, и показаны результаты. Подробности изложены в гл. 10.

Сначала создадим таблицы обоих шрифтов, чтобы знать, где какой символ находится. Это можно легко сделать с помощью стандартного LaTeX'овского файла nfssfont.tex.

```
$ latex nfssfont
This is TeX, Version 3.14159 (C version 6.1)
(/cdrom/texmf/tex/latex/base/nfssfont.tex
LaTeX2e <1995/12/01> patch level 2
```

[1] По той причине, что в данном пакете уже продекларирован шрифт `\Chess`, мы должны прибегнуть к команде низкого уровня `\font`, а не `\newfont`, поскольку в LaTeX'е нет команды `\renewfont`.

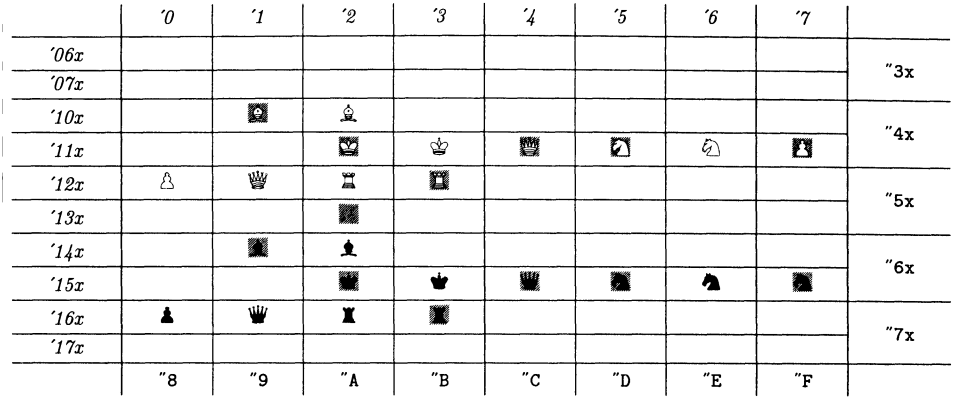
```
...
*******************************************
* NFSS font test program version <v2.0d>
*
* Follow the instructions
*******************************************

Name of the font to test = chess10
Now type a test command (\help for help):)
*\table

*\init

Name of the font to test = cheq at 10pt
Now type a test command (\help for
help):
*\table

*\bye
[1]
Output written on nfssfont.dvi (1 page, 9396 bytes).
Transcript written on nfssfont.log.
```

Результат работы LaTeX'а, показанный на рис. 8.7, предоставляет нам информацию о реальном расположении каждого символа в этих шрифтах.

Чтобы изменить раскладку PostScript'овского шрифта, мы должны определить новый вектор раскладки, который соответствовал бы раскладке кодировки шрифте `chess10`, и перечислить PostScript'овские имена используемых в нем символов. Эти имена можно найти в `afm`-файле (или в `pfa`-файле). Мы должны составить список ровно из 256 имен, помещая в тех местах, где не должно быть никакого символа, «пустышки» `/.notdef`.

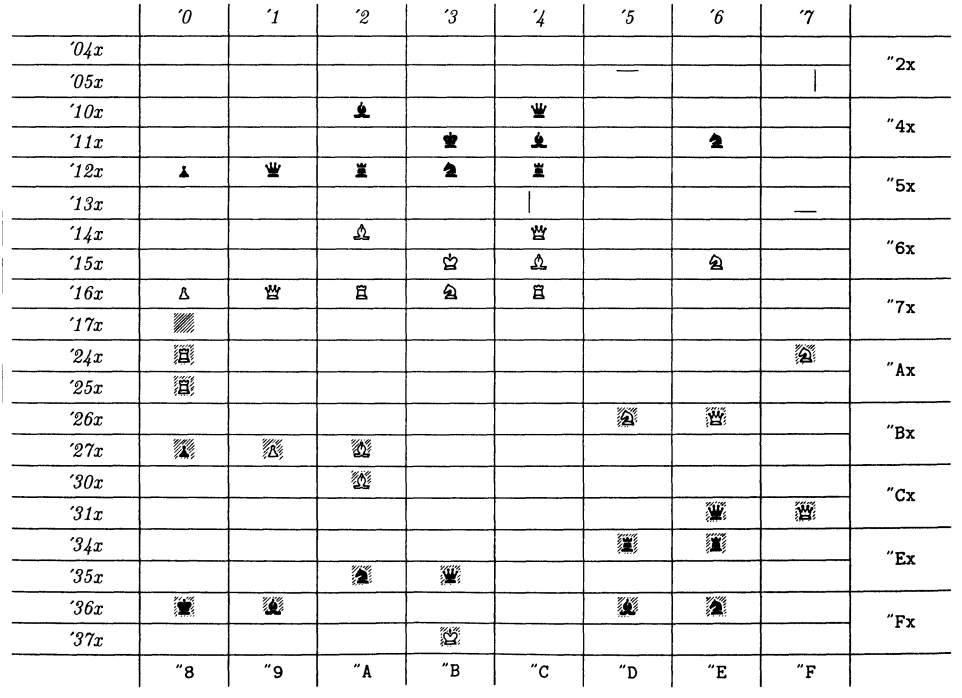
```
/ChessEncoding [
/.notdef /.notdef /.notdef /.notdef /.notdef /.notdef
/.notdef /.notdef
% many more such lines
  ...
% '100 - '107
/.notdef /WBishopOnBlack /WBishop /.notdef /.notdef /.notdef
/.notdef /.notdef
% '110 - '117
/.notdef /.notdef /WKingOnBlack /WKing /WQueenOnBlack
/WKnightOnBlack /WKnight /WPawnOnBlack
% '120 - '127
/WPawn /WQueen /WRook /WRookOnBlack /.notdef /.notdef /.notdef /.notdef
% '130 - '137
/.notdef /.notdef /BSquare /.notdef /.notdef /.notdef /.notdef /.notdef
% '140 - '147
/.notdef /BB
```

Test of chess10 on January 14, 1997 at 2056



	'0	'1	'2	'3	'4	'5	'6	'7	
'06x									"3x
'07x									
'10x									"4x
'11x									
'12x									"5x
'13x									
'14x									"6x
'15x									
'16x									"7x
'17x									
	"8	"9	"A	"B	"C	"D	"E	"F	

Test of cheq at 10pt on January 14, 1997 at 2056



	'0	'1	'2	'3	'4	'5	'6	'7	
'04x									"2x
'05x									
'10x									"4x
'11x									
'12x									"5x
'13x									
'14x									"6x
'15x									
'16x									"7x
'17x									
'24x									"Ax
'25x									
'26x									"Bx
'27x									
'30x									"Cx
'31x									
'34x									"Ex
'35x									
'36x									"Fx
'37x									
	"8	"9	"A	"B	"C	"D	"E	"F	

Рис. 8.7. Раскладки шрифтов chess10 и cheq.

```
ishopOnBlack /BBishop /.notdef /.notdef /.notdef /.notdef
/.notdef
% and a few similar lines for the remaining Black pieces
  ...
% followed by a lot of
/.notdef /.notdef /.notdef /.notdef /.notdef /.notdef /.notdef
/.notdef    ...
% and finished with
] def
```

Наконец остается сделать последний шаг — на основе этого вектора кодировки и исходного шрифта создать шрифт с измененной раскладкой (для LaTeX'а, дав ему другое имя). Новый шрифт можно было бы назвать, например, `xcheq`. Чтобы создать `tfm`-файл, необходимый для использования шрифта в LaTeX'е, мы можем воспользоваться программой afm2tfm, как это описано ниже в предположении, что вектор раскладки был сохранен в файле `xchess.enc` (с другими способами решения этой задачи можно познакомиться в разд. 10.5.6 на с. 453):

```
$ afm2tfm cheq -T xchess.enc xcheq
```

Это приводит к созданию файла `xcheq.tfm`, так что теперь мы можем использовать данный шрифт в LaTeX'е. Как и прежде, нам следует позаботиться о том, чтобы PostScript'овский драйвер загрузил в принтер данный шрифт (на этот раз с измененной раскладкой). В случае использования драйвера dvips этого можно достичь, обновив `map`-файл и добавив в него следующую строку:

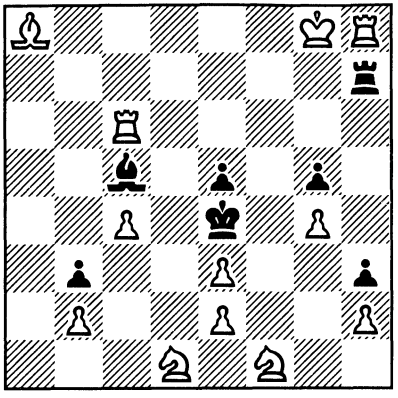
```
xcheq Cheq " ChessEncoding ReEncodeFont " <xchess.enc <cheq.pfa
```

Сделано! Теперь вы можете ознакомиться с решением шахматной задачи, заданной пару страниц назад. Оно показано на рис. 8.8, который также демонстрирует применение шахматного шрифта компании Adobe совместно с пакетом chess.

8.2 Сянци — китайские шахматы

Китайский вариант шахмат сянци имеет те же корни, что и европейский, но обладает некоторыми существенными отличиями. Например, две армии разделены рекой, которую отдельные фигуры не могут пересекать. Кроме того интересен тот факт, что армии состоят из разных наборов фигур: белые слоны заменены на противоположной стороне министрами, и если белого генерала окружают советники, то по бокам от черного главнокомандующего стоят офицеры. В европейских версиях этой игры, в которых китайские иероглифы заменены пиктограммами, эти различия обычно игнорируются.

Жак Рише создал METAFONT'овский шрифт, в котором фигуры китайских шахмат изображаются в виде традиционных китайских символов. Как видно на табл. 8.2, этот шрифт `cchess46` содержит набор черных и белых фигур



8-1-4

```
\usepackage {chess}
\font\Chess=xcheq at 18pt
\setlength{\fontdimen2\Chess}{0pt}
\board{B* * *KR}
      {* * * *r}
      { *R* * *}
      {* b p p }
      { *P*k*P*}
      {*p* P *p}
      { P *P* P}
      {* *N*N* }
\[ \showboard \]
```

Рис. 8.8. Решение задачи: белые делают ход, не дав мат черным.

Таблица 8.2. Раскладка фигур игры сянци в шрифте `cchess46`.

белые	черные	альтернативные черные	обычные названия
k 帥	K 將	S 將	король, генерал;
g 仕	G 士	T 士	страж, помощник;
b 相	B 象	U 象	епископ, слон;
n 馬	N 馬	V 馬	рыцарь, конь;
r 車	R 車	W 車	ладья, колесница, машина;
c 炮	C 炮	X 炮	пушка, орудие, артиллерист;
p 兵	P 卒	Y 卒	пешка, пехотинец.

(прописные буквы обозначают черные фигуры, строчные — белые). Выбранные буквы отвечают общепринятым именам аналогичных фигур в европейских шахматах, например генерал обозначается буквой «k», соответствующей королю. Кроме того, шрифт содержит третий набор фигур: альтернативный набор символов черной армии, помещенных на белом фоне. Этот набор полезен при создании цветных диаграмм хода игры — традиционно все фигуры изображаются зелеными и красными символами на белом фоне.

Изначально в дистрибутив не входило никакого пакета, содержащего окружения или команды для набора партий. Но он содержал LaTeX'овский документ `cchessboard.tex` для создания изображения доски, на которой фигуры

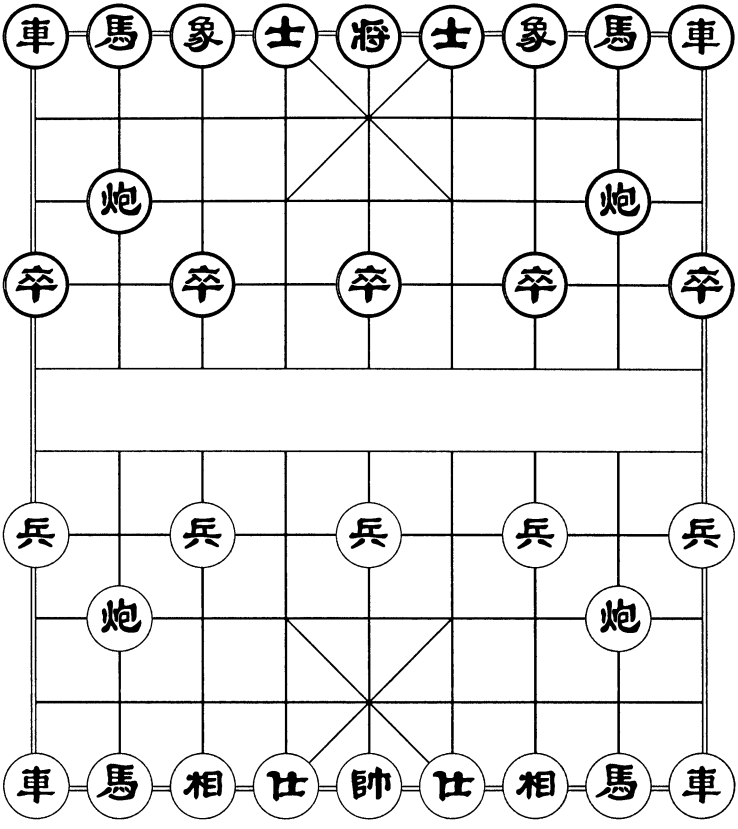


Рис. 8.9. Исходное расположение фигур в китайских шахматах (сянци).

занимают свои исходные позиции, как показано на рис. 8.9 (с той лишь разницей, что мы воспользовались альтернативным набором черных фигур).

Перегруппировав текст этого документа, Франк Миттельбах создал небольшой пакет cchess, в котором с помощью окружения `position` и команд `\piece` и `\textpiece` строятся изображения произвольных позиций.

`\textpiece{`*name*`}`

Команду `\textpiece` можно использовать внутри текста. Возможными значениями аргумента *name* могут быть имена фигур, приведенные в табл. 8.2 выше. С помощью этой команды и, например, окружения `tabbing` можно легко записывать и снабжать комментариями ход игры. Приведенная ниже запись игры с матом на четвертом ходу дает представление об использовании этой команды. Расположение фигур после этих четырех ходов показано на рис. 8.10.

1. 炮 h3–e3 馬 b0–a8
2. 炮 e3×e7 車 a0–a9
3. 炮 b3–b5 馬 h0–g8
4. 炮 b5–e5 mates!

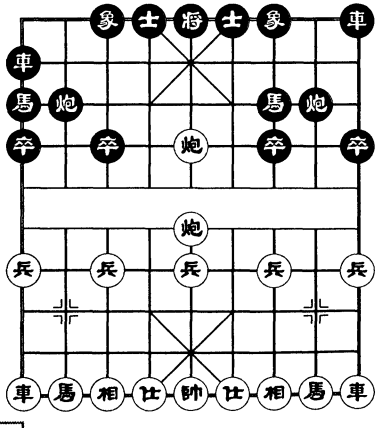
8-2-2

```
\usepackage {cchess}
\newcommand\x{$\times$} % a shortcut to
                        % denote capture
\begin{tabbing}
1. \= \textpiece{c}h3--e3 \qquad
      \=\textpiece{N}b0--a8 \\
2. \> \textpiece{c}e3\x e7
      \>\textpiece{R}a0--a9 \\
3. \> \textpiece{c}b3--b5
      \>\textpiece{N}h0--g8 \\
4. \> \textpiece{c}b5--e5 mates!
\end{tabbing}
```

Окружение `position` рисует позицию целиком. Для указания положения отдельных фигур внутри него используется команда `\piece`.

`\piece{`*file*`}{`*rank*`}{`*name*`}`

Команда `\piece` помещает фигуру *name* в стопку (file) и на уровень (rank) (горизонтальную и вертикальную позиции). Аргумент *file* может быть буквой из диапазона от `a` до `i` (отсчитываемая слева направо), тогда как аргумент *rank* нумеруется числами от `1` до `10` (отсчитываемыми снизу вверх). Пример использования этой команды приведен на рис. 8.10.



8-2-3

```
\usepackage {cchess}
\smallboard
\begin{position}
\piece{a}{1}{r}\piece{i}{1}{r}
\piece{b}{1}{n}\piece{h}{1}{n}
\piece{c}{1}{b}\piece{g}{1}{b}
\piece{d}{1}{g}\piece{f}{1}{g}
\piece{e}{5}{c}\piece{e}{7}{c}
\piece{a}{4}{p}\piece{c}{4}{p}
\piece{e}{4}{p}\piece{g}{4}{p}
\piece{i}{4}{p}\piece{e}{1}{k}
\piece{a}{9}{R}\piece{i}{10}{R}
\piece{a}{8}{N}\piece{g}{8}{N}
\piece{c}{10}{B}\piece{g}{10}{B}
\piece{d}{10}{G}\piece{f}{10}{G}
\piece{b}{8}{C}\piece{h}{8}{C}
\piece{a}{7}{P}\piece{c}{7}{P}
\piece{g}{7}{P}
\piece{i}{7}{P}\piece{e}{10}{K}
\end{position}
```

Рис. 8.10. Положение мата после четырех ходов.

Помимо этого в пакете определены команды `\largeboard`, `\normalboard` и `\smallboard`, с помощью которых задается размер доски и фигур. На рис. 8.9 показана доска обычного размера, а на рис. 8.10 применена команда `\smallboard`.

В текущей реализации пакета cchess для размещения фигур на доске без перестройки изображения заново используются команды `\special` для вставки PostScript'овских команд. Но как мы увидим ниже в разд. 8.3 это — не единственный способ решения подобной задачи. Возможно, в будущем появится новая улучшенная версия данного пакета, в которой будут реализованы концепции, использованные в пакете chess: она позволит вести запись турниров, снабжать партии комментариями и проводить анализ этой интересной восточной игры.

8.3 Го

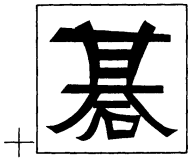
В го, возможно самой популярной в Азии игре, участвуют две стороны, которые располагают черные и белые камни на пересечениях сетки размера 19×19. После того как камни попали на доску, они никогда не перемещаются за исключением ситуации окружения, поэтому в процессе записи хода игры камни часто изображаются пронумерованными. Алгебраическая нотация, подобная той, что применяется в шахматах, обычно не используется.

Для набора диаграмм го и записи частично или полностью сыгранных партий Ханна Колодзейска разработала несколько шрифтов и сопутствующий пакет go. Существующие в трех размерах `10pt`, `15pt` и `20pt` шрифты содержат камни с номерами от ❶ до ㉕, а также камни без номеров, различные символы, рисунки пересечений и границ для конструирования внешнего вида игрового поля. Кроме того, есть шрифт, который содержит единственную букву — традиционный символ игры го, который показан на рис. 8.11.

Вообще говоря, в игре может использоваться более 250 камней, в случае чего позиция не может поместиться на одной диаграмме, созданной с помощью существующих шрифтов. Однако, поскольку ради большей наглядности на одной диаграмме не рекомендуется показывать более 100 ходов (после чего нумерация начинается заново), это теоретическое ограничение не имеет на практике большого значения.

Как и в пакете chess, в пакете go существуют команды для изображения начальной позиции, расположения камней (что в данном случае эквивалентно ходам) и команды для изображения доски или ее частей. Однако здесь используется совсем другой синтаксис.

Для расположения на доске камней с помощью команд из пакета go каждой точке пересечения присваиваются метка строки (строчная буква из множества `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`, `i`, `k`, `l`, `m`, `n`, `o`, `p`, `q`, `r`, `s`, `t`, обратите внимание на отсутствие буквы `j`!) и номер столбца (из диапазона от `1` до `19`).



```
\newfont\gosign{gosign50}
\begin{center}
  \fbox{\gosign\symbol{0}}
\end{center}
```

8-3-1

Рис. 8.11. Символ го.

`\inifulldiagram`
`\inidiagram` *row*$_1$`-`*row*$_2$`:` *col*$_1$`-`*col*$_2$␣

Первая команда `\inifulldiagram` полностью освобождает всю доску, тогда как вторая команда `\inidiagram` освобождает поля, которые расположены в строках, начиная с буквы *row*$_1$ по *row*$_2$, и в столбцах с номерами от *col*$_1$ по *col*$_2$. Обратите внимание, что за последним номером столбца *обязательно* должна идти новая или пустая строка.

После загрузки пакета go на доске автоматически создается исходное положение, т. е. удаляются все камни. Следовательно, приведенные выше команды необходимы только в том случае, когда требуется рисовать более одной диаграммы.

`\gofontsize{`*size*`}`

Команда `\gofontsize` предназначена для указания размера используемых камней. Аргумент *size* может принимать значения 10, 15 или 20 и обозначает размер шрифта в пунктах. На рис. 8.12 показана диаграмма, набранная шрифтом 15pt, а на рис. 8.13 — часть диаграммы, набранная шрифтом 20pt.

Команда `\pos` позиционирует камни на доске. В общем случае ее синтаксис имеет следующий вид:

`\pos{`*row*`}{`*col*`}=\`*player*`{`*display*`}`

Аргумент *row* представляет собой букву, обозначающую строку, а *col* — номер столбца. Команда `\`*player* может быть одной из команд `\black`, `\white` или `\letter` (или `\symbol`[1]). Аргумент *display* задает число, которое помещается внутри окружности; точка указывает на то, что окружность набирается без каких-либо украшений. Внутрь изображения особо важных камней принято

[1] Обратите внимание, что пакет, к сожалению, переопределяет стандартную LaTeX'овскую команду `\symbol`. В конце этого раздела даны рекомендации по решению технических проблем, которые могут возникнуть в связи с этим.

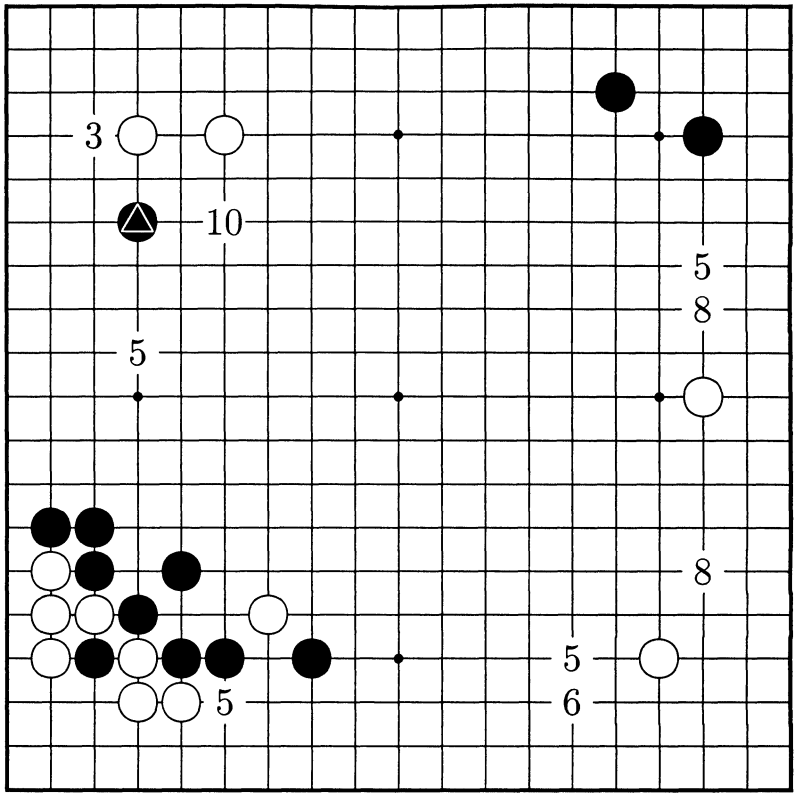


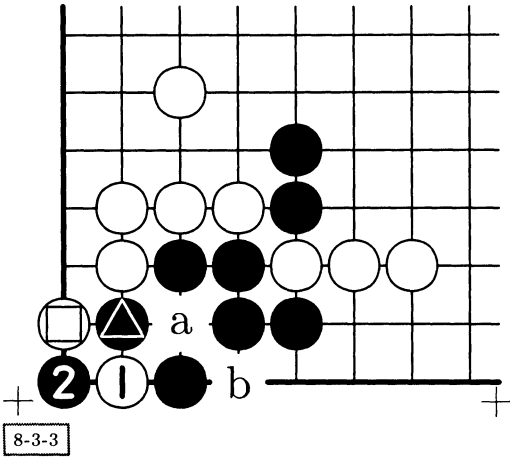
Рис. 8.12. Игра го на ранней стадии.

вписывать треугольник или квадрат. Это можно выполнить с помощью команд `\triangle` или `\square`, указанных в качестве аргумента *display*. Если используется команда `\letter`, то *display* обозначает букву или число, которую надо поместить в точке пересечения. Этот метод часто используется при обсуждении задач в игре го. На рис. 8.13 приведен пример, демонстрирующий все перечисленные варианты команды `\pos`.

Для демонстрации текущей ситуации на доске существуют две следующие команды:

`\showfulldiagram`
`\showdiagram` row_1`-`row_2`:` col_1`-`col_2␣

Команда `\showdiagram` имеет такой же синтаксис, что и обсуждавшаяся ранее команда `\inidiagram`. При анализе задач в го и демонстрации их решений в




```
\usepackage {go}
\inifulldiagram \gofontsize{20}
\pos{p}{5}=\black{.} \pos{q}{2}=\white{.}
\pos{o}{3}=\white{.} \pos{q}{3}=\white{.}
\pos{q}{4}=\white{.} \pos{q}{5}=\black{.}
\pos{r}{2}=\white{.} \pos{r}{3}=\black{.}
\pos{r}{4}=\black{.} \pos{r}{5}=\white{.}
\pos{r}{6}=\white{.} \pos{r}{7}=\white{.}
\pos{s}{1}=\white{\square}
\pos{s}{2}=\black{\triangle}
\pos{s}{3}=\letter{a} \pos{s}{4}=\black{.}
\pos{s}{5}=\black{.}  \pos{t}{1}=\black{2}
\pos{t}{2}=\white{1}  \pos{t}{3}=\black{.}
\pos{t}{4}=\letter{b}
\begin{center}
  \mbox{\showdiagram  n-t:1-8 }
\end{center}
```

Рис. 8.13. Двухшаговое ко (выигрышное для черных).

случае различных ходов часто применяются частичные диаграммы. Поэтому пример, набранный с использованием команды `\showdiagram` и показанный на рис. 8.13, является типичным.

`\textblack{`*display*`}` `\textwhite{`*display*`}`

Изображения камней можно вставлять и в текст, воспользовавшись командами `\textblack` и `\textwhite`; эти команды имеют единственный аргумент в виде числа или специального символа (`\triangle` или `\square`), определяющий, что должно быть изображено на этом камне. Таким образом, команда `\textblack{5}` приводит к появлению в комментарии ❺.

8.3.1 Возможные проблемы

Изначально пакет go был предназначен для работы совместно с plain TEX'ом, поэтому в случае использования LATEX'а он иногда может конфликтовать с другими приложениями. Здесь перечислены наиболее вероятные проблемы и возможные способы их решения.

- Как уже упоминалось, в пакете go происходит переопределение команды LATEX'а `\symbol`. Поскольку новое определение вам не нужно (оно является всего лишь альтернативным именем для команды `\letter`), в случае необходимости вы можете восстановить исходное определение, задав после загрузки пакета `\renewcommand\symbol[1]{\char#1\relax}`.
- По тем же причинам вы не можете использовать пакет go совместно с классом letter.
- Внутри себя пакет переопределяет plain-TEX'овскую команду `\empty`. Это может привести к странным последствиям в совершенно неожиданных ме-

стах итогового документа, так как эту команду иногда используют другие пакеты.

- Не вставляйте данный пакет вместе с какими-нибудь другими внутрь одной декларации `\usepackage`, поскольку это может не сработать.

8.4 Нарды

В написанном Йоргом Рихтером пакете `bg` для изображения партий игры в нарды определены два LaTeX'овских окружения: `position` и `game`. Окружение `position` рисует доску целиком и, таким образом, это удобно для обсуждения задач, тогда как с помощью окружения `game` вы можете вводить каждый ход отдельно. В последнем случае расположение камней на доске сохраняется внутри окружения, что дает возможность в любой момент построить изображение текущей позиции.

По умолчанию дома обоих игроков расположены слева, причем дом белых находится сверху, а дом черных — снизу. В отличие от пакетов, обсуждавшихся до сих пор, поля на доске не имеют абсолютных координат, а нумеруются в соответствии с тем, ход какого из игроков наносится на доску, например, поле 24 на рис. 8.14 с точки зрения белых соответствует полю 1 с точки зрения черных и т. д. Ходы всегда выполняются в направлении от бóльших номеров к меньшим, а кости всегда находятся на правой части доски.

`\begin{position}` ... `\end{position}`

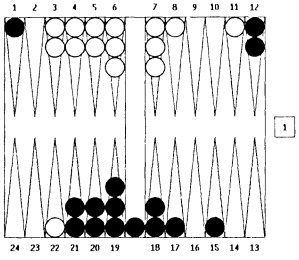
Окружение `position` создает пустую доску, на которой с помощью описываемых ниже команд располагаются камни. В него также входят команды, позволяющие настраивать некоторые детали внешнего вида доски. Изображение доски выводится, как только встречается команда `\end{position}`. На рис. 8.14 показан пример, в котором присутствуют различные команды, используемые в окружении `position`.

`\blackpoint{`p`}{`n`}` `\whitepoint{`p`}{`n`}`

Эти две команды используются для размещения камней на доске. Аргумент n обозначает количество размещаемых камней, а p — позицию, в которую их нужно поместить. Важно не забывать, что эти позиции нумеруются в порядке убывания, начиная с 24, относительно расположения дома каждого из игроков.

`\blackbar{`n`}` `\whitebar{`n`}`

Битые камни помещаются на бар (планку в середине доски). Эту операцию выполняют команды `\blackbar` и `\whitebar` с аргументом n, соответствующим числу камней, которое надо туда поместить.



8-4-1 Белые играют 3–2

```
\usepackage {bg}
\usepackage [cp866]{inputenc}
\usepackage [LCY]{fontenc}
\begin{position}
\smallboard
\whitepoint{3}{2}   \whitepoint{4}{2}
\whitepoint{5}{2}   \whitepoint{6}{3}
\whitepoint{7}{3}   \whitepoint{8}{1}
\whitepoint{11}{1}  \whitepoint{22}{1
\blackpoint{24}{1}  \blackpoint{13}{2
\blackpoint{10}{1}  \blackpoint{8}{1}
\blackpoint{7}{2}   \blackpoint{6}{3}
\blackpoint{5}{2}   \blackpoint{4}{2}
\blackbar{1}
\shownumbers \middlecube{1} \showcube
\whiteonmove
\boardcaption{Белые играют 3--2}
\end{position}
```

Рис. 8.14. Пример позиции в нардах.

`\blackcube{`*n*`}` `\whitecube{`*n*`}` `\middlecube{`*n*`}`
`\showcube` `\dontshowcube`

Указанные выше команды задают местоположение костей и число выпавших очков. По умолчанию кости находятся в средней позиции, т. е. ни у одного из игроков. Их можно сместить в сторону любого игрока, воспользовавшись командой `\whitecube` или `\blackcube`, аргумент *n* которой обозначает текущее число выпавших очков. В случае использования команды `\dontshowcube` изображение костей отсутствует.

`\smallboard` `\normalboard` `\bigboard`
`\fullboard` `\halfboard`
`\blackonmove` `\whiteonmove`
`\shownumbers` `\dontshownumbers` `\togglenumbers`
`\boardcaption{`*text*`}`

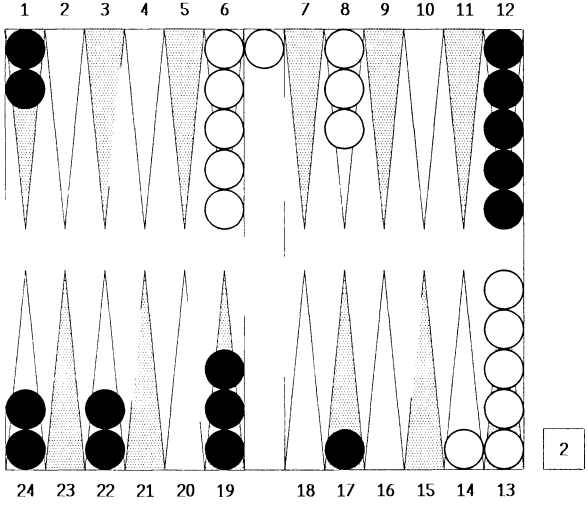
Существуют три команды, предназначенные для определения размера доски. По умолчанию используется `\normalboard`. Обычно выводится изображение доски целиком, но при обсуждении задач иногда бывает полезно нарисовать только ее половину. Это можно сделать с помощью команды `\halfboard`. Можно указать, чья очередь ходить (что влияет на нумерацию полей доски). Нумерацию полей можно удалить совсем (`\dontshownumbers`) или переключить на противоположную, т. е. показать ее с точки зрения другого игрока. В любом случае с помощью команды `\boardcaption` вы можете поместить под доской заголовок.

```
\usepackage {bg}
\usepackage [LCY]{fontenc}
\begin{game}{Черные}{Белые}
\normalboard  \whiteonmove
\move{64}{24-18, 18-14} \move{55}{6-1,6-1,8-3,8-3}
\textmove{Удвоение.} \textmove{Передача костей.} \takecube
\boardcaption{31-й ход, очередь черных}
\printboard
\end{game}
```

• Черные ○ Белые

1. ○ 64 : 24–18, 18–14
2. ○ Удвоение.

1. • 55 : 6–1*, 6–1, 8–3, 8–3
2. • Передача костей.



31-й ход, очередь черных 8-4-2

Рис. 8.15. Первые ходы партии в нарды.

`\begin{game}{`*Black*`}{`*White*`} ... \end{game}`

Окружение game используется для того, чтобы сопровождать комментариями ход игры. В отличие от окружения position, его начальным состоянием является не пустая доска, а доска, заполненная камнями в исходных позициях. Два его аргумента можно использовать для указания информации об игроках, например их имен или числа набранных очков. На рис. 8.15 показана позиция после нескольких ходов.

```
\move{die}{moves}
\printboard          \rawboard
```

Ходы вводятся с помощью команды `\move`. Значения аргумента *die* вводятся в виде двузначных чисел. Аргументы *moves* задаются в форме `x-y` и разделяются запятой, причем `x` и `y` обозначают начальную и конечную позиции, пронумерованные с точки зрения текущего игрока.

Все ходы записываются, а этапу вывода предшествует небольшая проверка на правильность цветов. В любой момент игры команда `\printboard` позволяет продемонстрировать текущую расстановку на доске на центрированной диаграмме с подписью, а команда `\rawboard` позволяет сконструировать изображение доски без подписи, заключенное внутрь бокса командой `\mbox`. Последний вариант можно применять даже в середине текстовой строки.

```
\textmove{text}
\takecube
```

Команда `\textmove` вместо описания хода выводит содержимое аргумента *text*, а затем переключается на другого игрока. Это можно использовать, например, в случае удвоения ходов, передачи хода или в том случае, когда один из игроков не имеет возможности сделать ход. Команда `\takecube` передает кости другому игроку и удваивает выпавшее количество очков.

```
\fullincr    \indentwhite        \showmoves
\halfincr    \dontindentwhite    \dontshowmoves
```

Выводимый командами `\move` материал можно настраивать несколькими способами. По умолчанию счетчик ходов после каждого хода увеличивается. Если применяется команда `\halfincr`, то счетчик увеличивается после каждого второго хода. При желании можно подавить отступ при печати ходов белых, а в случае использования команды `\dontshowmoves` вывод ходов подавляется полностью.

Помимо перечисленных команд, существуют также команды, действующие внутри окружения `position`, такие, как `\dontshowcube` и т. д.

8.5 Карточные игры

Символы ♣, ♠, ♡ и ♢ уже являются частью стандартного (LA)TEX'а: по каким-то причинам Дональд Кнут включил их в свои математические шрифты и сделал доступными с помощью команд `\clubsuit`, `\spadesuit`, `\heartsuit` и `\diamondsuit`. Все эти команды имеют длинные имена и доступны только в математических формулах, многократное использование которых в обычном тексте весьма мучительно. В такой ситуации наилучшим решением может

оказаться определение новых команд, что позволяет уменьшить количество набираемых символов, например,

```
\newcommand\club{\ensuremath{\clubsuit}}
\newcommand\diam{\ensuremath{\diamondsuit}}
\newcommand\heart{\ensuremath{\heartsuit}}
\newcommand\spade{\ensuremath{\spadesuit}}
```

Приведенные выше определения можно использовать как в формулах, так и в обычном тексте, поэтому ввод последовательности `\heart␣A` приводит к появлению ♡A. Добавление еще нескольких определений, подобных приведенным ниже, позволит вам описывать правила и комментировать ход партий любых игр, в которых используются традиционные игральные карты.

♠ A
♡ A
♢ A 9
♣ Q

```
\newcommand{\hand}[4]{%
 \begin{minipage}[t]{8em}
 \begin{tabbing}
  \spade{} \= #1\\ \heart \> #2\\
  \diam    \> #3\\ \club  \> #4
 \end{tabbing}
 \end{minipage}}
\hand{A}{A}{A 9}{Q}
```

8-5-1

Это определение располагает карты в последовательности, соответствующей покеру или бриджу. В других карточных играх старшей мастью обычно считается ♣, с учетом чего должно быть изменено данное определение.

8.6 Бридж

Используя обозначенные в предыдущем разделе идеи, Кес ван дер Лан создал LaTeX'овские макро и окружение `bidding`, которые предназначены для набора партий игры в бридж с комментариями в стиле, часто встречающемся в литературе по бриджу (van der Laan, 1989). В следующих разделах мы покажем команды, присутствующие в файле `bridge.tex`, и разработаем некоторые дополнительные команды для особых случаев.

8.6.1 Сдача карт

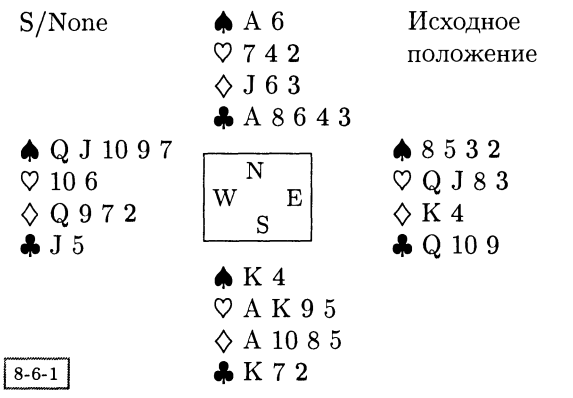
Расклад карт в литературе по бриджу часто представляется в виде диаграммы, на которой показаны положение и набор карт каждого игрока. Традиционно игрокам присваиваются имена четырех сторон света N (North — север), E (East — восток), S (South — юг) и W (West — запад), которые размещаются в центре рисунка, обозначающем игровой стол. Такой рисунок может быть создан командой `\crdima`.

`\crdima{`*dealer*`}{`*info*`}{`*north*`}{`*west*`}{`*east*`}{`*south*`}`

Первый параметр *dealer* содержит информацию о сдающем и о том, открыты ли карты; например запись `N/None` обозначает, что сдает «North» и все карты

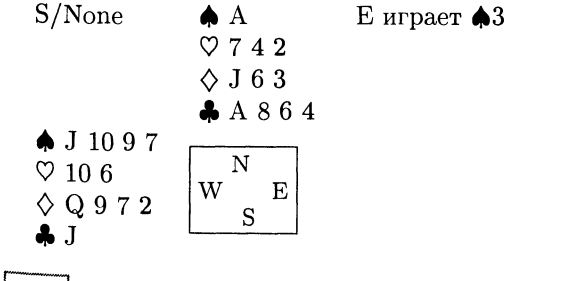
закрыты. Второй параметр *info* представляет собой текст, описывающий данную партию. Последние четыре аргумента содержат наборы карт всех игроков, перечисленные в порядке N, W, E, S.[1] Обычно все они содержат вызов команды `\hand`, но кроме этого существует возможность разместить в них другую или дополнительную информацию.

Затем общими усилиями обе команды создают диаграммы, похожие на приведенную ниже — исходное положение, демонстрирующее наборы карт каждого из игроков.




```
\input {bridge.tex}
\crdima{S/None}
   {\begin{tabular}[t]{l}
          Исходное\\положение
    \end{tabular}}
   {\hand{A 6}{7 4 2}{J 6 3}
         {A 8 6 4 3}}
   {\hand{Q J 10 9 7}{10 6}
         {Q 9 7 2}{J 5}}
   {\hand{8 5 3 2}{Q J 8 3}
         {K 4}{Q 10 9}}
   {\hand{K 4}{A K 9 5}
         {A 10 8 5}{K 7 2}}
```

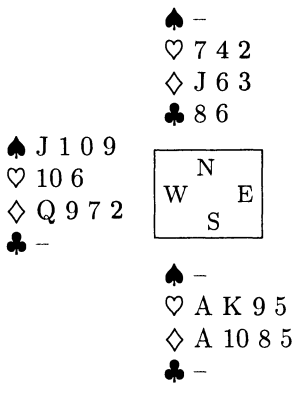
При обсуждении действий разыгрывающей стороны часто демонстрируются два набора карт, ваш собственный и «болвана». Этого можно добиться с помощью команды `\crdima` с пустыми аргументами для тех наборов, которые не демонстрируются. В приведенном ниже примере показана та же самая партия на ранней стадии.




```
\input {bridge.tex}
\crdima{S/None}
   {E играет \spade 3}
   {\hand{A}{7 4 2}{J 6 3}
         {A 8 6 4}}
   {\hand{J 10 9 7}{10 6}
         {Q 9 7 2}{J}}
   {}{}
```

[1] В статье (van der Laan, 1989) наборы карт перечислялись по часовой стрелке, т. е. N, E, S, W, но макро, которые хранятся в архиве CTAN в файле `bridge.tex`, действуют в другом порядке.

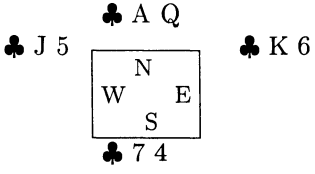
Обычно отсутствие какой-нибудь масти обозначается символом en-dash (т. е. двойным дефисом --, порождающим –). Это показано в следующем примере, представляющем ту же самую партию после того, как сыграли еще несколько карт.




```
\input {bridge.tex}
\crdima{}{}
 {\hand{--}{7 4 2}{J 6 3}
       {8 6}}
 {\hand{J 1 0 9}{10 6}
       {Q 9 7 2}{--}}
 {\hand{8 5}{Q J 8 3}
       {K 4}{--}}
 {\hand{--}{A K 9 5}
       {A 10 8 5}{--}}
```

8-6-3

При обсуждении путей продолжения игры часто демонстрируется только расклад карт определенной масти. В таком случае хотелось бы в аргументах команды \crdima избежать применения команды \hand, но, к сожалению, результат отличается от ожидаемого.



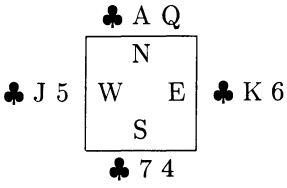

```
\input {bridge.tex}
\crdima{}{}{\club{} A Q}
  {\club{} J 5}{\club{} K 6}
  {\club{} 7 4}
```

8-6-4

В данном случае более приемлемые результаты дает использование окружения tabular. Первый аргумент указывает интересующую масть, остальные соответствуют четырем игрокам (перечисляемым в том же порядке, что и в команде \crdima).

```
\newcommand{\crdexa}[5]{{\renewcommand\arraystretch{1.2}%
 \begin{tabular}{l|@{}c@{}|l}
  \multicolumn{1}{c}{} & \multicolumn{1}{c}{#1 #2} \\ \cline{2-2}
                       &    N      &       \\
                 #1 #3 &W\hfill   E& #1 #4 \\
                       &    S      &       \\ \cline{2-2}
  \multicolumn{1}{c}{} & \multicolumn{1}{c}{#1 #5}
 \end{tabular}}}
```

Обратите внимание на использование команды `\multicolumn` для подавления вертикальных линий в первой и последней строках. Показанное определение приводит к следующему результату:




```
\crdexa{\club}{A Q}{J 5}{K 6}{7 4}
```

8-6-5

8.6.2 Торговля

Важной частью партии в бридж является начальная часть торговли, во время которой игроки определяют, кто будет разыгрывать контракт. Для записи последовательности заявок Кес ван дер Лан на основе стандартного LaTeX'овского окружения `tabbing` создал окружение `bidding`.

West	North	East	South
–	1♣	no	1♠
no	2♠	no	4♠
a.p.			

```
\input {bridge.tex}
\begin{bidding}
-- \> 1\club\> no \> 1\spade\\
no \> 2\spade\> no \> 4\spade\\
a.p.
\end{bidding}
```

8-6-6

При обсуждении теории торговли в литературе по бриджу такая последовательность заявок часто демонстрируется вместе с набором карт одного из игроков. Это может быть достигнуто благодаря следующему использованию описанных выше команд:

♠ 3 2
♡ K J 10 8 5 2
♢ Q 6 3
♣ K 3

West	North	East	South
–	1♢	1♡	no
2♡			

```
\input {bridge.tex}
\hand{3 2}{K J 10 8 5 2}
     {Q 6 3}{K 3}       \qquad
\begin{minipage}[t]{8em}
\begin{bidding}
-- \> 1\diam\> 1\heart \> no \\
2\heart
\end{bidding}
\end{minipage}
```

8-6-7

Альтернативное представление, в котором присутствуют только заявки двух партнеров, может быть получено путем определения простой команды `\bid`. В этом определении для представления набора карт одного из партнеров

используется команда \hand, а последовательность заявок выводится с помощью окружения tabular, поэтому на этот раз в пятом аргументе нам придется воспользоваться символом &.

♠ Q J 10 5
♡ A 10 9 6 3
◇ A 5 2
♣ 3

свои	партнера
	1♣
1♡	1♠
4♠	

```
\input {bridge.tex}
\newcommand\bid[5]{\hand{#1}{#2}{#3}{#4}%
          \qquad   \begin{tabular}[t]{ll}%
                     свои & партнера \\ #5
                   \end{tabular}}
\bid{Q J 10 5}{A 10 9 6 3}{A 5 2}{3}
 { & 1\club \\ 1\heart & 1\spade \\
   4\spade }
```

8-6-8

8.7 Кроссворды

В напечатанной в *TUGboat* статье (Hamilton Kelly, 1990) Брайан Хамильтон Келли представил набор LaTeX'овских макро, предназначенных для построения кроссвордов. Его система задумана таким образом, чтобы обеспечивать правильное сочетание элементов «сетки». Например, на рис. 8.16 показан кроссворд[1], который следует заполнить с помощью данных ключей (его решение можно найти на рис. 8.17 на с. 359).

\begin{crossword}{*size*}{*show*}

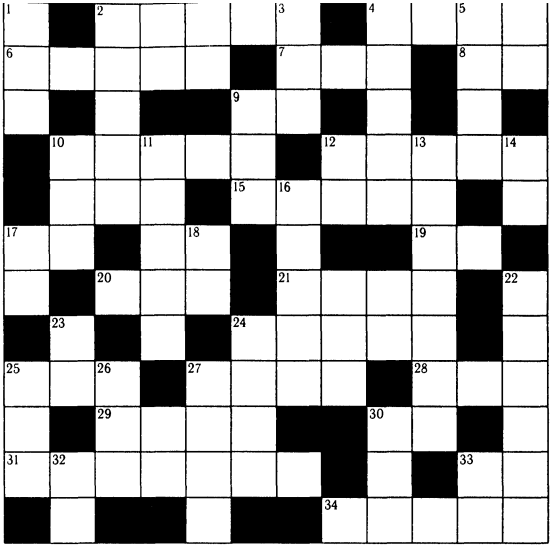
Используемый для набора кроссворда материал заключается в окружении crossword. Его первый параметр *size* определяет количество строк и столбцов (таким образом, создаваемые этим пакетом кроссворды всегда имеют квадратную форму).

Значение параметра *show* определяет, следует ли демонстрировать решение или нет. На этапе публикации кроссворда этому параметру обычно присваивается значение n или N, которые оставляют клетки кроссворда пустыми. Если указывается значение y или Y, то кроссворд рисуется вместе с решением, что иногда бывает полезно на стадии его создания. Любые другие значения приведут к тому, что система в режиме диалога постарается выяснить, выводить или нет решение. Внутри окружения crossword ключи вводятся с помощью команды \clue:

\clue{*no*}{*dir*}{*col*}{*row*}{*answer*}{*text*}{*help*}

Первый аргумент *no* определяет номер данного ключа, который помещается в первой клетке сетки и в перечне ключей. Направление слова внутри сетки

[1] Мы благодарим Герда Нойгебауера за создание этой головоломки на основе указателя к данной книге. Ради усложнения задачи добавленные авторами ключи иногда полностью скрывают первоисточник.



По горизонтали

2 Расстояние между метками узла дерева и самим узлом в PSTricks (5)
4 Современная замена ножниц и клея (4)
6 Unicode'овская разновидность TEX'а (5)
7 . . . , что вы хотели знать, но не решались спросить (3)
8 Графический ключ, нуждающийся в четырех числах (2)
10 Носит название bb в схеме наименования шрифтов Карла Берри (5)
12 Способ превратить ваши страницы в миниатюрные изображения (5)
15 Вы можете сделать это с боксом, но это — неправильный LATEX (5)
19 В LATEX'е обозначает $\wp$; в остальных случаях может обозначать текстовый процессор (2)
20 Результат работы TEX'а (3)
21 Язык, чье название, возможно, должно было бы состоять из пяти букв, но который был усовершенствован для Unix'а (4)
24 Это — не Intel (5)
25 Неверно записанная ссылка (3)
27 Проверка вашего знания LATEX'а: $\prec$ (4)
28 Метка сигнальной линии (3)
29 Другое имя команды разработчиков проекта LATEX3, встречающееся в c.t.t. (4)
30 Один из способов получения диеза в MusiXTEX (2)
31 Рисунок или схема, предназначенная скорее для объяснения, чем для описания реального внешнего вида (7)
33 В дюйме их 72.27 (2)
34 см. **1 по вертикали** (5)

По вертикали

1 & 34 по горизонтали Величайший знаток TEX'а (3,5)
2 Готовый к запуску вариант TEX'а в ОС Unix (5)
3 Мечта начинающего игрока в гольф (3)
4 Название стиля документов в LATEX 2_ε (5)
5 Двойная перекладина в MusiXTEX (4)
9 Или или — на языке математики (3)
10 Немецкое пиво (3)
11 Сохранение координат (PSTricks) (5)
12 Аппроксимация номера TEX'овской версии (2)
13 Оператор в PostScript'е (7)
14 Функция вероятности (2)
16 Небесный посланник с опечаткой (5)
17 Как вы получаете Å? (2)
18 ξ (2)
22 В LATEX'е есть жесткие и мягкие (6)
23 Амур ими пользуется, а Xy-pic их вызывает (2)
24 Длина отрезка в месте, где соединительная линия подходит к первой вершине (4)
25 Файлы, в которых содержится LATEX'овская документация, описывающая шрифты (3)
26 η — не говорите, что это для вас китайская грамота (3)
27 $\perp$, а также первые буквы вечно продолжающегося (4)
30 Мы изобразили это в главе 4 (3)
32 TEX'овское обозначение дюйма (2)
33 Что возглавляет Лула? (2)

8-7-1

Рис. 8.16. Образец кроссворда, который вам предлагается заполнить.

задается параметром *dir*, принимающим значение A, обозначающее *across* (по горизонтали), или D, обозначающее *down* (по вертикали). Следующие два параметра *col* и *row* указывают положение первой клетки, с которой начинает записываться решение. Столбцы нумеруются слева направо, строки — сверху вниз.

Аргумент *answer* представляет собой ответ на вопрос, т. е. расшифровку ключа, текст которого записывается в аргументе *text*. Этот ответ должен быть целиком набран прописными буквами, тогда как параметр *text* может содержать произвольный LaTeX'овский текст, включая команды, и располагаться под сеткой вместе с номерами ключей в двух списках: один содержит ключи «по горизонтали», а второй — «по вертикали». Последний параметр *help* выводит заключенную в скобки вспомогательную информацию. Часто он используется для того, чтобы указать количество букв в ответе, или, если ответ состоит более чем из одного слова, он может содержать что-нибудь наподобие `see␣2d`.

В качестве примера ниже представлен один из ключей кроссворда с рис. 8.16 (за исключением ответа; вместо него проставлены знаки вопроса, чтобы не лишать вас возможности решить кроссворд самостоятельно).

```
%  \clue{29}{A}{3}{10}{????}{Другое имя команды разработчиков проекта
%  \LaTeX3, встречающееся в c.t.t.}{4}%
\clue{29}{A}{3}{10}{????}{Another name for the \LaTeX3
                          project team on c.t.t.}{4}
```

Некоторые из перечисленных аргументов могут содержать весьма сложные конструкции. Например, если ответ состоит из более чем одного слова, вы можете разместить информацию о положении клеток, в которых начинаются другие слова, внутри аргумента *no*, отделив их номера друг от друга пробелами или другими символами, отличными от цифр. В этом случае вы должны представить длину всех составляющих решение слов в виде разделенного запятыми перечня, помещенного внутрь аргумента *help*. После этого остальные слова решения могут вводиться с пустым аргументом *text* и расположенной внутри аргумента *help* ссылкой на ключ в описании первого слова.

```
%  \clue{1 \& 34 по горизонтали}{D}{1}{1}{???}{Величайший
%  знаток \TeX{}'a}{3,5}%
%  \clue{34}{A}{8}{12}{?????}{см. \textbf{1 по вертикали}}{5}%
\clue{1 \& a34}{D}{1}{1}{???}{Grand wizard of \TeX{}}{3,5}
\clue{34}{A}{8}{12}{?????}{see \textbf{1d}}{5}
```

Если аргумент *help* абсолютно пуст, то этот ключ не входит в список ключей, расположенный под кроссвордом, однако номер ключа в сетке кроссворда появится (запись и проверка решения выполняются автоматически). Такая возможность бывает полезной в случае многосложных решений, занимающих соседние клетки, где дополнительные ключи обычно удаляются.

После выполненной Франком Миттельбахом адаптации этого пакета к LaTeX 2_ε заголовки списков ключей ACROSS и DOWN стали настраиваемыми

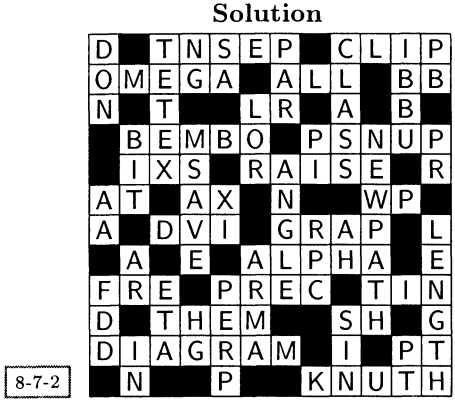


Рис. 8.17. Ответы к кроссворду на рис. 8.16.

с помощью команд `\ACROSStext` и `\DOWNtext`. Например, для создания немецкого кроссворда вам следует ввести следующие определения:

```
\renewcommand\ACROSStext{Waagrecht}
\renewcommand\DOWNtext{Senkrecht}
```

Задание второго аргумента окружения `crossword` для демонстрации решения обычно бывает полезным только на стадии составления, поскольку список ключей под кроссвордом сохраняется. Если вы хотите подавить вывод этого списка и напечатать только решение «кроссворда, опубликованного в прошлом месяце», воспользуйтесь вариантом окружения со звездочкой `crossword*`, который создает сетку меньшего размера для демонстрации решения.

`\begin{crossword*}{`*size*`}{`*title*`}`

Второй параметр в этом окружении позволяет вам помещать над сеткой заголовок, скажем, для того, чтобы сообщить читателям, решение какого кроссворда здесь представлено. В теле просто окружения поместите копии команд `\clue` исходного кроссворда и тогда это приведет к появлению решения, сходного с показанным на рис. 8.17.

8.7.1 Последние достижения

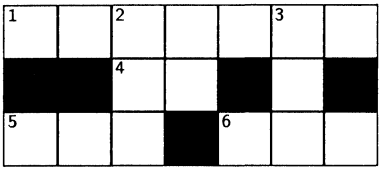
Незадолго до момента сдачи этой книги в печать в архиве CTAN появился новый LaTeX'овский пакет для создания кроссвордов. Написанный Гердом Нойгебауером пакет `cwpuzzle` применим для создания классических кроссвордов, числовых головоломок и заполненных головоломок. (В числовых головоломках все буквы заменяются числами и задача состоит в том, чтобы определить, какое число обозначает какую букву. В заполненных головоломках список клю-

чей заменяется списком всех присутствующих в ней слов, а целью является определение местоположения слов.)

Во всех случаях головоломка создается с помощью окружения Puzzle:

\begin{Puzzle}{*columns*}{*rows*}

Аргументы *columns* и *rows* задают количество клеток по горизонтали и по вертикали. Внутри окружения каждая строка описывает содержимое одного одного ряда клеток. Например,




```
\usepackage {cwpuzzle}
\begin{Puzzle}{7}{3}
|[1]T | A |[2]B | U |   L |[3]A | R |.
|   * | * |[4]I | M |   * |   X | * |.
|[5]D | E |   G | * |[6]D |   E | T |.
\end{Puzzle}
```

8-7-3

Клетки разделяются вертикальными линиями, а окончание строки обозначается точкой. Содержимым клетки может быть символ *, обозначающий черное поле, или одна буква, которой, возможно, предшествует заключенное в квадратные скобки число, обозначающая начало слова. В числовых головоломках вам придется помещать такое число в каждой клетке, а в заполненных они не нужны вовсе. Наконец, внутрь клетки вы можете поместить {}. В этом случае данная клетка останется совершенно пустой, даже не будет окружена рамкой. Это дает вам возможность конструировать головоломки непрямоугольной формы или оставлять внутри головоломки свободное место для рекламы или рисунка.

\begin{PuzzleClues}{*heading*}

Воспользуйтесь этим окружением для набора списка ключей в случае составления классического кроссворда. Отдельные ключи вводятся в теле окружения с помощью команды \Clue, как показано в следующем примере:

```
\Clue{1}{TABULAR}{Environment to produce tables}
```

В отличие от окружения crossword, вводимый вами материал *не проверяется* на корректность, а используется лишь для набора списка ключей — на самом деле в настоящее время второй аргумент вообще не используется.

\begin{PuzzleWords}{*number*}

При создании заполненной головоломки применяйте это окружение столько раз, сколько потребуется. Аргумент *number* указывает длину слов из данного

списка, а внутри тела окружения команда \Word применяется для нумерации всех слов этой длины, например в случае слов длины два мы пишем:

```
\begin{PuzzleWords}{2}\Word{IM}\Word{UM}\end{PuzzleWords}
```

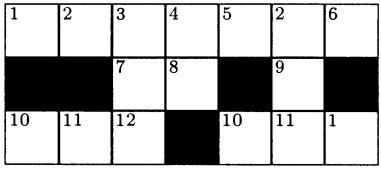
В головоломках подобного рода обычно перечисляются все «слова», даже если они не имеют смысла.

Наконец, следующие команды можно использовать при создании ключей для числовой головоломки:

\PuzzleLetters{*letters*} \PuzzleNumbers{*letters*}

С помощью команды \PuzzleLetters указывается список всех букв *letters*, встречающихся в головоломке (предпочтительно в алфавитном порядке), а команда \PuzzleNumbers создает список боксов, в котором читатель мог бы записать решение.

```
\begin{Puzzle}{7}{3} ... \end{Puzzle}
\renewcommand\PuzzleLettersText{Используются следующие буквы: }
\PuzzleLetters{ABDEGILMRTUX} \medskip\par
\PuzzleNumbers{TABULRIMXDEG}
```



Используются следующие буквы: ABDEGILMRTUX

1 2 3 4 5 6 7 8 9 10 11 12

8-7-4

\PuzzleSolution \PuzzleUnsolved

Эти две команды определяют то, как осуществляется набор любых последующих головоломок. Команда \PuzzleSolution представляет головоломку с внесенными решениями, а (используемая по умолчанию) команда \PuzzleUnsolved выводит пустую сетку вместе со списком ключей. В документации к пакету описаны несколько дополнительных параметров, влияющих на внешний вид рисунка и модифицирующих создаваемые по умолчанию строки.

Мир цвета

Для многих людей цвет является неотъемлемым элементом графики. Все современные интерактивные графические пакеты способны раскрашивать линии, заполнять цветом заданные области и т. д., а все стандартные растровые графические форматы, такие, как GIF (Graphics Interchange Format — формат обмена графикой), JPEG (Joint Photographic Experts Group — объединенная группа фотоэкспертов), PBM (Portable Bitmap — мобильное растровое представление), TIFF (Tagged Image File Format — формат маркированных графических файлов) и BMP (Windows Bitmap — растровое представление ОС Windows), наряду с инкапсулированным PostScript'ом поддерживают многоцветность. Таким образом, если с помощью графического пакета вы создаете какое-нибудь изображение и импортируете его в LaTeX'овский документ с помощью средств, описанных в гл. 2, то у вас не должно возникнуть никаких проблем при условии, что ваше устройство для просмотра или печати поддерживает многоцветность. Однако вам следует кое-что знать о том, как кодируется цвет и какую систему представления цвета вы используете. Эти вопросы обсуждаются в первой части данной главы.

Если вы создаете графический материал средствами самого LaTeX'а или просто хотите раскрасить текст, то вам понадобятся специальные возможности как LaTeX'а, так и используемого вами драйвера. Основная часть этой главы посвящена описанию стандартного LaTeX'овского пакета color, возможностей которого по нашему мнению достаточно, чтобы удовлетворить практически любым потребностям, и который функционирует совместно с большинством других пакетов. Этот пакет, однако, не умеет решать такую задачу, как раскрашивание таблиц, и поэтому мы рассмотрим еще и специальный пакет colortbl.

Пользователям LaTeX'а цвет часто нужен для создания прозрачных пленок, используемых во время презентаций. Конечно, стандартный пакет color можно использовать вместе со стандартным классом slides, но некоторую часть главы мы посвятим изучению более сложного класса seminar и демонстрации его возможностей.

В конце этой главы мы рассмотрим практические стороны профессиональной цветной печати и цветоделения. Это рассмотрение предназначено для того, чтобы познакомить вас с теми вопросами, которые вам придется обсуждать с типографскими работниками, а не для того, чтобы полностью обучить вас искусству цветоделения.

Поскольку цветная печать по-прежнему остается дорогим удовольствием, в этой книге мы ограничились единственной группой цветных иллюстраций. Большинство рисунков напечатано в тексте, причем цветовые эффекты на них обозначены оттенками серого, а затем некоторые примеры повторены на цветных вклейках. Мы указываем в тексте те случаи, когда читатель может ознакомиться с полноцветной версией.

9.1 Введение в цвет

Вам следует думать о цвете в документе как об инструменте, а не техническом приеме, предназначенном только для того, чтобы страницы выглядели «симпатичнее». Французский художник Эжен Делакруа написал (Duplan and Jauneau, 1986):

> «Цвет в наибольшей степени является той составной частью живописи, которая обладает магическим даром, в то время как тема, форма и линия в первую очередь касаются рассудка, цвет не имеет никакой связи с разумом; он властвует над ощущением и покоряет ваши чувства».

В этом предложении очень точно характеризуется та роль, которую цвет играет при построении визуального образа. Выбирая нужный цвет, печатник или художник может увеличить силу воздействия образа, облегчая понимание заложенного в него послания.

9.1.1 Теории цвета

С доисторических времен цвет играл важную роль в зрительной коммуникации, о чем явно свидетельствуют наскальные изображения животных, найденные в пещерах Альтамира или Ласкаукс. Египетская, ассирийская и другие древние культуры Ближнего Востока, а позднее финикийцы знали, как делаются краски и красители, и любили раскрашенные камни. Но только греческий философ Аристотель стал в IV веке до н. э. первым, кто попытался «понять» цвет, изучая комбинации цветов в разноцветных стеклянных сосудах. Для объяснения своих наблюдений он постулировал, что каждый цвет является комбинацией белого и черного, темные цвета возникают благодаря отражению света от окрашенной поверхности. Эта интерпретация, в которой цвета упорядочены вдоль прямой линии от самого светлого до самого темного, начиная с белого и заканчивая черным, просуществовала почти 2200 лет. Только в XVII в. благодаря экспериментам Исаака Ньютона со стеклянными призмами была

создана спектральная теория цвета. Ньютон упорядочил цвета на замкнутом округлом кольце. Это представление используется до сих пор.

Иоганн Гете, бóльшую часть своей жизни посвятивший изучению физики и химии, особенно сильно интересовался цветом. Несмотря на то что сегодня его теория цвета больше не признается, его мнение по этому вопросу послужило основой некоторого существенного прогресса в данной области. Английский доктор Томас Юнг в 1801 г. разработал трехцветную теорию света. Дальнейшим ее развитием занимался Герман фон Гельмгольц, поэтому теория трехцветного зрения называется законом *Юнга—Гельмгольца*. Работы Максвелла, Гельмгольца и Грассмана положили начало науке колориметрии. Законы сочетаний света разных цветов были впервые сформулированы Грассманом (1853 г.), который первым наглядно показал связь между светом и цветом. Он обобщил свои открытия в виде трех теорем, называемых *законами Грассмана*:

(а) Цвета, полученные путем *аддитивного* смешения цветов, полностью определяются своими трехцветными компонентами и не зависят от спектрального состава (теорема о *смешении цветов*).

(б) Для полного описания цвета необходимы и достаточны три величины (теорема *аддитивности*).

(в) В условиях дневного освещения цветовое восприятие не зависит от интенсивности освещения (теорема *пропорциональности*).

Недавние исследования и эксперименты обобщили наши знания в области цвета (подробнее см. Déribéré, 1993; Gerritsen, 1988; Nemcsics, 1993).

9.1.2 Системы представления цвета

Сегодня мы знаем несколько способов описания цветов, выбор каждого из которых зависит от сферы применения, например аддитивное или субтрактивное смешение цветов, теория цветового восприятия, телевидение, графическое искусство, красильное дело, полутона. В частности, PostScript подразделяет цветовые пространства на три категории:

- *Аппаратные цветовые пространства*: цвета или оттенки серого явно описываются в тех единицах, которые распознает устройство вывода. Примерами такой модели служат RGB (***R**ed, **G**reen, **B**lue — Красный, Зеленый, Синий*) и модификация для телевидения YIQ, CMYK (***C**yan, **M**agenta, **Y**ellow, **Bl**ack — Голубой, Пурпурный, Желтый, Черный*) и HSB, еще называемый HSV (***H**ue* (*Оттенок*), ***S**aturation* (*Насыщенность*), ***B**rightness* (*Яркость*) или ***V**alue* (*Величина*)). Взаимосвязь моделей RGB и HSB см. на цветных вклейках VII и VIII.
- *Цветовые пространства CIE*: цвета описываются аппаратно-независимым образом. В 1931 г. *Commission Internationale de l'Éclairage* (CIE) (Международная комиссия по свету) создала колориметрическую систему, в которой все цвета описывались однозначным и объективным способом. В 1964

(CIEUCS) и 1976 (CIELAB/CIELUV) годах в модель CIE были внесены изменения, корректирующие некоторые недочеты первоначальной версии.

- *Специальные цветовые пространства*: применяются для специальных целей, таких, как растровые изображения, цветные карты и цветоделение.

Вводный материал, посвященный применению цветов в PostScript'e, можно найти в Kunkel (1990) и McGilton and Campione (1992). Подробное описание всех упомянутых моделей и алгоритмов преобразования одного пространства в другое выходит за рамки данной книги. Более подробное изложение можно найти в Adobe Systems (1990), Foley et al. (1990), Jackson et al. (1994) и Salmon and Slater (1987). В Gerritsen (1988) все представленные цветовые модели разбиты на группы (см. также Nemcsics, 1993). Мы используем модели RGB и CMYK, поскольку они широко применяются в типографской и компьютерной индустриях.

Аддитивное цветовое пространство RGB

В пространстве RGB все цвета получаются путем суперпозиции (сложения) трех *первичных* компонентов, определенных CIE как красный (700 нм), зеленый (546.1 нм) и синий (435.8 нм) цвета. Для получения такого цвета электроны в электронно-лучевой трубке возбуждают матрицу элементов, изготовленную из фосфоров трех сортов, которые испускают свет с соответствующей длиной волны. Комбинация (сложение) этих источников и дает необходимый цвет (см. цветную вклейку VII).

Субтрактивное цветовое пространство CMYK

В типографском деле используются не первичные цвета модели RGB, а дополнительные к ним: голубой, пурпурный и желтый. Причина этого заключается в том, что краски «вычитают» свои дополнительные цвета из падающего на поверхность белого света, например, голубая краска поглощает красную составляющую белого света и, таким образом, в терминах аддитивных первичных цветов, голубой оказывается белым минус красный цвет, т. е. синий плюс зеленый. Аналогично, пурпурная краска поглощает зеленую составляющую и соответствует сочетанию красного и синего цветов, а желтая краска, которая поглощает синий цвет, соответствует сочетанию красного и зеленого (см. цветную вклейку VII). На практике в типографской индустрии применяется процесс «удаления цветной краски», в ходе которого добавляется четвертый цвет, а именно черный, для получения оттенка более темного, чем тот, который получается путем смешения красок всех трех цветов. Эта цветовая модель называется моделью CMYK, где последняя буква «K» обозначает черную составляющую. Учет влияния четвертой компоненты на формирование цвета в этой модели происходит следующим образом:

- *подсчет величины черной составляющей*: она равна минимуму компонент `C`, `Y` и `M`, потому что это значение соответствует тому, насколько краски всех трех оттенков перекрываются для получения черного цвета: `K=min(C,Y,M);`

- *удаление цветной краски*: для устранения той доли черного цвета, которая порождается использованием черной краски, полученное значение черной составляющей вычитается из остальных трех компонент C, M и Y: `C = C-K; Y = Y-K; M = M-K`.

9.1.3 Символические значения цвета

Значение отдельных цветов, как и символов, сильно различается в разных культурах. На цветной вклейке IX сравниваются символические значения некоторых цветов в разных культурах (на основе Greenwood (1993)). Например, в Соединенных Штатах Америки, как и в большинстве западных культур, красный цвет означает опасность, а в Индии этот цвет символизирует жизнь и творческое начало. Создавая плакат, обложку книги или интерфейс программы, важно принимать во внимание эти различия.

Чтобы уменьшить возможную путаницу, международные группы стандартизации отобрали цвета, имеющие общепринятое значение. Например, в случае знаков дорожного движения красный цвет является символом опасности, а зеленый обозначает медицинские службы или сообщает, что путь свободен. В подобном контексте синий, белый и черный цвета служат только для вспомогательных целей, т. е. используются для нанесения информации на дорожное покрытие, в зонах отдыха и т. д. Аналогичные стандарты разработаны при прокладке труб и в элсктротехнике.

Имеющий общепринятое значение цвет может помочь донести нужную мысль и сыграть важную психологическую роль в создании требуемой атмосферы или передаче культурного, религиозного или политического послания. Однако стоит отметить, что любая человеческая сфера деятельности, профессия или группа со сходными интересами даже в рамках одной и той же культуры обладает собственным «цветовым» жаргоном. Поэтому вам следует всегда помнить о возможных побочных эффектах использования цвета в любом конкретном контексте.

9.1.4 Цветовая гармония

Цветовая гармония представляет собой приятное для глаза сочетание цветов. На эту тему написаны горы книг, но получаемые в них выводы часто противоречат друг другу. Этому легко найти объяснение (Judd and Wyszecki, 1963):

- Цветовая гармония относится к области индивидуальной эмоциональной реакции на приятное и неприятное, и даже один и тот же человек с течением времени может изменять свои предпочтения, например, из-за того, что старые сочетания цветов могут наскучить, а часто встречающиеся новые комбинации могут начать нравиться.
- Цветовая гармония зависит от абсолютных размеров областей, занимаемых цветами, а также от общего дизайна и самих цветов. Например, раду-

ющая глаз мозаика может стать весьма неприглядной, будучи увеличенной в десять раз.

- Цветовая гармония зависит от относительных размеров областей, а также от покрывающих их цветов.
- Цветовая гармония зависит от формы элементов, а также их цветов.
- Цветовая гармония зависит от значения или интерпретации дизайна, а также от выбранных цветов. Очевидно, что цветовая гармония для портретиста сильно отличается от цветовой гармонии в абстрактном дизайне или типографском деле.

Тем не менее все же интересно попытаться сформулировать несколько принципов построения цветовых гармоний.

- Цветовая гармония получается в результате сопоставления целенаправленно выбранных цветов из набора узнаваемых и эмоционально приятных цветов.
- Сравнивая две сходные последовательности цветов, наблюдатель считает более гармоничной ту, которая привычна.
- Гармоничными считаются группы цветов, которые кажутся обладающими сходными характеристиками или особенностями.
- Цвета воспринимаются гармоничными, только если схема выбора данной комбинации определяется однозначным образом.

Экспериментально было установлено, что человеческий глаз предпочитает такие комбинации, в которых основные цвета уравновешиваются своими дополнительными цветами, а наше восприятие цвета меняется в зависимости от его окружения. На цветной вклейке X показан эффект *насыщенности* или *поглощения* трех основных цветов в присутствии белого (крайний левый столбец), черного (второй столбец), дополнительного (третий столбец) цветов и темного оттенка самого первичного цвета (крайний правый столбец).

В работе Иттена (Itten, 1974) для объяснения теории цвета применяется модель, основанная на цветовом гармоническом кольце, которое поделено на двенадцать равных частей (см. цветную вклейку XI). Оно содержит три основных цвета: желтый, красный и синий, удаленные друг от друга на 120°. Их дополнительные цвета: фиолетовый, зеленый и оранжевый, еще называемые вторичными, расположены диаметрально противоположно соответствующим первичным цветам. Кольцо содержит еще шесть цветов, промежуточных между каждым первичным и соседними вторичными. Это гармоническое цветовое кольцо является упрощением. На самом деле, все возможные цвета можно разместить на поверхности сферы, на экваторе которой расположились бы гармонические цвета, на северном полюсе — белый цвет, а на южном — черный. Таким образом, при перемещении от экватора к южному (северному) полюсу можно получить более темные (светлые) оттенки данного цвета. Из этого также следует, что для каждой точки на цветовой сфере существует диаметрально противоположная точка с обратными характеристиками, напри-

мер светло-зеленовато-синий цвет противоположен темно-оранжево-красному. Накопленный художниками за столетия опыт свидетельствует, что несколько простых базовых правил позволяют живописцам конструировать цветовые гармонии для своих работ. Ниже мы обсудим некоторые из них, следуя изложению Иттена.

Двуцветные гармонические комбинации

Двуцветные гармонии состоят из взаимодополняющих цветов, расположенных в диаметрально противоположных точках цветовой окружности (сферы), например, ими являются пары красный—зеленый, синий—оранжевый и почти бесконечное число вариантов, построенных с помощью сферы.

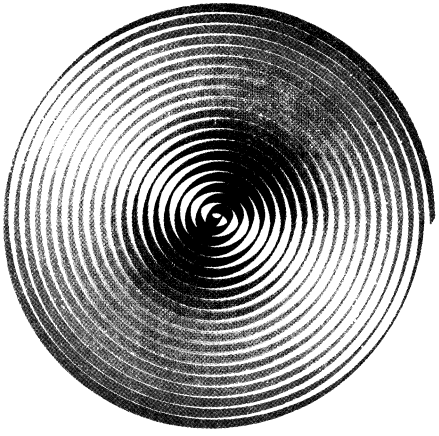
Трехцветные гармонические комбинации

Если внутри цветовой окружности построить равносторонний треугольник, то оказавшиеся в его вершинах цвета составят трехцветную гармонию. Самая фундаментальная тройка (желтый, красный, синий) хорошо известна во всех видах искусства, в типографском деле и рекламной индустрии благодаря своей эффектности и возможности использования в самых разнообразных узорах и форматах, а также в любых сочетаниях светлых и темных оттенков. Вторичная цветовая тройка (фиолетовый, зеленый, оранжевый) тоже часто применяется и обладает сильным воздействием. Существуют и другие гармонические тройки, и вы можете их построить путем замены равностороннего треугольника равнобедренным или работая на цветовой сфере и комбинируя светлые и темные варианты. В качестве специального случая вы можете совместить одну вершину треугольника с белым цветом (северный полюс) для создания гармонии (белый, темно-зеленовато-синий, темно-оранжево-красный) или с черным цветом (южный полюс) для создания гармонии (черный, светло-зеленовато-синий, светло-оранжево-красный).

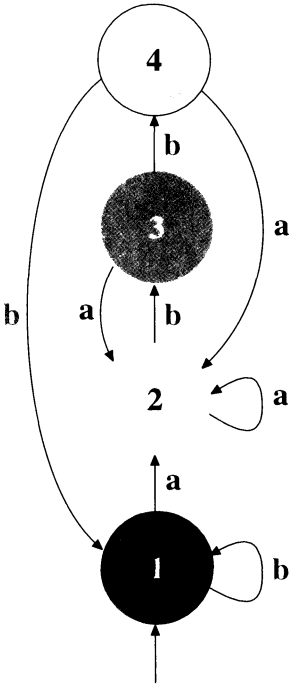
Четырехцветные гармонические комбинации

Вы можете построить четырехцветную гармонию, взяв цвета, которые расположены в углах квадрата, вписанного в цветовую окружность, например четверку (желтый, оранжево-красный, фиолетовый, зеленовато-синий). Также можно использовать прямоугольник, объединяющий две пары взаимодополняющих цветов.

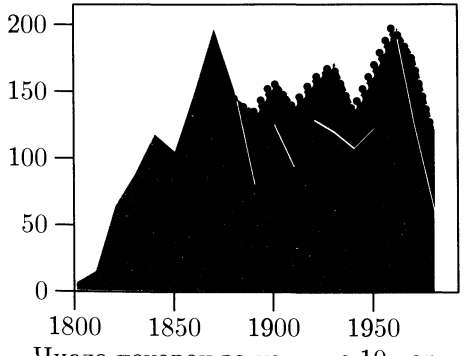
Гармонии более высокого порядка (например, шестицветные) тоже легко получить с помощью геометрических моделей подобного типа, реализованных на цветовой окружности или сфере. Однако обратите внимание, что каждой комбинации присущи свои собственные характеристики и набор основополагающих правил, и только по мере накопления опыта станет ясно, какая из многочисленных гармоний наилучшим образом подходит для данного приложения.



(a)

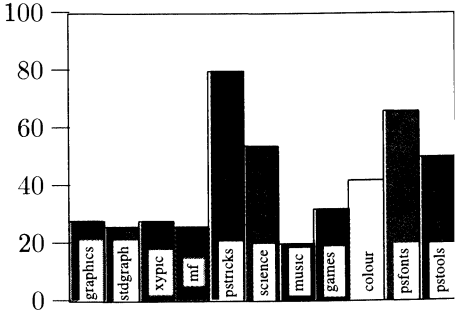


(b)



Число похорон за каждые 10 лет
($n \approx 4300$)

(c)



(d)

Цветная вклейка I. Примеры в METAPOST'е.

мер светло-зеленовато-синий цвет противоположен темно-оранжево-красному. Накопленный художниками за столетия опыт свидетельствует, что несколько простых базовых правил позволяют живописцам конструировать цветовые гармонии для своих работ. Ниже мы обсудим некоторые из них, следуя изложению Иттена.

Двуцветные гармонические комбинации

Двуцветные гармонии состоят из взаимодополняющих цветов, расположенных в диаметрально противоположных точках цветовой окружности (сферы), например, ими являются пары красный—зеленый, синий—оранжевый и почти бесконечное число вариантов, построенных с помощью сферы.

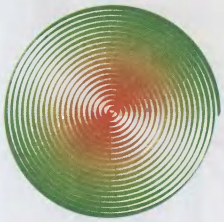
Трехцветные гармонические комбинации

Если внутри цветовой окружности построить равносторонний треугольник, то оказавшиеся в его вершинах цвета составят трехцветную гармонию. Самая фундаментальная тройка (желтый, красный, синий) хорошо известна во всех видах искусства, в типографском деле и рекламной индустрии благодаря своей эффектности и возможности использования в самых разнообразных узорах и форматах, а также в любых сочетаниях светлых и темных оттенков. Вторичная цветовая тройка (фиолетовый, зеленый, оранжевый) тоже часто применяется и обладает сильным воздействием. Существуют и другие гармонические тройки, и вы можете их построить путем замены равностороннего треугольника равнобедренным или работая на цветовой сфере и комбинируя светлые и темные варианты. В качестве специального случая вы можете совместить одну вершину треугольника с белым цветом (северный полюс) для создания гармонии (белый, темно-зеленовато-синий, темно-оранжево-красный) или с черным цветом (южный полюс) для создания гармонии (черный, светло-зеленовато-синий, светло-оранжево-красный).

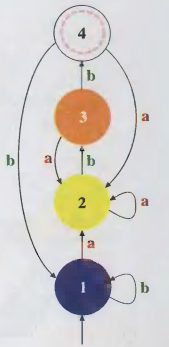
Четырехцветные гармонические комбинации

Вы можете построить четырехцветную гармонию, взяв цвета, которые расположены в углах квадрата, вписанного в цветовую окружность, например четверку (желтый, оранжево-красный, фиолетовый, зеленовато-синий). Также можно использовать прямоугольник, объединяющий две пары взаимодополняющих цветов.

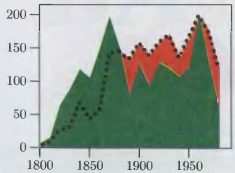
Гармонии более высокого порядка (например, шестицветные) тоже легко получить с помощью геометрических моделей подобного типа, реализованных на цветовой окружности или сфере. Однако обратите внимание, что каждой комбинации присущи свои собственные характеристики и набор основополагающих правил, и только по мере накопления опыта станет ясно, какая из многочисленных гармоний наилучшим образом подходит для данного приложения.



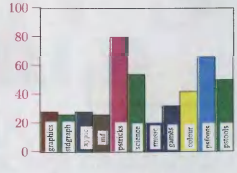
(a)



(b)



(c)

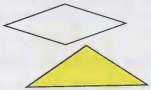


(d)

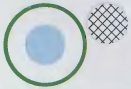
Цветная вклейка I. Примеры в METAPOST'е.

(a)

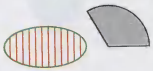
(b)



(c)

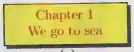


(d)



(e)

(f)

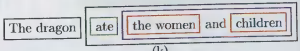


(g)

(h)

(i)

(j)



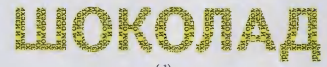
(k)

Цветная вклейка II. Примеры на PSTricks (1).

(a)

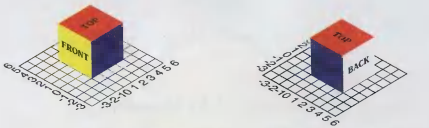
(b)

(c)



(d)

Цветная вклейка III. Примеры на PSTricks (2).



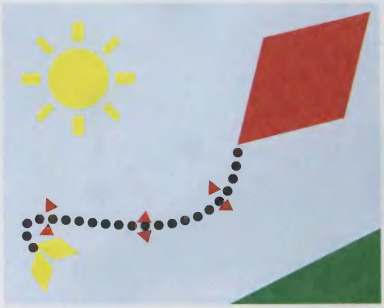
Направление наблюдения: -1 -1 1 **Направление наблюдения: 1 -1 1**

Направление наблюдения: 1 1 1 **Направление наблюдения: -1 1 1**

(a)

(b)

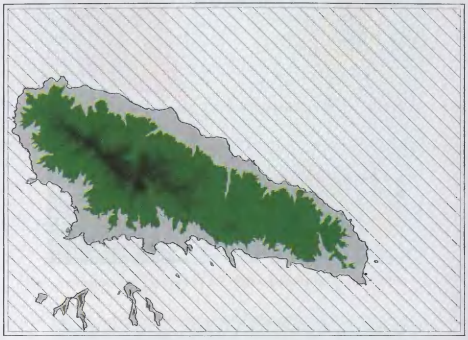
Цветная вклейка IV. Примеры на PSTricks (3).



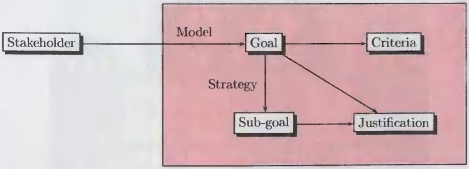
(a)

(b)

Цветная вклейка V. Примеры на PSTricks (4).



(a)



(b)

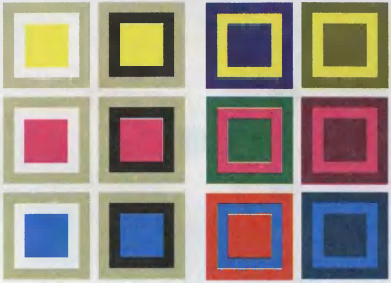
Цветная вклейка VI. Примеры на PSTricks (5).

Цветная вклейка VII. Взаимосвязь моделей RGB и CMYK.

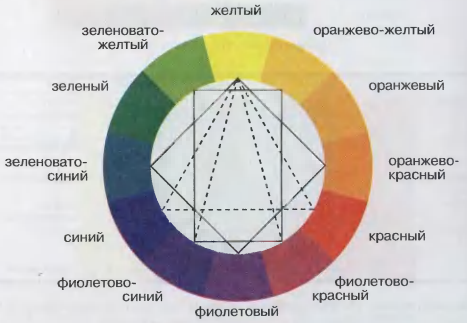
Цветная вклейка VIII. Модель HSB.

	красный	голубой	зеленый	желтый	белый
Китай	счастье	небо облака	процветание жизнь природа	рождение богатство мощь	смерть непорочность
Египет	смерть	добродетель вера правда	плодородие сила	счастье процветание	радость
Франция	аристократия	свобода мир	прогулочность	сумасшедшие	нейтральность
Индия	жизнь творчество сила		процветание благородие	успех	смерть непорочность
Япония	гнев опасность	позор	будущее молодость энергия	трусы благородство	смерть
США	опасность	мужественность	безопасность	трусость	непорочность

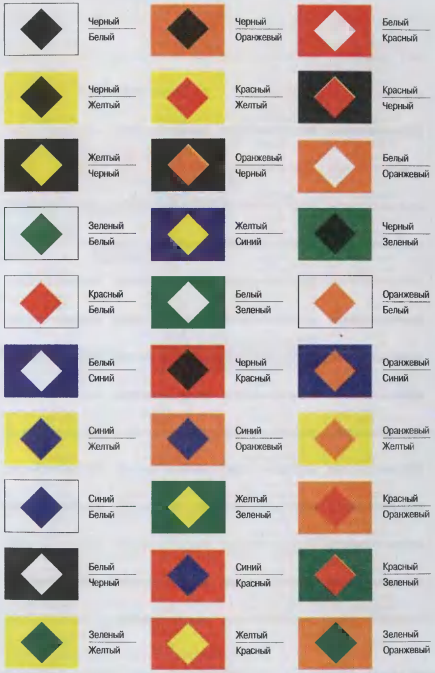
Цветная вклейка IX. Символические значения цветов в разных странах (см. Greenwood, 1993).



Цветная вклейка X. Цветовая гармония и основные цвета.



Цветная вклейка XI. Цветовые гармонии и цветовое кольцо (согласно Itten, 1974).



Цветная вклейка XII. Цветовые контрасты для усиления восприятия и большей удобочитаемости (согласно Duplan and Jauneau, 1986, с. 192).

Цветная вклейка XIII. Список предопределенных цветов «Crayola», представленных в цветовой модели *named* для драйвера dvips.

Текст начинаем зеленым чуть красного с вкраплением синего и завершаем зеленым

1. пурпурный cmyk черный
2. предопределенный синий серый текст

Начинает [черный] затем [оранжевый] и завершает снова черный

Начинаем с зеленого, см. [черный] затем [оранжевый] и снова зеленый

Черный текст на красном фоне

Черный текст, красный фон, синяя рамка

Белый текст, синий фон, красная рамка

Белый на фоне | Красный на фоне

Светлый фон | Темный фон

Наслаждайтесь цветом | Наслаждайтесь цветом

Наслаждайтесь цветом | Наслаждайтесь цветом

- Farallon's Replica
  - работает только в ОС Windows;
  - преобразует PostScript'овские шрифты type 1 в растровые изображения;
  - хорошо интегрирована с операционной средой.
- Common Ground фирмы No Hands Software
  - работает в ОС Mac и Windows;
  - имеет только четыре фиксированных разрешения;
  - хорошо интегрирована с операционной средой.
- Acrobat фирмы Adobe
  - работает в ОС Mac, Windows и Unix (Sun, HP, Irix);
  - с помощью программы Distiller обрабатывает любой PostScript'овский документ;
  - содержит бесплатную программу Reader.

Цветная вклейка XIV. Примеры в LaTeX'овском пакете color.

LONDON					Price
Sydney	OG4G	Thu Oct 10	Mon Oct 21 or 28	11 or 18 days	999GBP
		Thu Oct 17	Mon Oct 21 or 28	4 or 11 days	999GBP
	OG7A	Sun Oct 13	Mon Oct 21 or 28	8 or 15 days	999GBP
		Sun Oct 20	Mon Oct 28	8 days	999GBP

Цветная вклейка XV. Простая разноцветная таблица.

United Kingdom	London	Thames
France	Paris	Seine
Russia	Moscow	Moskva

Цветная вклейка XVI. Таблица с раскрашенными линейками.

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmmm
Row two	mmmmm	mmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmm
Totals	mmmmmm	mmmmmm

(a)

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmmm
Row two	mmmmm	mmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmm
Totals	mmmmmm	mmmmmm

(b)

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmmm
Row two	mmmmm	mmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmm
Totals	mmmmmm	mmmmmm

(c)

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmmm
Row two	mmmmm	mmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmm
Totals	mmmmmm	mmmmmm

(d)

Цветная вклейка XVII. Двуцветные таблицы (часть I).

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(a)

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(b)

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(c)

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(d)

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(e)

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(f)

Цветная вклейка XVIII. Двуцветные таблицы (часть II).

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(a)

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(b)

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(c)

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(d)

Table title

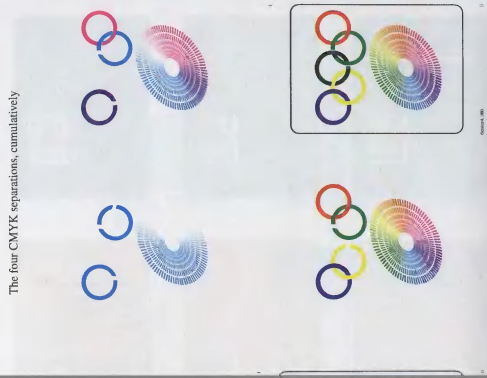
Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(e)

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

(f)



9.1.5 Цвет и удобочитаемость

Удобочитаемость текста или знака тесно связана с тем, как зрительная система человека обрабатывает поступающую информацию. На восприятие цветов оказывают влияние следующие факторы:

- *интенсивность*: чистые цвета спектра обладают наибольшей интенсивностью;
- *контраст*: между разными цветами;
- *чистота*: чистые цвета воспринимаются лучше, чем смешанные варианты, в которые добавлен белый, делая их светлее, или черный цвет, делая их темнее.

На цветной вклейке XII показаны некоторые наиболее эффективные с точки зрения усиления восприятия или максимальной удобочитаемости цветовые контрасты, которые можно использовать, например, на слайдах, дорожных знаках или рекламных буклетах.

9.2 Цвета и LaTeX — пакет color

Поддержка цветов в LaTeX'е основана на введенном выше понятии *системы представления цвета* (цветовых моделей). Наборы поддерживаемых разными драйверами цветовых моделей могут различаться, но обычно в них входят:

rgb — *red, green, blue* (красный, [illegible])

Цветная вклейка XXI. Цветоделение в модели CMYK.

9.1.5 Цвет и удобочитаемость

Удобочитаемость текста или знака тесно связана с тем, как зрительная система человека обрабатывает поступающую информацию. На восприятие цветов оказывают влияние следующие факторы:

- *интенсивность*: чистые цвета спектра обладают наибольшей интенсивностью;
- *контраст*: между разными цветами;
- *чистота*: чистые цвета воспринимаются лучше, чем смешанные варианты, в которые добавлен белый, делая их светлее, или черный цвет, делая их темнее.

На цветной вклейке XII показаны некоторые наиболее эффективные с точки зрения усиления восприятия или максимальной удобочитаемости цветовые контрасты, которые можно использовать, например, на слайдах, дорожных знаках или рекламных буклетах.

9.2 Цвета и LaTeX — пакет color

Поддержка цветов в LaTeX'е основана на введенном выше понятии *системы представления цвета* (цветовых моделей). Наборы поддерживаемых разными драйверами цветовых моделей могут различаться, но обычно в них входят:

`rgb` *red green blue* (*красный зеленый синий*). Цвет кодируется тремя действительными числами, разделенными запятыми, принимающими значения между 0 и 1 (цветовые компоненты аддитивной модели RGB).

`cmyk` *cyan magenta yellow black* (*голубой пурпурный желтый черный*). Цвет кодируется разделенным запятыми набором из четырех действительных чисел, принимающих значения между 0 и 1. Это задает цветовые компоненты субтрактивной модели CMYK, которая используется в большинстве принтеров.

`gray` *grayscale* (*полутоновая модель*). Яркость пикселов задается единственным действительным числом между 0 (черный цвет) и 1 (белый цвет).

`named` Используется имя, выбранное из списка предопределенных цветов.

Цветовая модель `named` поддерживается не всеми драйверами. Если она доступна, то драйвер может заранее сопоставить каждому имени цвет, описанный в одной из других цветовых моделей, или воспользоваться именами цветов, встроенных в принтер или предопределенных в языке описания страниц, таком, как PostScript. Драйвер dvips предоставляет 68 цветов «Crayola», изначально предложенных Джимом Хафнером в пакетах colordvi и foiltex. Эти имена и цвета перечислены на цветной вклейке XIII. Важно иметь в виду, что эти цвета определены не в терминах параметров модели CMYK, а на уровне PostScript'а в загружаемом заголовочном файле, который можно подстраивать под разные принтеры. Это позволяет получить бóльшую гибкость в отличие

от использования фиксированных CMYK- или RGB-значений, которые могут плохо выглядеть на вашем конкретном устройстве.

Пользователи могут определять имена цветов и с помощью команды `\definecolor`, описанной в разд. 9.2.2 на с. 372, которая связывает эти имена с численными значениями одной из цветовых моделей. Могут использоваться и другие цветовые модели (такие, как Pantone, X11, HSB, и т. д.). Для каждой новой модели в файлы настройки драйвера, входящие в данный пакет поддержки цвета, должен быть включен соответствующий набор инструкций. Некоторые драйверы позволяют регулировать даже насыщенность цвета, например команда `ForestGreen .4` задает насыщенность данного оттенка зеленого цвета равной 40%.

9.2.1 Поддерживаемые параметры

Как было сказано в разд. 2.1.1 при обсуждении графических команд, именно `dvi`-драйвер вашего принтера наносит цвета на бумажный или какой-либо другой носитель. Поэтому этот драйвер должен быть указан в качестве параметра команд `\documentclass` или `\usepackage`, или определен в конфигурационном файле `color.cfg`, например, с помощью команды

```
\ExecuteOptions{dvips}
```

Существует возможность отключить действие всех цветовых команд, указав параметр `monochrome`. Это может быть полезно при предварительном просмотре.

Функционирование модели `named` управляется тремя параметрами. По умолчанию в цветовой модели `named` нет предопределенных имен. Все имена определяются параметром `dvipsnames` среди настроек драйвера dvips, относящихся к цвету. То есть указание параметра `dvips` автоматически задействует `dvipsnames`, если только вы явно не отменили подобное действие, указав параметр `nodvipsnames`. Если вы хотите использовать эти имена с другим драйвером, например, `Textures`, то должны явно указать параметр `dvipsnames`. Наконец, параметр `usenames` делает все имена цветовой модели `named` доступными в качестве LaTeX'овских имен цветов (описанных в разд. 9.2.2).

Работу с цветом в полном объеме поддерживают совсем немного драйверов. На момент написания книги наилучшую поддержку обеспечивали драйверы dvips, dvipsone, pctexps и Textures.

Пакет pstcol

Пакет PSTricks, обсуждавшийся в гл. 4, обеспечивает всестороннее взаимодействие между PostScript'ом и LaTeX'ом. К сожалению, поддержка цвета в этом пакете не до конца совместима с механизмом поддержки цвета в пакете color. Чтобы исправить эту (временную) несовместимость, пакет pstcol модифицирует небольшое число внутренних PSTricks'овских функций. Поскольку PSTricks загружается пакетом pstcol, вы можете использовать внутри PSTricks'овских команд любые цвета, определенные командами пакета color, и наоборот.

9.2.2 Использование цветов

Синтаксис команд, изменяющих цвета, сходен с синтаксисом команд, изменяющих шрифты, и имеет две разновидности: одну декларативную, а другую с аргументами для внесения локальных изменений.

Использование предопределенных цветов

Это — основной способ использования простого набора цветов, заданных пакетом color. Пользователь не обязан знать, какая именно цветовая модель используется. Все драйверы определяют цвета `black` (черный) и `white` (белый). Если они поддерживают модель RGB, то определяют цвета `red` (красный), `green` (зеленый) и `blue` (синий), а если поддерживают модель CMYK, то определяют `cyan` (голубой), `magenta` (пурпурный) и `yellow` (желтый).

`\color{`*name*`}` `\textcolor{`*name*`}{`*text*`}`

Первый вариант `\color` представляет собой команду, изменяющую текущее состояние, которая делает цвета «активными» до конца текущей (явной или неявной) группы. Второй вариант `\textcolor` предназначен для небольших фрагментов текста, выделенных цветом — `\textcolor{`*name*`}{`*text*`}` эквивалентно `{\color{`*name*`}`*text*`}`. Следующий пример также приведен на цветной вклейке XIV:

Текст начинаем зеленым чуть красного с вкраплением синего и завершаем зеленым

```
\usepackage {color}
{\color{green}
Текст начинаем зеленым
\textcolor{red}
          { чуть красного}
{\color{blue}с вкраплением синего}
и завершаем зеленым}
```

9-2-1

Определение значений для цветовых моделей

Если предопределенных цветов недостаточно, вы можете задать цвет явным образом с учетом конкретной цветовой модели.

`\color[`*model*`]{`*specification*`}`
`\textcolor[`*model*`]{`*specification*`}{`*text*`}`

Обратите внимание, что значение первого обязательного аргумента (*specification*) команд `\color` и `\textcolor` меняется, если указан необязательный аргумент *model*. Вместо предопределенного имени цвета при этом вводятся соответствующие значения для данной цветовой модели.

Пример из предыдущего раздела (с предопределенными цветами) можно переписать в терминах модели RGB, как показано ниже (см. также цветную вклейку XIV):

```
\usepackage {color}
{\color[rgb]{0,1,0}
Текст начинаем зеленым
\textcolor[rgb]{1,0,0}
                { чуть красного}
{\color[rgb]{0,0,1}с вкраплением
                  синего}
и завершаем зеленым}
```

9-2-2

Текст начинаем зеленым чуть красного с вкраплением синего и завершаем зеленым

Вот еще один пример (также представленный на цветной вклейке XIV) в виде перечня, в котором объединены предопределенные и локально определенные цвета.

```
\usepackage {color}
\begin{enumerate}
\item \textcolor[cmyk]
   {0,1,0,0}{пурпурный cmyk} черный
\item \color[gray]{0.5}
  \textcolor{blue}{предопределенный
   синий} серый текст
\end{enumerate}
```

9-2-3

1. пурпурный cmyk черный
2. предопределенный синий серый текст

Поскольку использование в документе локальных цветовых определений уменьшает его мобильность и весьма затрудняет его обновление, обычно лучше этой возможностью не пользоваться. Более предпочтительный прием состоит в том, чтобы определить собственные цвета в преамбуле документа и в дальнейшем обращаться к ним по имени, как описано в следующем разделе.

Определение цветов

Любой драйвер должен поддерживать следующие встроенные цвета: `black` (черный), `white` (белый), `red` (красный), `green` (зеленый), `blue` (синий), `cyan` (голубой), `magenta` (пурпурный) и `yellow` (желтый). Программные пакеты могут определять другие цвета. Чтобы определять цвета самостоятельно, вы можете воспользоваться командой `\definecolor`:

`\definecolor{`*name*`}{`*model*`}{`*color specification*`}`

Она определяет *name* как имя цвета, которое может использоваться в цветовых командах в дальнейшем, например,

```
\definecolor{Myorange}{cmyk}{0,0.42,1,0}
\definecolor{Myblue}{rgb}{0.8,0.85,1}
\definecolor{Mygrey}{gray}{0.75}
```

После таких определений команды `Myorange`, `Myblue` и `Mygrey` можно применять наравне со встроенными цветами.

Использование поименованных цветов

Чтобы использовать «поименованные» цвета, поддерживаемые конкретным драйвером, вы указываете имя данного цвета и цветовой модели, как это было описано в предыдущем разделе. Таким образом, строка `\color[named]{SpringGreen}` позволяет выбрать цвет `SpringGreen`.

Вместо того чтобы каждый раз указывать параметр `named`, можно избрать другой путь и с помощью команды `\definecolor` определить для данного цвета псевдоним, например,

```
\definecolor{MyGreen}{named}{SpringGreen}
```

чтобы потом вызывать цвет `SpringGreen` через обращение `\color{MyGreen}`.

Если вы хотите сделать доступными все существующие в модели `named` имена, как будто каждое из них было введено с помощью команды `\definecolor`, то можете воспользоваться необязательным параметром `usenames` данного пакета. При этом для каждого цвета из модели `named` выполняется вызов команды `\definecolor`, что позволяет наравне с командой `\color[named]{SpringGreen}` использовать `\color{SpringGreen}`.

Использование цветовой модели `named` дает определенные преимущества по сравнению с применением других цветовых моделей. Во-первых, поскольку `dvi`-файл обращается к некоторому цвету по имени *name*, реальное смещение первичных цветов, необходимое для получения требуемого цвета, может подстраиваться в соответствии с характеристиками конкретного принтера. В частности, драйвер `dvips` использует внешний файл настроек `color.pro`, который содержит определения цветовых имен. Для обеспечения постоянства цветов при печати на различных устройствах в идеале вам следует задействовать свою версию этого файла для каждого используемого устройства вывода.

Во-вторых, в отличие от так называемых «составных» цветов, получаемых путем смешения первичных цветов в процессе печати, вы можете захотеть использовать «индивидуальные» цвета, для которых данное цветовое имя относится не к сочетанию первичных цветов, а к краске одного определенного цвета. Указав это имя, можно отдельно напечатать части документа, содержащие этот цвет.

Связанные с цветом особые случаи в LaTeX'е

Вам следует иметь в виду некоторую специфику, связанную с хранением боксов в LaTeX'е. Следующий пример (также показанный на цветной вклейке XIV) демонстрирует, что цвет определяется в момент *создания* бокса с помощью команды `\sbox`, а не в момент его *использования* (т. е. цветовые характеристики хранятся вместе с боксом и цвет окружающего материала не оказывает на них никакого влияния).

Начинает
[черный] затем [оранжевый] и
завершает снова черный
Начинаем с зеленого, см.
[черный] затем [оранжевый] и снова
зеленый

```
\usepackage {color}
\newsavebox{\X}
\sbox{\X}{[черный] затем
 \color[cmyk]{0,0.6,0.8,0}
             [оранжевый]}

Начинает \usebox{\X}
и завершает снова черный

{\color{green}Начинаем с зеленого, см
\usebox{\X} и снова зеленый}
```

9-2-4

Это аналогично тому, как LaTeX обращается с атрибутами шрифтов, которые также фиксируются в момент создания бокса. Однако используемые внутренние механизмы сильно отличаются, что может привести к неприятным сюрпризам, если структура новых команд будет недостаточно продумана.

Внутри бокса, созданного, например, с помощью команды `\sbox` или команды низкого уровня `\hbox`, каждый символ содержит информацию о том, каким шрифтом он набран. В противоположность этому, цвет определяется сообщениями о «начальном цвете» и «конечном цвете», которые помещаются в выходной поток, поэтому, если такой бокс подвергнется декомпозиции при помощи TeX'овских команд низкого уровня, соответствие между цветом и текстом может быть утрачено. В качестве примера проблем, которые могут при этом возникнуть, рассмотрим ситуацию, в которой набранный красным текст начинается в конце одной страницы и переходит на следующую. Когда эти страницы начинает обрабатывать ничего не знающий о «цвете» TeX'овский формат, такой, как plainTeX (но не LaTeX 2_ε), верхний и нижний колонтитулы, предшествующие и следующие за разрывом страницы, тоже набираются красным цветом. Сходные трудности возникают при обработке форматами такого же рода перечней (например, метки элементов списка могут быть неверно окрашены).

Стандартный LaTeX прошел долгий путь, прежде чем сумел обеспечить работу с цветом, и множество потенциальных проблем уже учтены в основной LaTeX'овской программе (сейчас вы можете не беспокоиться о том, что случайно получите цветные колонтитулы), но вы все-таки можете столкнуться с неожиданными цветовыми эффектами, если используемые пакеты создавались без учета ограничений, накладываемых при работе с цветом. Трудности такого рода подробно обсуждаются в документе «LaTeX 2_ε for class and package writers», который распространяется вместе с LaTeX'ом, и в работе Rokicki (1994). Если только вы не являетесь опытным LaTeX'овским программистом, разумно ограничить свою работу с цветом простыми ситуациями, пользуясь при этом стандартным LaTeX'ом, а не TeX'овскими командами низкого уровня (например, `\savebox` вместо `\setbox`, `\mbox` вместо `\hbox`, и т. д.), и тогда вы можете быть уверены, что получите тот результат, на который рассчитывали.

9.2.3 Цвет страницы

`\pagecolor{name}`

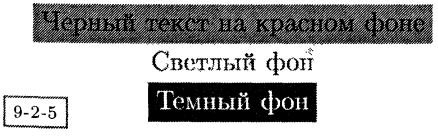
Цвет фона всей страницы можно задать с помощью команды `\pagecolor`, которая обладает такими же, как у команды `\color`, аргументами, но устанавливает цвет фона текущей и всех последующих страниц. Чтобы восстановить белый цвет фона (значение, принятое по умолчанию), вы обязательно должны задать команду `\pagecolor{white}`, поскольку эта команда является глобальным определением.

9.2.4 Цветной фон бокса

Боксы с фоном заданного цвета создаются двумя командами, которые похожи на `\fbox`.

`\colorbox{`*background color*`}{`*text*`}`

Команда `\colorbox` помещает текст *text* внутрь бокса и окрашивает фон. Этот фон распространяется на расстояние `\fboxsep` во всех четырех направлениях:

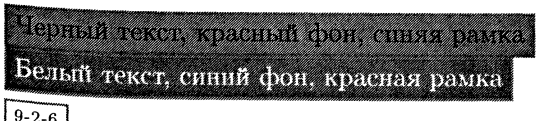


9-2-5

```
\usepackage {color}
\definecolor{Light}{gray}{.80}
\definecolor{Dark}{gray}{.20}
\colorbox{red}{Черный текст
              на красном фоне}
\par\colorbox{Light}{%
  \textcolor{Dark}{Светлый фон}}
\par\colorbox{Dark}{%
  \textcolor{white}{Темный фон}}
```

`\fcolorbox{`*frame color*`}{`*background color*`}{`*text*`}`

Команда `\fcolorbox` вокруг окрашенного бокса дополнительно создает рамку (указанного цвета) (этот пример также приведен на цветной вклейке XIV):

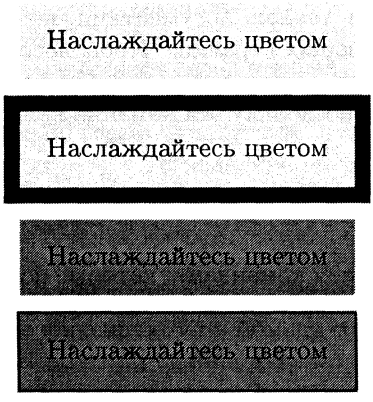


9-2-6

```
\usepackage {color}
    \fcolorbox{blue}{red}
        {Черный текст,
         красный фон, синяя рамка}
    \fcolorbox{red}{blue}{%
        \color{white}Белый текст,
         синий фон, красная рамка}
```

Несколько дальнейших примеров (также показанных на цветной вклейке XIV) демонстрируют, как задающие толщину линий и размер закрашенной

области параметры \fboxrule и \fboxsep команды \fbox позволяют контролировать форму бокса.




```
\usepackage {color}
\setlength{\fboxrule}{6pt}%
\setlength{\fboxsep}{10pt}%
\colorbox{yellow}
     {Наслаждайтесь цветом}
     \par\medskip\par
\fcolorbox{red}{yellow}
     {Наслаждайтесь цветом}
     \par\medskip\par
\setlength{\fboxrule}{1pt}%
\colorbox{green}
     {Наслаждайтесь цветом}
     \par\medskip\par
\fcolorbox{red}{green}
     {Наслаждайтесь цветом}
```

9-2-7

Комбинируя использование PostScript'овских шрифтов и цвета, вы можете создавать перечни с цветными элементами (также приведенные на цветной вклейке XIV). Команда \ding взята из пакета pifont, описанного в разд. 10.1.1:

```
\usepackage{pifont,color}
\newenvironment{coldinglist}[2]
        {\begin{list}{\textcolor{#2}{\ding{#1}}}{}}
        {\end{list}}
\begin{coldinglist}{113}{magenta}
  \item \textcolor{cyan}{Farallon's Replica}
  \begin{coldinglist}{42}{green}
    \item работает только в ОС Windows;
    \item преобразует PostScript'овские шрифты type 1 в растровые
          изображения;
    \item хорошо интегрирована с операционной средой.
  \end{coldinglist}
  \item \textcolor{cyan}{Common Ground фирмы No
   Hands Software}
   ....
```

- Farallon's Replica
  - работает только в ОС Windows;
  - преобразует PostScript'овские шрифты type 1 в растровые изображения;
  - хорошо интегрирована с операционной средой.
- Common Ground фирмы No Hands Software
  - работает в ОС Mac и Windows;

- имеет только четыре фиксированных разрешения;
- хорошо интегрирована с операционной средой.

Acrobat фирмы Adobe

- работает в ОС Mac, Windows и Unix (Sun, HP, Irix);
- с помощью программы Distiller обрабатывает любой PostScript'овский документ;
- содержит бесплатную программу Reader.

Более сложные операции с цветом можно выполнять с помощью пакета colortbl, который позволяет вам создавать цветные таблицы (см. разд. 9.3), или с помощью класса seminar, который предназначен для создания цветных слайдов (см. разд. 9.4).

9.2.5 Вычисление цветов

Используя возможности пакета color, вы легко можете получать оттенки произвольного цвета, устанавливая доли голубой, пурпурной, желтой и черной компонент в диапазоне от 0 до 90%. В следующем примере при создании рисунка используется управляющая конструкция \whiledo, определенная в пакете ifthen.

```
\usepackage {ifthen,color}
\usepackage [cp866]{inputenc}
\usepackage [LCY]{fontenc}
\newcounter{Colr}
\setlength{\fboxsep}{2mm}
\begin{flushleft}
  \newcommand{\CBox}[1]{\colorbox[cmyk]{.#1,0.,0.,0.}{.#1}}
  \makebox[30mm][l]{голубой (C):}%
    \whiledo{\value{Colr}<10}{\CBox{\theColr}\stepcounter{Colr}}\\
  \renewcommand{\CBox}[1]{\colorbox[cmyk]{0.,.#1,0.,0.}{.#1}}
  \setcounter{Colr}{0}\makebox[30mm][l]{пурпурный (M):}%
    \whiledo{\value{Colr}<10}{\CBox{\theColr}\stepcounter{Colr}}\\
  \renewcommand{\CBox}[1]{\colorbox[cmyk]{0.,0.,.#1,0.}{.#1}}
  \setcounter{Colr}{0}\makebox[30mm][l]{желтый (Y):}%
    \whiledo{\value{Colr}<10}{\CBox{\theColr}\stepcounter{Colr}}\\
  \renewcommand{\CBox}[1]{\colorbox[cmyk]{0.,0.,0.,.#1}{.#1}}
  \setcounter{Colr}{0}\makebox[30mm][l]{черный (K):}%
    \whiledo{\value{Colr}<10}{\CBox{\theColr}\stepcounter{Colr}}
\end{flushleft}
```

голубой (C): .0 .1 .2 .3 .4 .5 .6 .7 .8 .9

пурпурный (M): .0 .1 .2 .3 .4 .5 .6 .7 .8 .9

желтый (Y): .0 .1 .2 .3 .4 .5 .6 .7 .8 .9

9-2-8 черный (K): .0 .1 .2 .3 .4 .5 .6

Данный прием хорошо подходит для работы с «полутоновой» цветовой моделью, которая может успешно применяться на обычных лазерных принтерах. На приведенном ниже рисунке автоматически полученные данные случайно распределены в таблице, причем яркость серого фона каждой ячейки соответствует значению в данной ячейке. Численные значения яркости представлены в исходных данных в процентах, и для соответствия полутоновой цветовой модели преобразуются в диапазон 0–1 путем добавления спереди десятичной точки «.» (по счастью, в примере нет значений 100%). Само число набирается белым или черным цветом в зависимости от цвета фона.

```
\usepackage{ifthen,color}

\newcommand{\sq}[2]{%
 \definecolor{thiscol}{gray}{.#2}%
 \ifthenelse{#2<50}%
    {\color{white}}%
    {\color{black}}%
 \colorbox{thiscol}{\makebox[2em]{#1}}
}
\setlength{\tabcolsep}{0pt}
\begin{tabular}{*{10}{c}}
\sq{102}{85}&\sq{10}{98}&\sq{110}{84}&
\sq{112}{83}&\sq{114}{83}&\sq{118}{82}&
\sq{119}{82}&\sq{11}{98}&\sq{122}{82}&
\sq{128}{81}& \\ \sq{128}{81} &\sq{12}{98}&

...

\end{tabular}
```

102	10	110	112	114	118	119	11	122	128
12	130	135	138	13	141	142	144	14	159
15	160	162	167	16	175	176	17	181	187
193	197	199	19	1	20	215	22	23	247
24	275	27	28	290	296	2	312	32	33
34	35	39	3	40	41	433	44	45	474
47	48	506	50	53	55	57	60	618	62
634	636	64	67	68	693	6	73	74	78
79	7	80	81	85	8	92	93	98	9

Поскольку LaTeX'овский пакет color в общем не зависит от какой-либо конкретной системы представления цвета, вы можете использовать любую систему, поддерживаемую вашим драйвером. Например, в случае драйвера dvips, представляющего результаты на PostScript'е, вы можете задавать цвета, основываясь на системе HSB (Hue, Saturation, Brightness — оттенок, насыщенность, яркость), которая весьма удобна для художников, так как цвет (оттенок) зада-

ется одним числом, а насыщенность и яркость легко изменяются независимо от выбранного цвета. Эта модель также удобна для цветовых вычислений. В следующем примере (также представленном на цветной вклейке VIII) мы по очереди меняем каждый из трех параметров модели HSB, которые за десять шагов пробегают значения от 0 до 1. В этом примере опять встречается управляющая конструкция `\whiledo`, определенная в пакете ifthen.

```
\usepackage {ifthen,color}
\usepackage [cp866]{inputenc}
\usepackage [LCY]{fontenc}
\begin{flushleft}
\newcounter{Col}
\setlength{\fboxsep}{2mm}
\newcommand{\ColBox}[1]{\colorbox[hsb]{.#1,1.,1.}{.#1}}%
\setcounter{Col}{0}\makebox[27mm][l]{Оттенок:}%
\whiledo{\value{Col}<10}{\ColBox{\theCol}\stepcounter{Col}}\\
\renewcommand{\ColBox}[1]{\colorbox[hsb]{.1,.#1,1.}{.#1}}%
\setcounter{Col}{0}\makebox[27mm][l]{Насыщенность:}%
\whiledo{\value{Col}<10}{\ColBox{\theCol}\stepcounter{Col}}\\
\renewcommand{\ColBox}[1]{\colorbox[hsb]{1.,1.,.#1}{.#1}}%
\setcounter{Col}{0}\makebox[27mm][l]{Яркость:}%
\whiledo{\value{Col}<10}{\ColBox{\theCol}\stepcounter{Col}}\\
\end{flushleft}
```

Оттенок: .0 .1 .2 .3 .4 .5 .6 .7 .8 .9

Насыщенность: .0 .1 .2 .3 .4 .5 .6 .7 .8 .9

Яркость: .0 .1 .2 .3 .4 .5 .6 .7 .8 .9

9-2-9

9.3 Раскраска таблиц

Добавление цвета в таблицы оказывается не такой простой задачей, как хотелось бы. Основная проблема состоит в том, что поддержка работы с цветом внутри ячеек или строк не может быть обеспечена на уровне TEX'а, но должна быть добавлена в тексты многочисленных пакетов. Пакеты для работы с таблицами сложны и разнообразны, а стандартные пакеты не обеспечивают непосредственную поддержку работы с цветом. LATEX'овский пакет color может осуществлять взаимосвязь со средствами поддержки цвета драйверов, но значительная часть работы все же должна быть выполнена внутренними командами строящих таблицы окружений (`tabular` или `array`).

Пакет colortab Тимоти ван Зандта (van Zandt, 1993c) служит связующим звеном с plain TEX'овским примитивом для построения таблиц `\halign`. Этот пакет позволяет окрашивать как ячейки, так и сетку таблицы, но он будет работать только в том случае, если пакет PSTricks поддерживает ваш PostScript'овский драйвер, потому что для выполнения своей работы colortab ис-

пользует PSTricks. Пакет colortab также предоставляет некоторые средства для раскраски ячеек LaTeX'овских окружений `array` и `longtable`, но используемый синтаксис ужасен и работает не со всеми пакетами построения таблиц. По этой причине мы описываем ориентированный на LaTeX пакет colortbl.

9.3.1 Пакет colortbl

Пакет colortbl был написан Дэвидом Карлайлом для реализации базовых средств раскраски ячеек и сетки таблиц. Он может быть использован вместе с другими дополнительными LaTeX'овскими пакетами для работы с таблицами, такими, как longtable, dcolumn и hhline.[1] Для обеспечения базовых возможностей он загружает стандартные пакеты color и array.

Основной командой этого пакета является:

`\columncolor[`*color model*`]{`*color*`}[`*left overhang*`][`*right overhang*`]`

Она может использоваться *только* в качестве аргумента спецификатора столбцов `>`, внутри преамбулы таблицы или в описании столбцов в команде `\multicolumn`. Она позволяет задать цвет фона данного столбца или ячейки. Описания цветовой модели *color model* и цвета *color* ничем не отличаются от подобных описаний пакета color. Аргументы «overhang» определяют, насколько далеко вправо или влево от содержимого столбца простирается цветной фон. По умолчанию они равны величине `\tabcolsep` (в случае окружения `tabular`) или величине `\arraycolsep` (в случае окружения `array`), т. е. цвет полностью заполняет столбец.

В качестве нашего первого примера приведем многоцветную таблицу, содержащую заманчивые предложения провести несколько дней в Австралии, также представленную на цветной вклейке XV:

```
\usepackage {colortbl}
\setlength{\extrarowheight}{2mm}
\setlength{\tabcolsep}{2mm}
\begin{tabular}{|l|%
  >{\columncolor{yellow}}c|c|>{\columncolor{yellow}}c|c|%
  >{\columncolor{red}\bfseries}c<{\textsc{GBP}}|}
\hline
 \multicolumn{3}{>{\columncolor{red}}l}{\color{white}\textsf{LONDON}}
&\multicolumn{3}{>{\columncolor{red}}r}{\color{white}\textsf{Price}}
\\[1pt]
\hline
Sydney  & OG4G &Thu Oct 10 &Mon Oct 21 or 28 &11 or 18 days &999\\
        &      &Thu Oct 17 &Mon Oct 21 or 28 & 4 or 11 days &999\\
```

[1] Вы также можете использовать colortbl вместе с `tabular*`, однако перед этим ознакомьтесь с документацией на предмет граничных условий, когда полученные результаты могут отличаться от ожидаемых.

```
        & OG7A &Sun Oct 13 &Mon Oct 21 or 28 & 8 or 15 days &999\\
        &      &Sun Oct 20 &Mon Oct 28       & 8 days       &999\\
\hline
\end{tabular}
```

LONDON					Price
Sydney	OG4G	Thu Oct 10	Mon Oct 21 or 28	11 or 18 days	
		Thu Oct 17	Mon Oct 21 or 28	4 or 11 days	
	OG7A	Sun Oct 13	Mon Oct 21 or 28	8 or 15 days	
		Sun Oct 20	Mon Oct 28	8 days	

9-3-1

Сравните ее со следующей упрощенной версией таблицы, в которой команда `\newcolumntype` определяет столбец с серым фоном, причем цвет покрывает только текст в ячейках, не распространяясь на свободное пространство по обе стороны:

```
\usepackage {colortbl}
\setlength{\extrarowheight}{2mm}
\setlength{\tabcolsep}{2mm}
\newcolumntype{G}{>{\columncolor[gray]{0.9}[0pt][0pt]}c}
\begin{tabular}{|l|c|G|c|G|c|G|c<{\textsc{GBP}}|}
\hline
  \multicolumn{3}{>{\columncolor[gray]{0.7}}l}{\textsf{LONDON}}
 &\multicolumn{3}{>{\columncolor[gray]{0.7}}r}{\textsf{Price}}
\\[1pt]
\hline
Sydney  & OG4G &Thu Oct 10 &Mon Oct 21 or 28 &11 or 18 days &999\\
        &      &Thu Oct 17 &Mon Oct 21 or 28 & 4 or 11 days &999\\
        & OG7A &Sun Oct 13 &Mon Oct 21 or 28 & 8 or 15 days &999\\
        &      &Sun Oct 20 &Mon Oct 28       & 8 days       &999\\
\hline
\end{tabular}
```

LONDON					Price
Sydney	OG4G	Thu Oct 10	Mon Oct 21 or 28	11 or 18 days	999
		Thu Oct 17	Mon Oct 21 or 28	4 or 11 days	999
	OG7A	Sun Oct 13	Mon Oct 21 or 28	8 or 15 days	999
		Sun Oct 20	Mon Oct 28	8 days	999

9-3-2

Подобная система хороша для вертикально окрашенных таблиц, но как нам справиться с таблицами, в которых разноцветными являются *строки*? Применение команды `\multicolumn` в данном случае было бы неудачным, поэтому в пакете colortbl предусмотрена еще одна команда:

`\rowcolor[`*color model*`]{`*color*`}[`*left overhang*`][`*right overhang*`]`

Она может быть помещена только в начале строки. Она отменяет любые текущие спецификации `\columncolor`, но ее действие ограничено текущей строкой:

```
\usepackage {colortbl}
\setlength{\extrarowheight}{2mm}
\begin{tabular}{|l|c|c|c|c|c|c|}
\hline
Sydney    & OG4G &Thu Oct 10 &Mon Oct 21 or 28 &11 or 18 days &999\\
\rowcolor[gray]{0.8}
          &      &Thu Oct 17 &Mon Oct 21 or 28 & 4 or 11 days &999\\
          &OG7A  &Sun Oct 13 &Mon Oct 21 or 28 & 8 or 15 days &999\\
\rowcolor[gray]{0.8}
          &      &Sun Oct 20 &Mon Oct 28       & 8 days       &999\\
\hline
\end{tabular}
```

Sydney	OG4G	Thu Oct 10	Mon Oct 21 or 28	11 or 18 days	999
		Thu Oct 17	Mon Oct 21 or 28	4 or 11 days	999
	OG7A	Sun Oct 13	Mon Oct 21 or 28	8 or 15 days	999
		Sun Oct 20	Mon Oct 28	8 days	999

9-3-3

Для определения цвета *линий сетки* в окружении `tabular`, в которое входят команды `\vline`, `\cline` и `\hline`, в пакете colortbl существует декларация

`\arrayrulecolor[`*color model*`]{`*color*`}`

которая имеет аргументы, похожие на аргументы команды `\color`, и определяет все последующие горизонтальные и вертикальные линии сетки таблицы. Она может находиться вне таблицы, в начале строки или внутри спецификации > в преамбуле таблицы. Ее влияние *не ограничивается* текущей областью видимости, но не отменяет спецификации, заданные в преамбуле таблицы. Так, в следующей таблице (также показанной на цветной вклейке XVI) цвет вертикальных линий сетки установлен зеленым в самом начале и не изменяется последующими командами `\arrayrulecolor`:

```
\usepackage {colortbl}
\setlength{\arrayrulewidth}{2pt}
\arrayrulecolor{green}
\begin{tabular}{|l|c|r|}
\arrayrulecolor{black}\hline
 United Kingdom & London & Thames\\
\arrayrulecolor{blue}\hline
 France         & Paris  & Seine \\
\arrayrulecolor{black}\cline{1-1}
\arrayrulecolor{red}\cline{2-3}
 Russia         & Moscow & Moskva\\
                                \hline
\end{tabular}
```

United Kingdom	London	Thames
France	Paris	Seine
Russia	Moscow	Moskva

9-3-4

Есть одна проблема: если окрашенные линии используются одновременно с окрашенными ячейками, частичные линии `\cline` приобретают цвет ячеек и, таким образом, становятся невидимыми. В случае необходимости для достижения желаемого эффекта вы можете воспользоваться более сложным пакетом hhline в сочетании с colortbl.

Еще одна область таблицы может быть окрашена — промежуток между линиями, созданными с помощью символов `||` в преамбуле таблицы или с помощью команд `\hline\hline` в теле таблицы. Это можно сделать командой

`\doublerulesepcolor[`*color model*`]{`*color*`}`

которая действует подобно `\arrayrulecolor`. Однако обратите внимание, что, окрашивая пространство, ограниченное двумя командами `\hline` в рамках пакета longtable, LaTeX не допускает в этой точке разрыва страниц. Это один из немногих случаев, когда добавление цвета в ваш документ может привести к изменению разрывов страниц.

9.3.2 Примеры

Теперь мы покажем, как можно изменять цвет элементов таблицы с помощью различных команд, описанных выше. Поддержка работы с цветом в LaTeX'е пока не совершенна, поэтому время от времени для достижения желаемого эффекта приходится использовать достаточно сложные конструкции. Все эти таблицы воспроизведены на цветных вклейках XVII–XIX.

Мы приглашаем наших читателей изучить охватывающую многие аспекты цвета замечательную книгу *Color for the Electronic Age* (White, 1990), чтобы узнать побольше о том, как использование цвета может увеличивать ценность табличного материала. В главе, посвященной схемам и диаграммам, Ян Уайт показывает, как можно избежать однообразия при демонстрации серии таблиц,

применяя один и тот же цвет разными способами. В следующих примерах мы реализуем некоторые из его рекомендаций.

Начнем с рассмотрения таблицы общего вида, которая послужит основой для нашего обсуждения (цветная вклейка XVIIa):

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmmm
Row two	mmmmm	mmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmm
Totals	mmmmmm	mmmmmm

```
\usepackage {colortbl}
\begin{tabular}{lrr}
  \large\textbf{Table title}
                          \\[2mm]
  \textbf{Description}
   & \textbf{Column 1}
     & \textbf{Column 2}\\[1mm]
  Row one  & mmmmmm & mmmmm  \\
  Row two  &  mmmmm & mmmm   \\
  Row three& mmmmmm & mmmmmm \\
  Row four & mmmmmm & mmmmm  \\
\cline{2-3}
  Totals   & mmmmmm & mmmmmm
\end{tabular}
```

9-3-5

Для начала поместим таблицу целиком внутрь бокса, скажем, оранжевого цвета, определив этот цвет с помощью команды `\definecolor`, а затем применив простую команду `\colorbox` (см. цветную вклейку XVIIb):

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmmm
Row two	mmmmm	mmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmm
Totals	mmmmmm	mmmmmm

```
\usepackage {colortbl}
\definecolor{orange}{cmyk}
                {0,0.52,1,0}
\colorbox{orange}{%
\begin{tabular}{lrr}
  \large\textbf{Table title}
                          \\[2mm]
  \textbf{Description}
   & \textbf{Column 1}
     & \textbf{Column 2}\\[1mm]
  Row one  & mmmmmm & mmmmm  \\
  Row two  &  mmmmm & mmmm   \\
  Row three& mmmmmm & mmmmmm \\
  Row four & mmmmmm & mmmmm  \\
\cline{2-3}
  Totals   & mmmmmm & mmmmmm
\end{tabular}%
}
```

9-3-6

Это не только не добавило новой информации в таблицу, но текст стало труднее читать, потому что он выглядит слишком темным. Поэтому мы можем попробовать напечатать текст белым цветом (цветная вклейка XVIIc):

```
\usepackage {colortbl}
\definecolor{orange}{cmyk}
                {0,0.52,1,0}
\colorbox{orange}{%
\color{white}%
\begin{tabular}{lrr}
  \large\textbf{Table title}
                          \\[2mm]
  \textbf{Description}
   & \textbf{Column 1}
     & \textbf{Column 2}\\[1mm]
  Row one  & mmmmmm & mmmmm  \\
  Row two  &  mmmmm & mmmm   \\
  Row three& mmmmmm & mmmmmm \\
  Row four & mmmmmm & mmmmm  \\
\cline{2-3}
  Totals   & mmmmmm & mmmmmm
\end{tabular}%
}
```

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmmm
Row two	mmmmm	mmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmm
Totals	mmmmmm	mmmmmm

9-3-7

Разумеется, этот способ не оказывает влияния на цвет горизонтальной линии, построенной с помощью команды `\cline`.

В зависимости от обстоятельств вы можете считать или не считать это улучшением. Если вы действительно хотите набрать светлый текст на темном фоне, то обычно лучше набирать текст шрифтом более тяжелого рисунка или полужирным вариантом того же шрифта (цветная вклейка XVIId):

```
\usepackage {colortbl}
\definecolor{orange}{cmyk}
                {0,0.52,1,0}
\colorbox{orange}{%
\color{white}\bfseries
\begin{tabular}{lrr}
  \large\textbf{Table title}
                          \\[2mm]
  \textbf{Description}
   & \textbf{Column 1}
     & \textbf{Column 2}\\[1mm]
  Row one  & mmmmmm & mmmmm  \\
  Row two  &  mmmmm & mmmm   \\
  Row three& mmmmmm & mmmmmm \\
  Row four & mmmmmm & mmmmm  \\
\cline{2-3}
  Totals   & mmmmmm & mmmmmm
\end{tabular}%
}
```

Table title

Description	Column 1	Column 2
Row one	mmmmmm	mmmmm
Row two	mmmmm	mmmm
Row three	mmmmmm	mmmmmm
Row four	mmmmmm	mmmmm
Totals	mmmmmm	mmmmmm

9-3-8

Однако до сих пор никакие элементы таблицы не были выделены, так что цвет служил лишь украшением, но не рабочим инструментом. Давайте теперь

посмотрим, как, раскрашивая отдельные элементы, можно привлечь внимание читателя (цветная вклейка XVIIIa):

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

```
\usepackage {colortbl}
\begin{tabular}{lrr}
  \color{green}
  \large\textbf{Table title}
                          \\[3mm]
  \textbf{Description}
   & \textbf{Column 1}
   & \textbf{Column 2}\\[1mm]
  Row one  & mmmmm & mmmm  \\
  Row two  &  mmmm & mmm   \\
  Row three& mmmmm & mmmmm \\
  Row four & mmmmm & mmmm  \\
\cline{2-3}
  \color{green} Totals
     & \color{green} mmmmm
         & \color{green}mmmmm
\end{tabular}
```

9-3-9

Верхняя и нижняя строки набраны зеленым цветом для привлечения внимания. Мы можем также выполнить обратную процедуру, изменив цвет тех фрагментов текста, значение которых мы *не хотим* подчеркивать, тем самым выделяя текст, набранный черным (цветная вклейка XVIIIb):

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

```
\usepackage {colortbl}
{\color{green}%
\begin{tabular}{lrr}
  \color{black}
  \large\textbf{Table title}
                          \\[3mm]
  \textbf{Description}
   & \textbf{Column 1}
   & \textbf{Column 2}\\[1mm]
  Row one  & mmmmm & mmmm  \\
  Row two  &  mmmm & mmm   \\
  Row three& mmmmm & mmmmm \\
  Row four & mmmmm & mmmm  \\
\arrayrulecolor{black}
\cline{2-3}
  \color{black} Totals
     & \color{black}mmmmm
         & \color{black}mmmmm
\end{tabular}%
}
```

9-3-10

Обратите внимание на команду `\arrayrulecolor`, которая окрашивает в черный цвет горизонтальную линию, отделяющую основную часть таблицы от последней строки.

Для привлечения внимания к отдельным строкам таблицы под ними можно поместить цветную полосу (вклейка XVIIIc):

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

9-3-11

```
\usepackage {colortbl}
\newcommand\panel[1]
                    {\multicolumn{1}%
      {>{\columncolor{cyan}}#1}}
\begin{tabular}{lrr}
  \large\textbf{Table title}\\[2mm]
  \textbf{Description}
     & \textbf{Column 1}
          & \textbf{Column 2}\\[1mm]
  Row one & mmmmm & mmmm     \\
  Row two &  mmmm & mmm      \\
  \panel{l}{Row three}
     & \panel{r}{mmmmm}
          & \panel{r}{mmmmm} \\
  Row four& mmmmm & mmmm     \\
                          \cline{2-3}
  Totals  & mmmmm & mmmmm
\end{tabular}
```

Но мы можем сделать еще лучше: окрасить фон всей таблицы, а фон строки, которую требуется выделить, оставить белым (цветная вклейка XVIIId):

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

9-3-12

```
\usepackage {colortbl}
\newcommand\panel[1]
                    {\multicolumn{1}%
        {>{\columncolor{white}}#1}}
\colorbox{cyan}{%
\arrayrulecolor{black}
\begin{tabular}{lrr}
  \large\textbf{Table title}\\[2mm]
  \textbf{Description}
     & \textbf{Column 1}
          & \textbf{Column 2}\\[1mm]
  Row one & mmmmm & mmmm     \\
  Row two &  mmmm & mmm      \\
  \panel{l}{Row three}
     & \panel{r}{mmmmm}
          & \panel{r}{mmmmm} \\
  Row four& mmmmm & mmmm \\
                          \cline{2-3}
  Totals  & mmmmm & mmmmm
\end{tabular}}
```

Этот вариант полностью аналогичен предыдущему примеру за исключением того, что в команде `\columncolor` теперь используется белый цвет, тогда как расположенная в начале команда `\colorbox` устанавливает для всей таблицы пурпурный цвет.

Теперь мы рассмотрим способы выделения не строк, а столбцов. Для задания цвета столбцов используется команда `\columncolor` (цветная вклейка XVIIIe):

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

```
\usepackage {colortbl}
\definecolor{Bluec}{cmyk}{.60,0,0,0}
\begin{tabular}
        {l>{\columncolor{Bluec}}rr}
\large\textbf{Table title}\\[2mm]
\textbf{Description}
    & \textbf{Column 1}
      & \textbf{Column 2} \\[1mm]
Row one  & mmmmm & mmmm   \\
Row two  &  mmmm &  mmm   \\
Row three& mmmmm & mmmmm  \\
Row four & mmmmm & mmmm   \\
Totals   & mmmmm & mmmmm
\end{tabular}
```

9-3-13

Цветовые решения такого рода часто применяются для выделения соседних областей таблицы. Голубой цвет (`Bluec`) установлен в самом начале с помощью стандартной команды `\definecolor`, хотя мы также могли бы определить его с помощью `\arrayrulecolor` следующим образом

```
\arrayrulecolor[cmyk]{.60,0,0,0}
```

Другим часто используемым инструментом при работе с цветом является цветовой градиент (цветная вклейка XVIIIf):

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

```
\usepackage {colortbl}
\definecolor{Bluea}{cmyk}{.20,0,0,0}
\definecolor{Blueb}{cmyk}{.40,0,0,0}
\definecolor{Bluec}{cmyk}{.60,0,0,0}
\definecolor{Blued}{cmyk}{.80,0,0,0}
\definecolor{Bluee}{cmyk}{1.0,0,0,0}
\begin{tabular}{>{\columncolor{Bluee}}l%
    >{\columncolor{Bluee}}r
    >{\columncolor{Bluee}}r}
\multicolumn{3}{>{\columncolor{Bluee}}l}
    {\large\textbf{Table title}}\\[2mm]
\textbf{Description} & \textbf{Column 1}
            & \textbf{Column 2} \\[1mm]
\rowcolor{Bluea}Row one  &mmmmm &mmmm \\
\rowcolor{Blueb}Row two  & mmmm & mmm \\
\rowcolor{Bluec}Row three&mmmmm &mmmmm\\
\rowcolor{Blued}Row four &mmmmm &mmmm \\
\rowcolor{Bluee}Totals   &mmmmm &mmmmm
\end{tabular}
```

9-3-14

Здесь мы использовали разные оттенки голубого (от `Bluea` до `Bluee`), определенные в самом начале программы в последовательно идущих строках. Хотя на первый взгляд метод выглядит немного пугающе, результат оказывается весьма удачным и подобный прием без сомнения является одним из наиболее часто применяемых для построения нарядного и легко читаемого табличного материала.

Для выделения заголовков и названий мы можем изменить цвет тех полей, которые мы считаем наименее важными, а чтобы еще лучше управлять направлением взгляда, мы можем добавить некоторое количество вертикальных и горизонтальных линий (цветная вклейка XIXa):

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

9-3-15

```
\usepackage {colortbl}
\newcommand{\panel}[1]{\multicolumn{1}
 {|>{\columncolor[cmyk]{.40,0,0,0}}
                                r}{#1}}
\begin{tabular}{l|r|r}
\multicolumn{1}{l|}
   {\large\textbf{Table title}}\\[2mm]
\textbf{Description} & \textbf{Column 1}
    & \textbf{Column 2} \\[1mm]\hline
Row one  & \panel{mmmmm} & \panel{mmmm}
\\\hline
Row two  &  \panel{mmmm} &  \panel{mmm}
\\\hline
Row three& \panel{mmmmm} & \panel{mmmmm}
\\\hline
Row four & \panel{mmmmm} & \panel{mmmm}
\\\hline
Totals   & mmmmm & mmmmm
\end{tabular}
```

Как уже отмечалось, мы можем выделить цветом ту область, которая содержит важный текст, а область с менее ценной информацией оставить белой. Поскольку мы хотим, чтобы окрашенный фон был немного больше размеров самой таблицы, параметр `\fboxsep` был установлен равным 3 мм, а в качестве цвета фона был выбран светлоголубой оттенок. В остальном данный пример похож на предыдущий. Интересно сравнить эти два примера, чтобы понять, какой из них более эффективен в донесении информации (цветная вклейка XIXb):

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

9-3-16

```
\usepackage {colortbl}
\newcommand{\panel}[1]{%
 \multicolumn{1}{|
       >{\columncolor{white}}r}{#1}
\setlength\fboxsep{3mm}
\colorbox[cmyk]{.40,0,0,0}{%
\begin{tabular}{l|r|r}
\multicolumn{1}{l|}
  {\large\textbf{Table title}}\\[2mm
\textbf{Description}
   & \textbf{Column 1}
   & \textbf{Column 2} \\[1mm]\hline
Row one  &\panel{mmmmm}&\panel{mmmm}
\\\hline
Row two  & \panel{mmmm}& \panel{mmm}
\\\hline
Row three&\panel{mmmmm}&\panel{mmmmm
\\\hline
Row four &\panel{mmmmm}&\panel{mmmm}
\\\hline
Totals   & mmmmm & mmmmm
\end{tabular}}
```

Не всегда нужно окрашивать сами ячейки столбца. Для выделения соответствующей информации может оказаться достаточно изменения цвета линий сетки внутри таблицы. Это можно сделать средствами пакета colortbl. Сначала мы заменяем принятые по умолчанию черные линии окрашенными другим цветом. Это выполняется с помощью команды `\arrayrulecolor` и оказывает влияние на вертикальные линии, заданные в преамбуле таблицы. После выхода из преамбулы мы переключаемся на другой оттенок голубого цвета, которым окрашиваются все горизонтальные линии, но правая часть самой нижней линии вновь окрашена черным с помощью новой команды `\arrayrulecolor`, указанной перед `\cline` (цветная вклейка XIXc):

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

```
\usepackage {colortbl}
\arrayrulecolor[cmyk]{.80,0,0,0}
\begin{tabular}{l|r|r}
\multicolumn{3}{l}
{\large\textbf{Table title}}\\[2mm]
\textbf{Description}
  & \textbf{Column 1}
  & \textbf{Column 2} \\[1mm]\hline
\arrayrulecolor[cmyk]{1.0,0,0,0}
Row one  & mmmmm & mmmm \\\hline
Row two  &  mmmm &  mmm \\\hline
Row three& mmmmm & mmmmm \\\hline
Row four & mmmmm & mmmm \\\cline{1-1}
\arrayrulecolor{black}\cline{2-3}
Totals   & mmmmm & mmmmm
\end{tabular}
```

9-3-17

Избрав другой путь, мы можем вернуться к голубому фону некоторых строк и нарисовать всего несколько линий, чтобы помочь читателю сориентироваться в информации (цветная вклейка XIXd):

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

```
\usepackage {colortbl}
\arrayrulecolor{black}
\begin{tabular}{l|r|r}
\multicolumn{3}
 {>{\columncolor[cmyk]{1.0,0,0,0}}l}
 {\large\textbf{Table title}}\\[2mm]
\rowcolor[cmyk]{.20,0,0,0}
\textbf{Description}
  & \textbf{Column 1}
    & \textbf{Column 2} \\[1mm]\hline
Row one  & mmmmm & mmmm \\
Row two  &  mmmm &  mmm \\
Row three& mmmmm & mmmmm \\
Row four & mmmmm & mmmm \\\cline{2-3}
Totals   & mmmmm & mmmmm
\end{tabular}
```

9-3-18

Для построения более сложных структур, разбивающих таблицу на несколько областей, мы можем воспользоваться комбинацией разноцветных

строк. Как было отмечено на с. 383, стандартный метод получения частичных горизонтальных линий работает в пакете colortbl некорректно, поэтому в следующем примере используется пакет hhline (цветная вклейка XIXe):

```
\usepackage {colortbl,hhline}
\definecolor{Blueb}{cmyk}{.40,0,0,0} \definecolor{Blued}{cmyk}{.80,0,0,0}
\arrayrulecolor{white}
\begin{tabular}{>{\columncolor{Blued}}l>{\columncolor{Blued}}r|%
                >{\columncolor{Blued}}r}
\multicolumn{3}{>{\columncolor{Blueb}}l}{\large\textbf{Table title}}\\[2mm]
\rowcolor{white}
  \textbf{Description} & \textbf{Column 1} & \textbf{Column 2} \\[1mm]
\arrayrulecolor{black}
\rowcolor{Blueb} Row one   & mmmmm & mmmm    \\\hhline{~--}
                 Row two   &  mmmm & mmm     \\\hhline{~--}
                 Row three & mmmmm & mmmmm   \\\hhline{~--}
                 Row four  & mmmmm & mmmm    \\\hhline{~--}
\rowcolor{white} Totals    & mmmmm & mmmmm
\end{tabular}
```

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

9-3-19

Сочетание различных стилей набора заголовков, цветных линий и разных цветов фона дает вам богатую палитру возможностей для представления сложных данных. В качестве последнего эффективного средства мы рассмотрим изменение *толщины* линий. В представленной ниже таблице толстая горизонтальная линия выделяет название таблицы и заголовки столбцов. Достаточно сложная программа демонстрирует большинство команд, встречавшихся в данном разделе. Обратите внимание на дополнительные столбцы, предназначенные для расширения цветового поля на \quad вправо и влево от основной части таблицы (цветная вклейка XIXf):

```
\usepackage {colortbl}
\definecolor{Blueb}{cmyk}{.40,0,0,0}\definecolor{Blued}{cmyk}{.80,0,0,0}
\definecolor{Bluee}{cmyk}{1.0,0,0,0}
\arrayrulecolor{black}\setlength\arrayrulewidth{1mm}
\begin{tabular}{llrrl}
\rowcolor{Blueb} \qquad&\multicolumn{3}{>{\columncolor{Blueb}}l}
                                {\large\textbf{Table title}}&\qquad\\[2mm]
\rowcolor{Blued}& \textbf{Description}
                        & \textbf{Column 1}  & \textbf{Column 2}& \\[2mm]
\hline
      \rowcolor{Blued} & Row one   & mmmmm & mmmm  &  \\
```

```
      \rowcolor{Blued} & Row two   &  mmmm & mmm   &  \\
      \rowcolor{Blued} & Row three & mmmmm & mmmmm &  \\
      \rowcolor{Blued} & Row four  & mmmmm & mmmm  &  \\
\cline{2-3}
      \rowcolor{Bluee} & Totals    & mmmmm & mmmmm &  \\[2mm]
\end{tabular}
```

Table title

Description	Column 1	Column 2
Row one	mmmmm	mmmm
Row two	mmmm	mmm
Row three	mmmmm	mmmmm
Row four	mmmmm	mmmm
Totals	mmmmm	mmmmm

9-3-20

Мы надеемся, что эти примеры продемонстрировали, насколько гибким инструментом являются команды для работы с цветом, которые позволяют придать материалу почти любой внешний вид. Единственная опасность заключается в том, чтобы не растеряться от многообразия возможностей. Цвет должен помогать читателю лучше воспринимать информацию — не позволяйте ему становиться помехой.

9.4 Цветные слайды, полученные с помощью LaTeX'а: класс seminar

Многие пользователи LaTeX'а хотят сохранить высокое качество набираемого TeX'ом материала при создании слайдов для презентаций. Такая возможность была впервые реализована в отдельном пакете SliTeX, но он обладает рядом недостатков:

- в нем могло использоваться лишь ограниченное число специальным образом масштабированных шрифтов из семейства Computer Modern, а добиться возможности пользоваться другими шрифтами было очень сложно;
- пользователь должен был иметь два отдельных файла, один для служебной информации, а второй — для самих слайдов;
- управление цветом и масками было ограниченным и грубым;
- существовал единственный «стиль» оформления слайдов, а способ создания другого стиля оформления (например, размещение на каждом слайде логотипа) не был описан в документации.

Появившийся в конце 1993 г. LaTeX 2_ε содержал простой LaTeX'овский класс slides, который был предназначен для эмуляции SliTeX'а и позволял избежать дополнительных неудобств, связанных с необходимостью загрузки

отдельного пакета. Однако через какое-то время появились более совершенные LaTeX'овские пакеты, одним из которых является foiltex Джима Хафнера (Hafner, 1992). Однако здесь мы подробно рассмотрим пакет seminar Тимоти ван Зандта (van Zandt, 1993b). В случае его использования в сочетании с PostScript'овским принтером и макрокомандами PSTricks, описанными в гл. 4, seminar позволяет реализовать почти любой мыслимый элемент оформления, включая:

- украшенные орнаментом рамки и колонтитулы;
- сочетание слайдов с вертикальной и горизонтальной ориентацией в одном документе;
- раскрашенный текст и таблицы;
- комбинирование аннотаций и слайдов;
- разбиение слайдов на «главы» и список слайдов;
- маски для выделения отдельных цветов.

Класс seminar представляет собой обычный LaTeX'овский класс, который можно использовать почти со всеми остальными LaTeX'овскими пакетами (предназначенными для изменения шрифта, включения графического материала или расширения возможностей для набора математики, например, $\mathcal{AMS}$-LaTeX). Хотя его основная задача заключается в создании прозрачных пленок, он также может на основе того же исходного файла генерировать сопроводительные записи.

9.4.1 Использование класса seminar

Применить его просто: начните ваш документ обычным образом, указав seminar в качестве имени используемого класса, а потом вводите то, что вы хотите разместить на пленках, внутри соответствующих окружений.

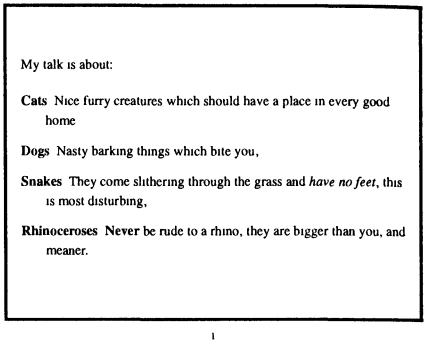
```
\begin{slide} ...\end{slide}
\begin{slide*}...\end{slide*}
```

Окружение `slide` служит для создания слайдов с горизонтальной ориентацией, а `slide*` — для слайдов с вертикальной ориентацией. Необязательный параметр `portrait` позволяет получить на выходе документ с вертикальной ориентацией вместо принятой по умолчанию горизонтальной. Команды изменения ориентации слайдов добавляются в зависимости от установленного значения этого параметра. Поскольку сам LaTeX не выполняет повороты, вы должны определить соответствующие команды `\special`, чтобы заставить ваш драйвер поворачивать страницы на 90 градусов: seminar не интегрирован со стандартным LaTeX'овским графическим пакетом. В случае PostScript'овского драйвера, такого, как dvips, команды поворота могут быть заданы в локальном файле настроек, описанном ниже (разд. 9.4.6). Если же драйвер не умеет

выполнять повороты, вы все же можете напечатать слайды с горизонтальной ориентацией, если сумеете настроить сам принтер на работу в таком режиме.

Принятое по умолчанию оформление показано в следующем примере:

```
\documentclass{seminar}
\usepackage{times}
\begin{document}
\begin{slide}
 My talk is about:
 \begin{description}
 \item[Cats] Nice furry creatures which
   should have a place in every good home
 \item[Dogs] Nasty barking things which
   bite you;
 \item[Snakes] They come slithering through
   the grass and \emph{have no feet}; this
   is most disturbing;
 \item[Rhinoceroses] \textbf{Never} be rude
   to a rhino; they are bigger than you,
   and meaner.
 \end{description}
\end{slide}
\end{document}
```



Большинство слайдов ничуть не сложнее только что представленного, поэтому могут быть получены с помощью таких стандартных LaTeX'овских окружений, как `itemize`, `enumerate` и `tabular`.

9.4.2 Стили рамок

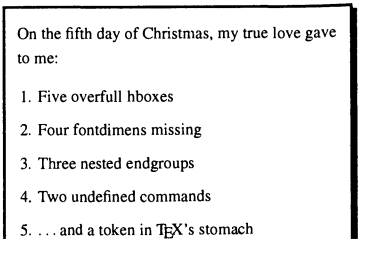
Класс seminar делает различие между *стилем страницы* и *рамкой слайда*. Первый, как в обычном LaTeX'е, определяет текст верхних и нижних колонтитулов, а вторая описывает бокс или рамку, окружающую текст слайда. Существует несколько стилей оформления рамок слайда, которые можно задавать командой `\slideframe`, например, `\slideframe{shadow}`. Ниже перечислены некоторые из предопределенных стилей рамок (часть из них предполагает наличие PostScript'овского принтера и загрузки пакета fancybox):

`none`

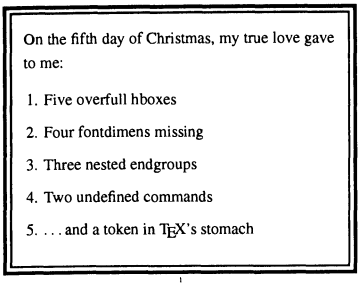
On the fifth day of Christmas, my true love gave to me:

1. Five overfull hboxes
2. Four fontdimens missing
3. Three nested endgroups
4. Two undefined commands
5. . . . and a token in TeX's stomach

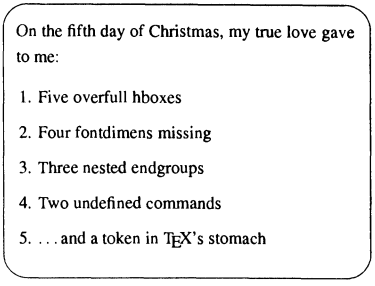
`shadow`



`double`

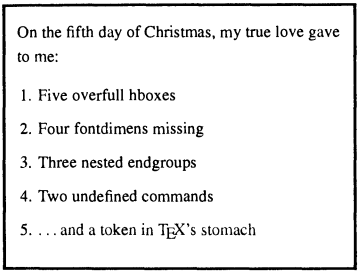


`oval`

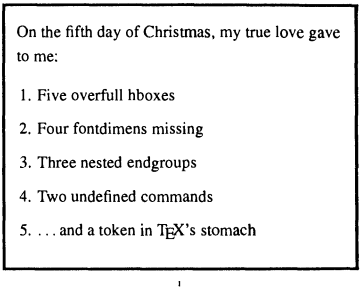


Аналогично, разнообразные стили оформления страниц (верхних и нижних колонтитулов) доступны благодаря команде `\pagestyle`, например, `\pagestyle{plain}`. Предопределены следующие стили:

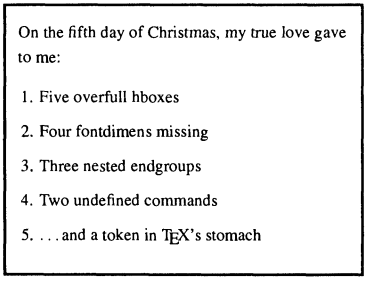
`empty`



`plain` (номера страниц снизу)



`align` (номера или + по углам)



Можно определять новые стили оформления рамок и страниц. Приведенные в этой главе примеры (см. цветную вклейку XX) были созданы для использования в Европейской организации ядерных исследований (CERN).

9.4.3 Комментарии и выборки

Ваши слайды легко снабжать собственными комментариями. Их можно просто размещать между окружениями `slide` или помещать внутрь специального окружения `note`. Внутри этих комментариев вы можете использовать любые

LaTeX'овские команды и в случае необходимости поместить там целую статью. Для вывода слайдов на печать применяются разнообразные параметры:

`slidesonly` печатаются только слайды;

`notesonly` печатаются только комментарии;

`notes` слайды перемежаются комментариями;

`article` комментарии набираются как обычный LaTeX'овский документ, а слайды представляются в виде рисунков (уменьшенные на 50%).

`\slideplacement{`*type*`}`

Команда `\slideplacement` влияет на то, как слайды располагаются в случае использования формата `article`. Аргумент *type* может принимать следующие значения:

`float` слайды являются плавающими (значение по умолчанию);

`float*` слайды являются плавающими, но при наборе в две колонки они занимают обе колонки;

`here` слайды располагаются там, где они встретились в тексте.

Дальнейшие подробности контроля за взаимодействием между слайдами и комментариями приведены в руководстве *User's Manual*, входящем в дистрибутив пакета seminar.

Отдельные слайды могут быть отобраны и включены или исключены из рассмотрения с помощью команд `\onlyslides` или `\noteslides`, которые в качестве параметра получают разделенный запятыми список слайдов. Этот список может содержать числа, диапазоны (например, 5–10) или LaTeX'овские команды `\ref`, ссылающиеся на метки слайдов `\label`.

9.4.4 Управление размером слайдов, шрифтами и масштабом

Пользователь может изменить любой из следующих параметров применительно к отдельному слайду или документу в целом:

- высоту и ширину слайда;
- правое, левое, верхнее и нижнее поля;
- тип выравнивания текста (по умолчанию принято выравнивание с рваным правым краем);
- разбиение страниц путем изменения параметра допустимости избыточного материала;
- расстояние между строками;
- размер шрифта и сам шрифт.

Процедура изменения принятых по умолчанию значений подробно объясняется в руководстве *User's Manual* (van Zandt, 1993b).

Поскольку работа пакета seminar основана на увеличении размера страниц, опытным пользователям следует прочитать руководство, чтобы узнать, как можно устанавливать и изменять TeX'овские параметры размеров. Большинству пользователей можно об этом не заботиться — все равно в таких командах, как `\includegraphics`, вы должны всегда выражать желаемую «ширину» и «высоту» в долях длины строки (`\linewidth` или `\columnwidth`).

9.4.5 Шрифты

Поскольку размер шрифтов на слайдах превышает размер шрифтов в обычном тексте (статьях, руководствах и т. д.), требуемых размеров некоторых шрифтов (особенно, математических) часто не существует. Такие программы, как dvips или emTeX, стараются создать эти шрифты, но не всегда могут найти необходимые для этого исходные METAFONT'овские файлы. Поэтому имеет смысл использовать масштабируемые PostScript'овские шрифты. В PostScript'овском формате Type1 доступно все семейство шрифтов Computer Modern: либо наборы шрифтов компаний Blue Sky или Y&Y, либо шрифты (немного хуже проработанные) из пакета bakoma, написанного Василием Малышевым.

9.4.6 Локальный файл настроек

В начале своей работы пакет seminar всегда просматривает TeX'овский путь для поиска и пытается найти файл `seminar.con`. Это предоставляет пользователю удобную возможность изменить принятые по умолчанию значения. Файл `seminar.con` может содержать любые LaTeX'овские команды, включая команды загрузки других пакетов.

Цветная вклейка XX была получена с использованием варианта `seminar.con`, настроенного на стиль оформления, принятый в CERN'е. Давайте изучим содержимое этого файла. Сначала мы определяем задающий горизонтальную ориентацию макро для драйвера dvips. Без него команда `\printlandscape` по умолчанию просто предупреждала бы пользователя о том, что файл следует печатать в горизонтальной ориентации:

```
1 \renewcommand{\printlandscape}{\special{papersize=297mm,210mm}}
```

Мы хотим иметь возможность включать графический материал и работать с цветом. Таким образом, нам потребуются следующие пакеты (имейте в виду, что pstcol, в свою очередь, загружает PSTricks):

```
2 \RequirePackage[usenames]{pstcol}
3 \RequirePackage{graphicx}
```

Поскольку этот файл настроек предназначен для PostScript'овского принтера и мы хотим воспользоваться такими PostScript'овскими эффектами, как закругленные углы бокса, мы загрузим некоторые дополнительные пакеты:

```
4 \RequirePackage{semcolor,fancybox}
```

Так как данный стиль оформления предназначен для использования в Европе, где стандартным размером бумаги является A4, мы должны задать несколько параметров размера и вновь запустить программу, чтобы изменить значения внутренних параметров:

```
\setlength\paperwidth{210mm}
\setlength\paperheight{297mm}
\input{sem-page.sty}
```

Теперь надо установить размер и местоположение слайда. Обратите внимание, что размеры полей являются командами, а не LaTeX'овскими длинами, и, следовательно, переопределять их надо с помощью команды `\renewcommand`:

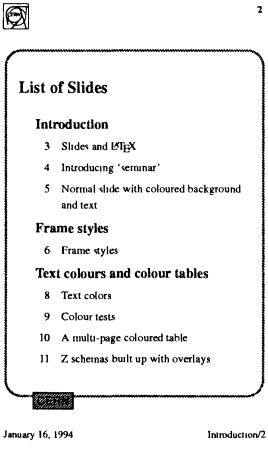
```
\setlength\slidewidth{205mm}
\setlength\slideheight{150mm}
\renewcommand\slideleftmargin{.8in}
\renewcommand\sliderightmargin{.8in}
```

Для группирования слайдов в «разделы», построения списка содержания и т. д. мы используем другой пакет, прилагающийся к пакету seminar:

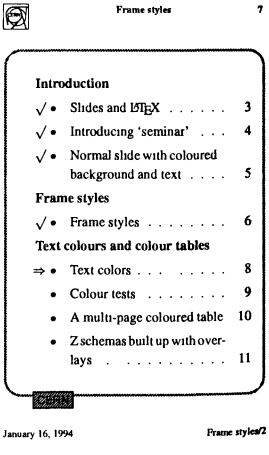
```
\RequirePackage{slidesec}
```

Пакет slidesec предоставляет нам доступ к различным внутренним командам; некоторые из них будут использованы ниже. Например, мы можем оформить перечень слайдов в двух разных стилях:

```
\begin{slide*}
\listofslides
\end{slide*}
```




```
\begin{slide*}
\Slidecontents
\end{slide*}
```



Для задания заголовков слайдов предусмотрена команда `\slideheading`. Мы изменим ее таким образом, чтобы заголовок был набран с «тенью», переопределив команду `\makeslideheading`:

```
\newsavebox\headbox
\renewcommand{\makeslideheading}[1]{%
 \gdef\theslideheading{#1}%
 \begin{lrbox}{\headbox}\large\bfseries#1\end{lrbox}%
 \begin{center}\shadowbox{\usebox\headbox}\end{center}}
```

Также определена команда \slidechapter:

```
\newcounter{slchapter}
\newcommand\l@chapslide[1]{\slide@undottedcline{#1}{}}
\newcommand\slidechapter[1]{%
   \begin{slide}%
    \addtocontents{los}{\protect\l@chapslide{\protect\large\bfseries #1}}%
   \end{slide}%
   \renewcommand\thechapterheading{#1}%
   \setcounter{slchapter}{0}}
\newcommand\thechapterheading{}
```

Это дает возможность пользователю разбивать слайды на группы. В случае стиля разделов, разработанного для CERN'а, заголовок главы слайдов (хранящийся в \thechapterheading) печатается в нижнем правом углу.

Теперь с помощью команды \newpagestyle мы определим предназначенный для CERN'а стиль оформления страниц. Простейший стиль «CERN» просто печатает регистрационные отметки «+», дату и номер страницы.

```
\newpagestyle{cern}
  {{\color{black}\small \textbf{+}\hfil  \today  \hfil\textbf{+}}}
  {{\color{black}\small \textbf{+}\hfil \thepage \hfil\textbf{+}}}
```

Стиль разделов, разработанный для CERN'а, содержит в левом верхнем углу логотип (PostScript'овский файл cernlogo.eps), сверху посередине — заголовок слайда, в правом верхнем углу — номер слайда. В нижнем левом углу мы печатаем текущую дату, а в нижнем правом — заголовок текущего раздела:

```
\newpagestyle{cernsections}
 {{\color{black}\small
   \raisebox{-.5cm}[0cm][0cm]{\includegraphics[height=.8cm]{cernlogo.eps}}%
   \hfil\textbf{\theslideheading}\hfil\textbf{\thepage}}}
 {{\small\color{black}\today \hfil\thechapterheading/\arabic{slchapter}}}
```

Чтобы правильно повернуть текст заголовка при смене ориентации слайдов, нам необходимо воспользоваться следующей командой:

```
\rotateheaderstrue
```

Что касается рамок слайдов, мы добавили к ним слово «CERN», помещенное в окрашенный бокс, который располагается в левом нижнем углу стандартного бокса слайда. Это выполнено с помощью макро PSTricks (автоматически загружаемого пакетом pstcol). Команды работы с цветом заданы с помощью параметра usenames пакета pstcol. Текущим цветом шрифта является \SlideFront (определяемый далее):

```
\newslideframe{cern}{{\SlideFront
 \boxput(-0.7,-1.11){%
   \psframebox[linecolor=black,fillcolor=ForestGreen,fillstyle=solid]{%
       \mbox{\normalsize\sffamily\color{black}CERN}}}{#1}%
   \color{black}}}
```

Наконец, мы делаем так, чтобы каждый слайд начинался с текущего цвета фона, а счетчик `slchapter` увеличивался соответствующим образом:

```
\renewcommand\everyslide{%
  \SlideFront
  \stepcounter{slchapter}%
}
```

Цвет слайдов настраивается с помощью команды `\SlideColors`, имеющей два параметра, указывающих цвета шрифта и фона. Для случая черного и белого цветов определен псевдоним.

```
\newcommand\SlideColors[2]{\gdef\SlideFront{\color{#1}}%
   \slideframe{\Framedefault}%
   \slideframe*[\psset{fillcolor=#2,fillstyle=solid}]{blueframe}}
\newcommand\blackandwhite{\SlideColors{black}{white}}
```

По умолчанию рамки окрашиваются PostScript'овским оператором `setgray`. Чтобы это изменить, мы должны воспользоваться необязательным параметром `\newslideframe`. Например, мы можем определить простую рамку синего цвета следующим образом:

```
\newslideframe{blueframe}
   [\psset{linecolor=NavyBlue,linewidth=\slideframewidth,%
         framesep=\slideframesep,cornersize=absolute,linearc=.5cm}]%
   {\psframebox{#1}}
```

Следующие значения по умолчанию предназначены для представления материала и логотипа CERN в виде желтого текста на синем фоне:

```
 \pagestyle{cernsections}
 \newcommand\Framedefault{cern}
 \SlideColors{Yellow}{RoyalBlue}
```

Эти команды окрашивают только пространство внутри рамки, поэтому чтобы изменить цвет всего слайда, вам потребуется дополнительная команда `\pagecolor`.

Чтобы расширить возможности пользователя по настройке пакета, загружается дополнительный файл `seminar.cfg`, если он существует:

```
\InputIfFileExists{seminar.cfg}
  {\typeout{Loading user configuration of seminar}}{}
```

Пример использования описанной настройки показан на цветной вклейке XX.

Чтобы изменить принятые по умолчанию значения цветов на черный и белый, вы можете перед началом документа добавить следующие команды `\begin{document}`:

```
\documentstyle{seminar}
\renewcommand{\SlideFront}{\color{black}}
\begin{document}
\begin{slide}
```

Если вы хотите сохранить управление цветами «SlideColors», чтобы задавать цвет фона ваших слайдов, и вам достаточно простой рамки, отсутствия логотипов и т. д., то можете воспользоваться следующими настройками:

```
\documentstyle{seminar}
\pagestyle{plain}
\renewcommand{\SlideColors}[2]{%
 \renewcommand{\SlideFront}{\color{#1}}%
 \slideframe%
  [\psset{fillcolor=#2,fillstyle=solid}]%
  {scplain}%
}
\begin{document}
\SlideColors{Red}{Yellow}
\begin{slide}
...
```

Если подобный этому файл `seminar.con` является частью настройки LaTeX'а, пользователи, которым не нравится данная система, могут совершенствовать ее на свой страх и риск до произвольного уровня сложности. В простейшем случае они могут просто создать свой собственный файл `seminar.con` и поместить его в директории, где лежат файлы с их слайдами:

```
% Ничего не делающий файл seminar.con
\endinput
```

Полное описание синтаксиса многих других команд, которые есть в вашем распоряжении, приведено в руководстве *User's Guide* пакета seminar. Вероятно, вам также следует прочесть гл. 4, посвященную пакету PSTricks!

9.5 Цвет в типографском деле и цветоделение

В типографском деле обычно используется либо цветовая модель CMYK в случае полноцветной печати, либо печать одной дополнительной краской в случае одно- или двухцветной печати. Последняя обычно выбирается в соответствии с диаграммами Munsell и Pantone. В последнее время для согласования красок стали чаще использоваться системы Focoltone и Trumatch.

Язык PostScript применяется почти во всех ситуациях благодаря высокому качеству получаемого материала, причем PostScript второго уровня обеспечивает полную поддержку цвета, включающую не только такие простые модели, как RGB, CMYK и HSB, но также систему CIE и другие специальные цветовые пространства. Подробное описание этих моделей и алгоритмы преобразования из одной в другую исчерпывающим образом описаны в Adobe Systems (1990), с. 176–199. Очень интересные дискуссии на тему цвета в PostScript'е содержатся в Kunkel (1990) и McGilton and Campione (1992). Полезный материал можно также найти в Reid (1988). Если вы читали старые книги, вы должны были заметить, что полный PostScript второго уровня содержит много важных новых команд, которые существенно облегчают осуществление цветоделения.

Вопросы, возникающие при работе с TeX'ом в индустрии профессиональной печати, обсуждаются в статье (Sofka, 1994). В Duggan (1994) цветоделение

рассматривается применительно к TEX'у, а в работе Sowa (1994) описывается, как выполнять цветоделение на уровне растровых изображений, выдаваемых TEX'ом.

Пользователь LATEX'а в значительной степени избавлен от сложностей, связанных с реальным процессом печати. Документ, который содержит цветной материал, может быть набран и обработан драйвером, подобным dvips для получения цветного документа на PostScript'е, который можно просмотреть на экране или распечатать на цветном принтере. Главная трудность заключается в том, что точность воспроизведения цветов обычными струйными принтерами не достаточна для подготовки оригинала для профессиональной печати. Однако эти принтеры очень хорошо подходят для создания прозрачных пленок, проспектов, плакатов и т. д., поэтому ваши усилия по совершенствованию навыков использования цвета в LATEX'е несомненно будут полезны для выполнения работ такого рода.

9.5.1 Цветоделение

Для создания «настоящей» цветной книги или журнала с применением офсетной печати требуются четыре варианта каждой страницы, которые содержат оттенки серого цвета, соответствующие пропорциям голубого, пурпурного, желтого и черного цветов. Каждая страница пропечатывается четыре раза, причем каждый раз наносится краска другого цвета. На цветной вклейке XXI, где вверху нарисованы пять олимпийских колец, а внизу — многоцветный эллипс, показано, как добавление краски разных цветов придает цветному изображению законченный вид. Мы начали с нанесения голубой краски (сверху слева), затем добавили пурпурную (сверху справа), желтую (снизу слева) и, наконец, черную краску (снизу справа) для получения полноцветного изображения. Видны четыре стадии процесса и результат.

Цветной материал обычно наносится на специальную пленку с высокой точностью, так как процесс печати каждой страницы повторяется четыре раза и совмещение оттисков должно быть полным. Некоторые системы набора могут создавать эти четыре цветоделенные фотоформы автоматически, но чаще это получается в результате работы с PostScript'ом. Профессиональные пакеты для набора и работы с графикой, наподобие Adobe Illustrator, Corel Draw или Adobe Photoshop, выполняют цветоделение непосредственно, а программа Adobe Separator может работать с произвольным описанием цвета на PostScript'е. Эффективность таких программ зависит от того, как описаны цвета в PostScript'овском файле и как PostScript второго уровня обеспечивает для этого хорошую поддержку.

При подготовке материала для цветоделения пользователь должен быть знаком с несколькими специальными понятиями:

- «индивидуальные» цвета;
- оттиск, полученный отдельным нанесением красок, и оттиск, полученный последовательным наложением красок;
- захват краски.

Печатные работы обычно делятся на те, которые можно выполнить в рамках модели CMYK (составные цвета), и те, где требуется указать определенный цвет (индивидуальный цвет). В то время как страницы, содержащие составные цвета, печатаются путем четырехкратного нанесения необходимых пропорций соответствующих цветов, страницы с индивидуальным цветом печатаются отдельно с применением специальной краски (обычно из стандартного набора, наподобие Pantone). LaTeX'овский пакет для операций с цветом не поддерживает работу с индивидуальными цветами.

Рассмотрим простой пример цветного изображения, состоящего из желтого круга на синем фоне (вроде цветной вклейки V). На каком-то этапе необходимо будет принять решение, создавать этот рисунок с помощью соответствующих значений цветов C, M, Y и K[1], а потом печатать их, последовательно накладывая друг на друга (это называется оттиском, полученным последовательным наложением красок), или оставить на синем фоне незакрашенное пространство круглой формы, в котором будет напечатан желтый круг (это называется оттиском, полученным отдельным нанесением красок). Модель CMYK (принятая по умолчанию в PostScript'е) подразумевает печать последовательным наложением красок (поскольку вся модель в целом действует на аддитивном принципе), но индивидуальные цвета естественным образом требуют отдельного нанесения красок, так как указанный точный цвет не должен ни с чем смешиваться. В PostScript'е для включения или выключения режима последовательного наложения красок существует оператор `setoverprint`. Этим оператором можно воспользоваться для управления каким-нибудь dvi-to-PostScript драйвером при выполнении сложных манипуляций с цветом.

Вне зависимости от того, какой метод мы используем, мы столкнемся с еще одной проблемой: совмещением последовательных отпечатков на листе бумаги. Даже самые точные механизмы испытывают проблемы при выравнивании материала, размер которого указан в масштабных пунктах, в случае разрешения, превышающего 2500 dpi. Поэтому на практике для качественной цветопередачи применяется *захват краски*. Если вернуться к примеру с желтым кругом на синем фоне, то, осуществляя захват краски, мы либо немного уменьшим размер незакрашенного пространства на синем поле (сужающийся захват), либо чуть-чуть увеличим размер круга (расширяющийся захват), и, таким образом, создадим небольшое наложение, чтобы избежать связанных с выравниванием трудностей.

Рассмотрение всего многообразия вопросов, связанных с высококачественной цветной печатью (таких, как описание трафаретных печатных форм), выходит за рамки данной книги, и мы настоятельно рекомендуем читателям получить квалифицированную консультацию, прежде чем приступать к серьезным полноцветным проектам на основе LaTeX'а. Хотя Майкл Софка (Sofka, 1994) с помощью собственного dvi-драйвера и другие авторы с помощью программы Textures для ОС Macintosh показали, что LaTeX может справиться с серьезной

[1] Общепринятый порядок цветов на самом деле другой: первым идет желтый, за ним голубой, пурпурный и последним черный.

полноцветной задачей, пользователи, чьи потребности ограничены относительно несложной цветной печатью (такой, как двухцветная книга, в которой все колонтитулы синего цвета и все таблицы напечатаны на синем фоне), вполне могут обойтись без коммерческого программного обеспечения. Цветоделение в рамках модели CMYK может быть выполнено с помощью пакета color и драйвера dvips с использованием только операторов языка PostScript первого уровня. Мы опишем это в следующем разделе.

9.5.2 Цветоделение с помощью LaTeX'а и драйвера dvips

Принцип получения драйвером dvips простого цветоделенного вывода на основе результата работы TeX'а состоит в том, что каждая итоговая страница создается четыре раза при указании управляющего параметра командной строки `-b 4` или параметра `b 4` в файле настроек и загрузки заголовочного файла, который переопределяет операторы работы с цветом для каждой из этих четырех страниц. В заголовочном файле `colorsep.pro` (распространяется вместе с dvips, обновляется Себастианом Ратцем и в значительной степени заимствован из работ Kunkel (1990) и Reid (1988); текст самого файла приведен в приложении А.4) для увеличения счетчика в начале обработки каждой страницы используется оператор `bop-hook` (см. разд. 11.2.5 на с. 498) и, таким образом, осуществляется проверка того, какая именно страница (C, M, Y или K) сейчас создается. После этого выполняется переопределение оператора `setcmykcolor` для получения одного из четырех цветов в оттенках серого. Перед выполнением этого процесса цвета RGB преобразуются в цвета CMYK. Оператор `setgray` активизируется только на странице, соответствующей черному цвету (K). К сожалению, эта система не умеет обрабатывать цветные растровые изображения.

Результат цветоделения примера **Внимание!** представлен на рис. 9.1. В нем бокс имеет цвет «ForestGreen», которому в модели CMYK соответствуют значения «0.91 0 0.88 0.12». Обратите внимание, что пурпурная страница (M) абсолютно пуста, поскольку ни зеленому боксу, ни черному тексту не требуется добавлять пурпурный цвет.

Для достижения большего контроля за цветоделением совместно с dvips или любым другим dvi-драйвером, способным загружать заголовочные файлы, может использоваться написанный Грэмом Фриманом пакет aurora. Он также действует, переопределяя PostScript'овские команды для работы с цветом, но страницы C, M, Y и K, соответствующие каждому из цветов, создаются с помощью своих собственных заголовочных файлов. Общий заголовочный файл (`aurora.pro`) загружается вместе с одним из `cyan.pro`, `magenta.pro`, `yellow.pro` или `black.pro`. Таким образом, команда драйверу dvips вывести

Голубой	**Внимание!**
Пурпурный	

Желтый	**Внимание!**
Черный	**Внимание!**

Рис. 9.1. Пример простого цветоделения.

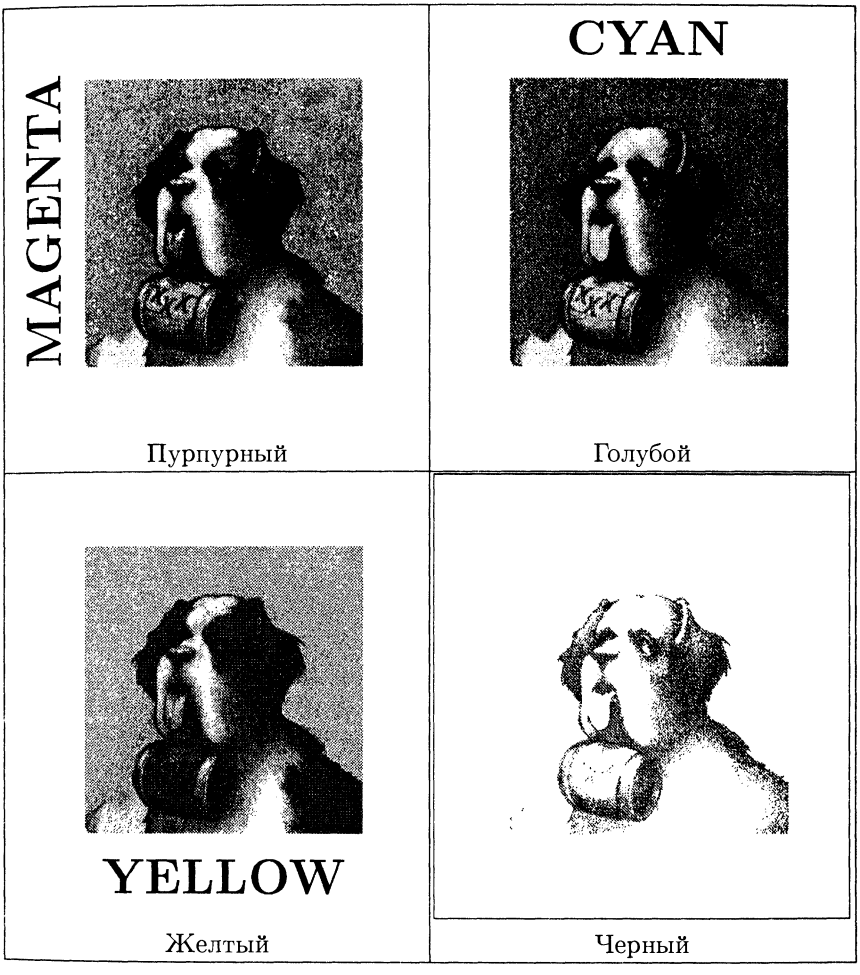


Рис. 9.2. Цветоделение растрового рисунка с помощью пакета aurora.

соответствующую пурпурному цвету информацию из файла `text.dvi` в PostScript'овский файл `text-magenta.ps` будет иметь вид

```
dvips text -h aurora.pro -h magenta.pro -o text-magenta.ps
```

Пакет aurora умеет обрабатывать цветные растровые изображения во многих форматах, но все же не предоставляет такого количества возможностей, как команда `colorimage` языка PostScript второго уровня (в частности, не поддерживается составной цвет). На рис. 9.2 показано, как выглядит цветоделенный вывод, полученный с помощью aurora. Изображение собаки представлено в оттенках серого, соответствующих количеству голубого, пурпурного, желтого и черного цветов. Результат еще отчетливее виден на рис. 9.3, где показан небольшой фрагмент того же самого рисунка (часть бочонка на шее у собаки).

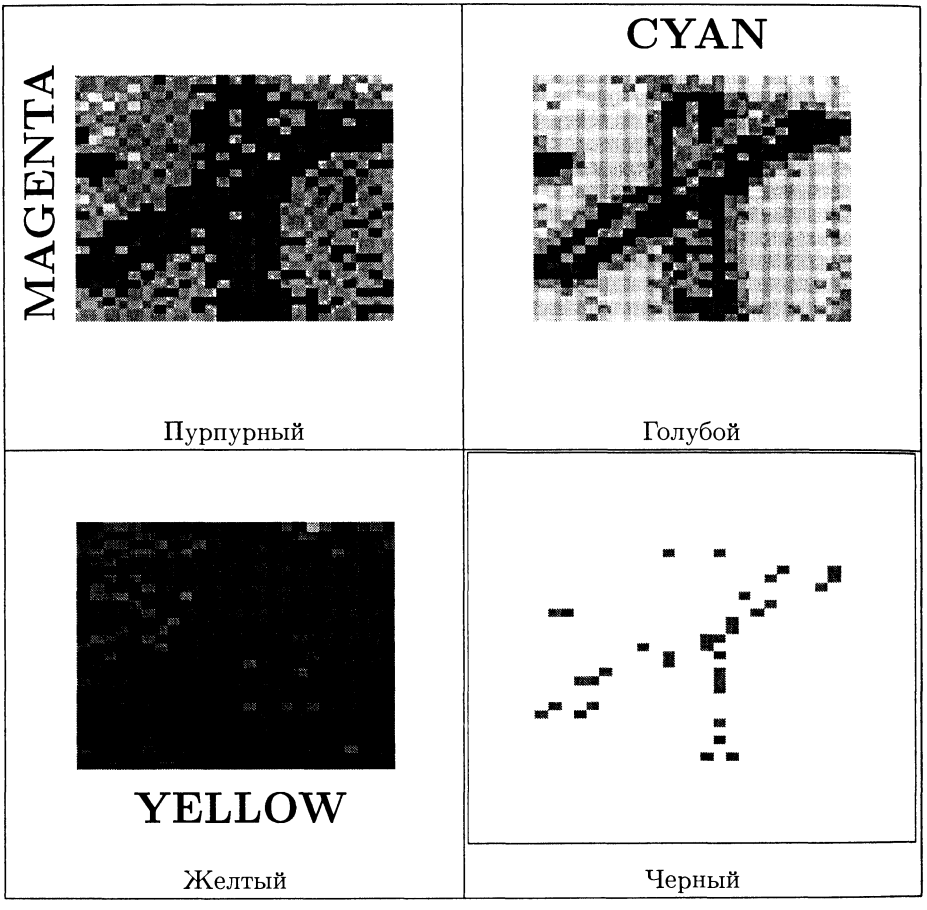


Рис. 9.3. Фрагмент растрового изображения, полученного в результате цветоделения.

Индивидуальный цвет может быть получен путем точного задания C-, M-, Y- и K-компонент и предоставления заголовочного файла, соответствующего цвету, который задает некоторую точку в CMYK-пространстве требуемого цветоделения. Если текущий цвет в документе совпадает с этим цветом, то данный объект будет напечатан. Такой метод отбора индивидуальных цветов для вывода требует, чтобы описывающие индивидуальный цвет параметры точно соответствовали характеристикам цвета, используемого в документе. Если они не совпадают, то не будет получено никакого результата, тогда как при обычном методе цветоделения компонента текущего первичного цвета определит, какая *доля* изображения войдет в результат.

Использование шрифтов в формате PostScript

В этой главе мы рассмотрим наиболее общий тип графических объектов в документах — символы, из которых построены слова. Начертание символов непосредственно не относится к системе TEX; все, что нужно знать TEX'у о символах — это информация об их метрике (ширине, высоте и т. п.). Создание графического образа страницы является задачей драйвера устройства, который для этого использует выходной файл TEX'а (`dvi`-файл). Здесь-то драйверу и понадобится информация о глифах — действительной форме символов. Эта информация хранится в так называемых шрифтах — коллекциях глифов, файлы которых могут иметь различный формат. Таким образом, любой существующий шрифт может быть использован в TEX'е, при условии что:

- необходимая TEX'у метрическая информация уже существует или может быть создана;
- существует драйвер, который умеет интерпретировать `dvi`-файлы и, кроме того, понимает формат, в котором хранятся шрифты.

В течение долгого времени, однако, число шрифтов, практически доступных для пользователей TEX'а, было меньше числа, определяемого указанными выше ограничениями.

Дональд Кнут в качестве дополнения к TEX'у разработал программу METAFONT для генерации шрифтов, используемых совместно с TEX'ом (графические возможности METAFONT'а были кратко рассмотрены в гл. 3). В течение некоторого времени для пользователей TEX'а были доступны только шрифты, созданные с помощью этой программы, из-за чего документы, набранные с помощью TEX'а или LATEX'а, имели легко узнаваемый вид (главным образом из-за использования семейства Computer Modern). В последние годы дизайнеры разработали с помощью METAFONT'а еще ряд шрифтов, однако полное число таких шрифтов, предлагаемых TEX'овскому сообществу, было и остается намного меньше числа шрифтов, доступных в других издательских системах (число коммерческих шрифтов в формате PostScript исчисляется тысячами).

Хотя PostScript'овские шрифты можно было использовать уже на самой ранней стадии развития TEX'а, их интеграция в такие макропакеты, как LATEX, была затруднена, поскольку доступ к шрифтам был жестко привязан к коду макро и его изменение требовало глубокого понимания внутреннего устройства TEX'а. Ситуация изменилась в 1989 г. с появлением так называемой новой схемы выбора шрифтов (new font selection scheme, NFSS), более поздняя версия которой (NFSS2) вошла в современный стандарт LATEX'а.

В современном LATEX'е стал возможным простой доступ к большому набору PostScript-шрифтов, причем (что важно) не нужно ни модифицировать внутренний код LATEX'а, ни строить собственные форматные файлы. В то время как в LATEX2.09 для создания документов, которые можно было бы читать на любых платформах TEX'а, требовалось использовать шрифты Computer Modern, LATEX'овские документы могут включать и другие шрифты, при этом не нарушая «мобильности» документа (естественно, при условии, что эти шрифты доступны и конечному пользователю). И даже если эти шрифты недоступны, документы все равно будут обработаны, поскольку NFSS автоматически выберет шрифты-заменители в случае, когда какие-либо из шрифтов отсутствуют.

В NFSS нет ограничений на число шрифтов, которое можно подключить к системе LATEX, и, подключив их один раз, их все можно использовать без изменения формата LATEX'а (хотя, возможно, не все их удастся использовать в одном документе, поскольку большинство программ TEX'а имеет ограничение на максимальное число шрифтов, одновременно присутствующих в одном документе). Таким образом, вы можете запросто набирать ваши документы, используя классические или современные гарнитуры, и объединить превосходные возможности TEX'а в форматировании документов с любыми профессионально разработанными гарнитурами этого столетия, большинство которых доступно в формате PostScript. Однако подключение новых PostScript'овских шрифтов к TEX'у требует некоторой предварительной работы со стороны пользователя, и в этой главе мы опишем основные этапы этой работы.

Сначала мы рассмотрим простейший случай, когда PostScript'овские шрифты уже подготовлены для использования в TEX'е, и рассмотрим систему PSNFSS (коллекцию небольших пакетов и дополнительных файлов для LATEX'а), которая облегчает подключение большого числа широко распространенных PostScript'овских шрифтов, входящих в большинство систем TEX'а. Затем мы опишем, какие шаги следует предпринять, чтобы установить компоненты PSNFSS для дополнительных шрифтов (которые вы можете приобрести), и объясним, как получить к ним доступ из вашего файла класса или документа. В последней части главы мы рассмотрим, как подготовить дополнительные PostScript'овские шрифты к подключению к системе TEX и как модифицировать существующую установку. Даже если вы не собираетесь предпринимать никаких подобных действий сами, эта информация позволит вам понять, как устроена эта технология, в чем состоят ее преимущества и ограничения и как все это работает вместе.

Эту часть главы мы начнем с того, что расскажем о том, как TEX обращается со шрифтами, и обсудим типы файлов, с которыми приходится встре-

чаться в системе TEX, а также некоторые полезные утилиты для манипуляции этими файлами. Затем мы рассмотрим некоторые важные концепции технологии PostScript-шрифтов, такие, как типы PostScript'овских шрифтов, способы представления в них метрической информации и способы кодирования PostScript'овских шрифтов, т. е. как в конечном счете в этих шрифтах осуществляется доступ к конкретному символу. Затем мы рассмотрим соглашение об именах PostScript'овских шрифтов, которое разработал и поддерживает Карл Берри. Этот материал, возможно, несколько суховат, но важен, если мы хотим работать одновременно с большим числом шрифтов более чем в одной операционной системе.

После того как мы проясним общие концепции, мы будем готовы применить наши знания и рассмотрим задачу подключения PostScript'овских шрифтов к системе TEX. Мы обсудим средства и способы получения необходимой для TEX'а метрической информации из метрики, поставляемой со шрифтами в формате PostScript, покажем, как использовать виртуальные шрифты для перекодировки существующих шрифтов с тем, чтобы они соответствовали стандартным кодировкам LATEX'а, и т. д. Наконец, в заключение, мы рассмотрим подключение так называемых шрифтов Multiple Master, которые использовались при наборе этой книги.[1]

10.1 Использование заранее сконфигурированных PostScript-шрифтов

Схема выбора шрифтов в LATEX'е подробно описана в гл. 7 книги Goossens et al. (1994), а краткое ее объяснение дано в файле `fntguide.tex`, являющемся частью дистрибутива LATEX. В последующей части этой главы мы будем считать, что читатель имеет доступ к более подробному описанию, приведенному в этих документах.

Действие по умолчанию, связанное с любой из команд NFSS, может быть изменено с помощью макро `\renewcommand`, а каждая из деклараций имеет соответствующую команду с суффиксом `default`. Например, если у вас есть семейство шрифтов Courier и вы хотите использовать их по умолчанию (переключаясь на них с помощью команды `\ttfamily`), вам надо указать это в преамбуле документа или в стилевом файле к пакету:

```
\renewcommand{\ttdefault}{courier}
```

(Заметим, что при этом могут возникнуть проблемы с кодировкой, о чем мы будем говорить ниже.) Но как нам действительно сделать так, чтобы этот Courier заработал? Здесь возможны следующие три ситуации:

1. Нас вполне удовлетворит использование шрифтов из стандартной поставки, распространяемой с LATEX'ом в рамках пакета PSNFSS. Этот вариант мы обсудим ниже.

[1] Речь идет об оригинальном издании на английском языке. — *Прим. перев.*

2. Метрические файлы и файлы описания для семейства шрифтов, которое мы хотим подключить, уже созданы кем-то ранее, так что остается установить их и создать файл пакета. Этот вариант мы обсудим в разд. 10.1.3.
3. У нас имеются только метрические файлы в формате Adobe (`afm`-файлы), и нам нужно создать весь набор метрических и других файлов, необходимых TEX'у; эта задача более сложна и будет рассмотрена в разд. 10.5.6.

10.1.1 Система PSNFSS

Рассмотрим сначала наиболее простой случай — использование заранее подготовленной системы. Стандартный LATEX'овский пакет PSNFSS, координируемый Себастианом Ратцем, обеспечивает прямую поддержку широко распространенных шрифтов в формате PostScript. Он состоит из набора простых пакетов, переопределяющих на высоком уровне установки по умолчанию для семейств шрифтов, и различных более сложных пакетов, таких, как пакет для использования шрифта Lucida Bright совместно с дополняющим его шрифтом Lucida New Math, и пакет для использования шрифта Times в математике. В табл. 10.1 перечислены эти пакеты и указано производимое ими действие.

Большинство пользователей хотели бы просто иметь поддержку шрифтов, встроенных почти во все PostScript'овские принтеры. Это — четыре семейства шрифтов с засечками (Times, Palatino, Bookman и New Century Schoolbook), два семейства рубленых шрифтов (Helvetica и AvantGarde), один моноширинный шрифт, имитирующий шрифт пишущей машинки (Courier), символьный шрифт (Symbol) и рукописный Zapf Chancery. Этими шрифтами фирма Apple снабдила свой первый лазерный принтер LaserWriter Plus в 1986 г. Вместе эти семейства шрифтов (с учетом различных начертаний) включают в себя 35 шрифтов, и вы часто встретите ссылки на эти «35 шрифтов» (в табл. 10.2 вы можете увидеть пример текста, набранного этими шрифтами). Если вас удовлетворяют эти шрифты, то можете быть уверены, что любой пользователь TEX'а сможет распечатать подготовленные вами документы.

Чтобы установить поддержку этих шрифтов, достаньте пакет PSNFSS, который можно найти в одном из архивов CTAN, и обработайте LATEX'ом файл `psfonts.ins`. При этом создадутся пакеты (times, palatino, helvet, avant, newcent, bookman, chancery, pifont и mathptm) для стандартных шрифтов, которые вы собираетесь установить в вашей системе TEX. После этого вам нужно добавить метрические файлы для TEX'а, файлы виртуальных шрифтов и файлы описания шрифтов, и информировать ваш `dvi`-драйвер об именах шрифтов. Все файлы, которые необходимо установить, можно взять из архивов CTAN в директории `fonts/psfonts/adobe`, однако удобнее воспользоваться дистрибутивом PSNFSS, содержащим готовые к установке файлы, которые собраны в файл `lw35nfss.zip`, находящийся в директории `macros/latex/required/psnfss`. Если вы используете драйвер dvips, вам будет достаточно добавить содержи-

Таблица 10.1. Действия, производимые файлами пакета PSNFSS.

Пакет	*Шрифт без засечек*	*Шрифт с засечками*	*Шрифт пишущей машинки*
times	Helvetica	Times	Courier
palatino	Helvetica	Palatino	Courier
helvet	Helvetica		
avant	AvantGarde		
newcent	AvantGarde	NewCenturySchoolbook	Courier
bookman	AvantGarde	Bookman	Courier
chancery		Zapf Chancery	
mathptm		Times	
basker[1]		Monotype Baskerville	
bembo[1]		Bembo	
charter		Bitstream Charter	
garamond[1]		Adobe Garamond	
mtimes[1]		Monotype Times	
nimbus[2]	URW NimbusSans-Regular	URW NimbusRoman-Regular	
utopia		Utopia	
lucbr[3]	LucidaSans	LucidaBright	LucidaTypewriter

Пакет mathptm обеспечивает максимально возможную поддержку математики с использованием PostScript'овского шрифта Times и других стандартных шрифтов. Пакет lucbr использует для шрифта Lucida Bright имена в соответствии со схемой, описанной в *Fontname*. В этом пакете есть опция `yy` для использования имен шрифтов, поддерживаемых Y&Y, которая загружает шрифт Lucida New Math; опция `expert` позволяет использовать дополнительные прямые математические символы и шрифты начертания капитель (small-caps).

[1] Эти пакеты входят в состав пакета psnfss-extra, который распространяется вместе с MikTEX. — *Прим. перев.*

[2] Этот пакет вместе со шрифтами распространяется отдельно от PSNFSS. — *Прим. перев.*

[3] Этот пакет входит в состав шрифтовых пакетов для Textures. — *Прим. перев.*

мое файла `psnfss.map`, находящегося в поддиректории `dvips` этого архива, к установленному в системе файлу `psfonts.map`.

Небольшие пакеты PSNFSS работают путем переопределения принятых по умолчанию семейств шрифтов с засечками, без засечек (рубленого шрифта) и шрифта пишущей машинки. Например, пакет times состоит из трех команд:

```
\renewcommand{\sfdefault}{phv}
\renewcommand{\rmdefault}{ptm}
\renewcommand{\ttdefault}{pcr}
```

Единственной сложностью при написании таких пакетов оказывается определение имени семейства шрифтов; мы обсудим этот вопрос в разд. 10.4.

Таблица 10.2. Образцы текста, набранные стандартными PostScript-шрифтами.

Times Roman For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

Palatino For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

New Century Schoolbook For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

AvantGarde For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

Bookman For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

Helvetica For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

Courier `For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345-67890 JUMPS! -- ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?`

Zapf Chancery *For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?*

10-1-1

Документы, содержащие математические формулы

У многих пользователей LaTeX'а, которые начинают применять такие шрифты, как Times, создается неприятное впечатление при виде страницы, на которой одновременно представлены текст и математические формулы. Математические шрифты Computer Modern плохо сочетаются с Times по толщине линий, средней высоте букв и общему дизайну. На рис. 10.1 показан пример такого сочетания, который далек от совершенства. К сожалению, выбор математических шрифтов невелик, а сами шрифты сильно отличаются друг от друга; пользователям TeX'а сейчас приходится выбирать между шрифтами Computer Modern, Conctere Math, Euler Math, MathTime и Lucida New Math. При этом возникают следующие трудности:

1. для научных публикаций основным шрифтом является Times, с которым ни Computer Modern, ни Euler Math, ни Lucida хорошо не сочетаются;
2. Lucida и MathTime являются коммерческими шрифтами с вытекающими из этого лицензионными ограничениями;
3. в создаваемый PostScript'овский файл приходится включать большое число шрифтов, хотя частичная загрузка шрифта помогает уменьшить размер файла.

Для решения этих проблем Алан Джеффри создал пакет mathptm, в котором для замены максимального числа букв и символов в математических шрифтах Computer Modern символами из стандартных шрифтов, имеющимися практически в каждом PostScript'овском принтере (Adobe Times, Symbol и Zapf Chancery для рукописного) используется технология виртуальных шрифтов (см. разд. 10.2.3). Результат этой замены показан на рис. 10.2. Если исключить некоторую неэлегантность, самым большим недостатком остается отсутствие полужирного начертания для символов, что может ограничивать применимость пакета mathptm в некоторых ситуациях. Если отсутствие этих символов вас не смущает, то использование этого пакета является хорошим решением при подготовке документов на основе шрифтов Times с тем, чтобы не сталкиваться с проблемами при распространении документов. Однако если вам нужна высококачественная печать, подумайте, не стоит ли вместо этого использовать шрифты MathTime.

Пример создания пакета, подобного mathptm, в котором шрифт Minion для текста и шрифт Lucida для математики объединяются с использованием технологии виртуальных шрифтов в рамках пакета Алана Джеффри fontinst, будет рассмотрен на с. 466.

Поддержка шрифта Symbol

Пакет pifont обеспечивает доступ к стандартным шрифтам специальных символов — так называемым Pi-шрифтам, таким, как Zapf Dingbats и Post-

The default math mode font is *math italic*. This should not be confused with ordinary *text italic*. `\mathbf` produces **bold – face roman** letters. Greek is available in upper and lower case: $\alpha, \beta, \Gamma, \Delta \ldots \omega, \Omega$ There is also a calligraphic font for uppercase letters; these are produced by the `\mathcal` command: $\mathcal{ABCDE}$

$$\phi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx \tag{1}$$

$$\prod_{j\geq 0} \left(\sum_{k\geq 0} a_{jk} z^k \right) = \sum_{k\geq 0} z^n \left(\sum_{\substack{k_0, k_1, \ldots \geq 0 \\ k_0 + k_1 + \ldots = n}} a_0 k_0 a_{1k_1} \ldots \right) \tag{2}$$

$$\pi(n) = \sum_{m=2}^{n} \left\lfloor \left(\sum_{k=1}^{m-1} \lfloor (m/k) / \lceil m/k \rceil \rfloor \right)^{-1} \right\rfloor \tag{3}$$

$$\{\overbrace{a, \ldots, a}^{k\ a's}, \overbrace{b, \ldots, b}^{l\ b's}\} \tag{4}$$
$$\underbrace{\qquad\qquad}_{k+1 \text{ elements}}$$

10-1-2

Рис. 10.1. Сочетание текста, набранного шрифтом Times, с математическими формулами, набранными шрифтами Computer Modern.

Script'овский шрифт Symbol, а также позволяет работать и с другими семействами символьных шрифтов. В этом пакете определены три новые команды и два окружения.

`\Pisymbol{`*family*`}{`*number*`}`

Команда `\Pisymbol` вставляет символ из Pi-шрифта. Команда имеет два параметра: имя семейства шрифтов и номер символа, который можно задать в десятичной, восьмеричной или шестнадцатеричной системе счисления с помощью стандартных TEX'овских процедур. Так, `\Pisymbol{pzd}{'166}` генерирует символ ❖.

`\Pifill{`*family*`}{`*number*`}`

Команда `\Pifill` (с теми же параметрами) действует аналогично команде `\fill` в TEX'е, но заполняет пространство указанным символом ➠ ➠ ➠ (как в этой строке).

The default math mode font is *math italic*. This should not be confused with ordinary *text italic*. `\mathbf` produces **bold – face roman** letters. Greek is available in upper and lower case: $\alpha, \beta, \Gamma, \Delta \ldots \omega, \Omega$ There is also a calligraphic font for upper-case letters; these are produced by the `\mathcal` command: $\mathcal{ABCDE}$

$$\phi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx \tag{1}$$

$$\prod_{j\geq 0} \left(\sum_{k\geq 0} a_{jk} z^k \right) = \sum_{k\geq 0} z^n \left(\sum_{\substack{k_0, k_1, \ldots \geq 0 \\ k_0 + k_1 + \ldots = n}} a_0 k_0 a_{1k_1} \ldots \right) \tag{2}$$

$$\pi(n) = \sum_{m=2}^{n} \left\lfloor \left(\sum_{k=1}^{m-1} \lfloor (m/k) / \lceil m/k \rceil \rfloor \right)^{-1} \right\rfloor \tag{3}$$

$$\{\overbrace{a, \ldots, a}^{k\ a's}, \overbrace{b, \ldots, b}^{l\ b's}\} \tag{4}$$

$$\underbrace{\qquad\qquad}_{k+1 \text{ elements}}$$

10-1-3

Рис. 10.2. Сочетание текста, набранного шрифтом Times, с математическими формулами, в которых частичная замена на шрифт Times осуществлена пакетом mathptm.

`\Piline{`*family*`}{`*number*`}`

Команда `\Piline` порождает отдельную строку символов с небольшими отступами справа и слева:

✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂

`\begin{Pilist}{`*family*`}{`*number*`}`

Окружение `Pilist` порождает ненумерованный список, в котором вместо жирной точки (bullet) используется указанный Pi-символ:

- ♣ Не забывайте о шрифте символов,
- ♣ с помощью которого можно получить

  ♠ ♠ ♠ ♠

- ♣ и отдельные символы, такие, как ⊗.

```
\begin{Pilist}{psy}{'247}
\item Не забывайте о шрифте символов,
\item с помощью которого можно получит
      \Piline{psy}{"AA}
\item и отдельные символы, такие, как
      \Pisymbol{psy}{"C4}.
\end{Pilist}
```

10-1-4

`\begin{Piautolist}{`*family*`}{`*number*`}`

Окружение `Piautolist` несколько сложнее. При заданном начальном значении счетчика *number* оно помечает строку с помощью соответствующего Pi-символа и затем увеличивает значение счетчика при переходе к новой записи:

① текст внутри списка,

α текст внутри списка,

β текст внутри списка,

i. текст внутри списка,

✎ текст внутри списка,

✏ текст внутри списка,

ii. текст внутри списка,

χ текст внутри списка,

② текст внутри списка.

Мы можем ссылаться на строку, помеченную ✎, обычным способом.

```
\begin{Piautolist}{pzd}{'254}
\item  текст внутри списка,
 \begin{Piautolist}{psy}{'141}
 \item  текст внутри списка,
 \item  текст внутри списка,
  \begin{enumerate}
  \item  текст внутри списка,
   \begin{Piautolist}{pzd}{'56}
    \item  текст внутри списка,%
          \label{pilabel}
    \item  текст внутри списка,
   \end{Piautolist}
   \item  текст внутри списка,
  \end{enumerate}
 \item  текст внутри списка,
 \end{Piautolist}
\item  текст внутри списка.
\end{Piautolist}
Мы можем ссылаться на строку,
помеченную \ref{pilabel}, обычным
способом.
```

10-1-5

Этот интерфейс может быть использован и с другими семействами шрифтов, отличными от `pzd` и `psy`. Все, что для этого нужно, — это шрифт в кодировке `U` насыщенностью `m` (medium) с прямым начертанием (normal). Это дает простой способ добавления математических символов (например, греческих букв) в обычный текст. Для этого нам потребуется иметь либо файл описания шрифта (файл `.fd`), либо файл пакета с командами, описывающими этот шрифт. Например, для использования одного из шрифтов Lucida New Math нам нужно определить семейство и начертание шрифта:

```
\DeclareFontFamily{U}{hlcm}{}
\DeclareFontShape{U}{hlcm}{m}{n}{<->hlcrm}{}
```

После этого мы можем обращаться к шрифту с помощью команды `\Pisymbol`:

```
\newcommand\textGamma{\Pisymbol{hlcm}{0}}
```

Отметим, что сам шрифт (`hlcrm`) загружается один раз, независимо от того, вызывается ли он только здесь или с помощью обычного математического пакета; все, что мы здесь сделали, так это установили еще один внутренний интерфейс к LaTeX'овскому NFSS.

10.1.2 Шрифты, метрические файлы: взгляд в целом

Наиболее общим источником недоразумений для пользователя, который хотел бы начать работать с шрифтами PostScript, является непонимание того, какие файлы ему следует установить и что при этом надо изменить в настройке драйвера. В этом разделе мы опишем подробнее, какие файлы используются в пакете PSNFSS. Более мелкие детали, которые могут меняться от драйвера к драйверу, в этой главе мы рассматривать не будем. В гл. 11 мы кратко обсудим ряд драйверов, но в целом отсылаем читателя к документации.

Пакеты PSNFSS используют в своей работе *виртуальные шрифты* (см. разд. 10.2.3). Это означает, что для каждого используемого шрифта в системе имеются три файла и добавлена одна строка в файл описания шрифта. Рассмотрим в качестве примера полужирный шрифт Times. В PSNFSS стилевой файл пакета times.sty выглядит примерно так:

```
\def\fileversion{5.2}
\def\filedate{1995/08/16}
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{times}[\filedate\space\fileversion\space
 Times PSNFSS2e package]
\renewcommand{\sfdefault}{phv}
\renewcommand{\rmdefault}{ptm}
\renewcommand{\ttdefault}{pcr}
```

Это означает следующее. Когда LaTeX'у надо набрать что-нибудь шрифтом с засечками, он ищет *файл кодировки (font-encoding file)*, имя которого составлено из названия кодировки и имени семейства шрифта, определенного как \rmdefault. Поскольку кодировкой по умолчанию является OT1, а в рассматриваемом пакете в качестве \rmdefault определено семейство ptm, то LaTeX открывает файл ot1ptm.fd[1], который выглядит примерно так:

```
\ProvidesFile{ot1ptm.fd}
   [1997/01/01 Fontinst v1.504 font definitions for OT1/ptm.]
\DeclareFontFamily{OT1}{ptm}{}
\DeclareFontShape{OT1}{ptm}{b}{n}{<-> ptmb7t}{}
\DeclareFontShape{OT1}{ptm}{b}{sc}{<-> ptmbc7t}{}
\DeclareFontShape{OT1}{ptm}{b}{sl}{<-> ptmbo7t}{}
\DeclareFontShape{OT1}{ptm}{b}{it}{<-> ptmbi7t}{}
\DeclareFontShape{OT1}{ptm}{m}{n}{<-> ptmr7t}{}
\DeclareFontShape{OT1}{ptm}{m}{sc}{<-> ptmrc7t}{}
\DeclareFontShape{OT1}{ptm}{m}{sl}{<-> ptmro7t}{}
\DeclareFontShape{OT1}{ptm}{m}{it}{<-> ptmri7t}{}
\DeclareFontShape{OT1}{ptm}{bx}{n}{<->ssub * ptm/b/n}{}
\DeclareFontShape{OT1}{ptm}{bx}{sc}{<->ssub * ptm/b/sc}{}
\DeclareFontShape{OT1}{ptm}{bx}{sl}{<->ssub * ptm/b/sl}{}
\DeclareFontShape{OT1}{ptm}{bx}{it}{<->ssub * ptm/b/it}{}
```

[1] Заметим, что имя кодировки переводится в нижний регистр перед тем, как начинается поиск файла.

В этом файле для каждой допустимой комбинации кодировки, семейства, насыщенности и начертания указано соответствующее имя шрифта. Если мы набираем текст полужирным шрифтом с помощью команды `\textbf`, то насыщенностью, выбираемой по умолчанию, является `bx`. Если при этом используется прямое начертание, то LaTeX ищет в приведенном файле строку, отвечающую кодировке `OT1`, семейству `ptm`, насыщенности `bx` и начертанию `n`. Полученное в результате поиска выражение `<->ssub * ptm/b/n` означает, что обычный полужирный шрифт без генерации предупреждений замещается на шрифт семейства `ptm` насыщенности `b` и начертания `n`. Повторный поиск в файле комбинации `OT1 + ptm + b + n` дает нам `<-> ptmb7t`. Таким образом, имя искомого шрифта — `ptmb7t`, а значок `<->` говорит, что для всех размеров шрифта будет использоваться один и тот же шрифтовой файл. Затем ядро TeX'а загружает файл `ptmb7t.tfm` (если, конечно, находит его) и работа продолжается.

После того как работа TeX'а завершена, мы обращаемся к `dvi`-файлу, в котором содержится запрос к шрифту `ptmb7t`. Драйвер устройства должен попытаться удовлетворить этот запрос, что может быть сделано одним из четырех способов:

1. загрузив *виртуальный шрифт* (с указанным именем), в котором содержатся дальнейшие инструкции;
2. распознав, что шрифт встроен в принтер, так что его более не нужно искать;
3. распознав, что шрифт может быть обработан прямо в принтере, но его надо туда предварительно загрузить;
4. найдя растровый файл шрифта (`pk`-файл), как это делалось бы в случае METAFONT'овских шрифтов Computer Modern.

Большинство драйверов следует этой последовательности попыток, хотя некоторые драйверы не поддерживают первый шаг (поиск виртуальных шрифтов) или не рассматривают второй и третий шаги (если они рассчитаны на простейшие принтеры, в которых нет встроенных шрифтов). Если ваш драйвер не поддерживает виртуальных шрифтов, вы можете предварительно обработать `dvi`-файл стандартной TeX'овской утилитой `dvicopy`, которая «упрощает» `dvi`-файлы, заменяя в них ссылки на виртуальные шрифты соответствующими инструкциями.

Виртуальный шрифт `ptmb7t` состоит из набора инструкций для каждого символа; большинство из этих инструкций просто дает указание драйверу взять соответствующий символ из реального шрифта, `ptmb8r`. В этом случае драйвер запускается вновь, пытаясь удовлетворить запросу к `ptmb8r`. Теоретически, в виртуальном шрифте можно указать ссылку на другой виртуальный шрифт и построить их целую цепочку, но практически требуется, чтобы драйвер нашел имя реального шрифта в своем списке зарегистрированных шрифтов. Как это делается, зависит от драйвера. В разд. 11.2.4 мы рассмотрим программу Тома Рокицки `dvips`, в которой для регистрации шрифтов исполь-

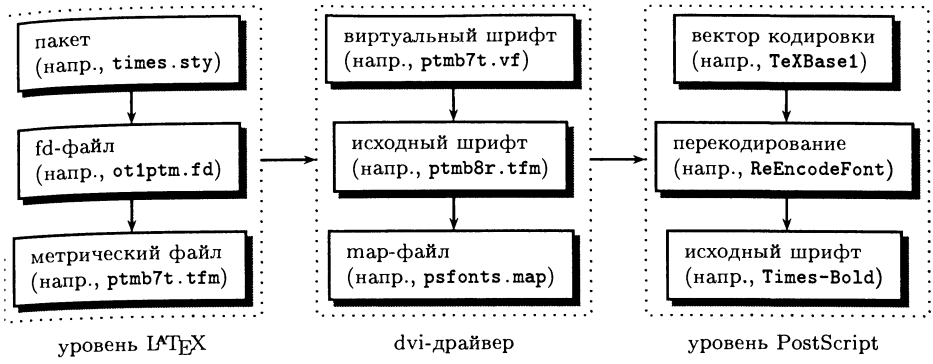


Рис. 10.3. Файлы и процессы, используемые LaTeX'ом, dvi-драйвером и интерпретатором PostScript'а.

зуется map-файлы; в других драйверах используются аналогичные подходы. Хотя эти map-файлы не могут прямо использоваться драйверами, отличными от dvips, они содержат всю информацию, необходимую для подключения шрифта. Например, строка

```
ptmb8r Times-Bold "TeXBase1Encoding ReEncodeFont" <8r.enc
```

означает, что TeX'овское имя шрифта ptmb8r соответствует PostScript'овскому шрифту Times-Bold после перекодировки его в кодировку TeXBase1Encoding. Эта кодировка хранится в файле 8r.enc, который также должен быть загружен в принтер. Подробнее об интерпретации map-файлов для dvips см. разд. 11.2.4.

Подытоживая сказанное (см. рис. 10.3), пользователь пакета PSNFSS должен найти и установить следующие ресурсы для каждого из семейств шрифтов, которые он намерен использовать:

1. файлы описания шрифта fd для каждой из поддерживаемых кодировок указанного семейства шрифтов (например, ot1ptm.fd и t1ptm.fd);
2. файлы метрики шрифтов tfm для каждого шрифта в семействе и каждой из кодировок (например, ptmr7t.tfm, ptmr8t.tfm, ptmb7t.tfm, ptmb8t.tfm и т. д.);
3. файлы виртуальных шрифтов vf для каждого из метрических файлов (например, ptmr7t.vf, ptmr8t.vf, ptmb7t.vf, ptmb8t.vf и т. д.);
4. файлы метрики tfm для каждого из имен исходных шрифтов, ссылки на которые могут возникать при обработке виртуальных шрифтов (например, ptmr8r.tfm, ptmb8r.tfm и т. д.). Заметим, что ptmr7t и ptmr8t разрешаются с помощью ссылок в один общий tfm-файл — ptmr8r;
5. строки в map-файле драйвера, определяющие соответствие между именами TeX'овских шрифтов (например, ptmr8r) и PostScript'овских шрифтов и возможную перекодировку этих шрифтов;

6. файлы векторов кодировки PostScript'овских шрифтов, на которые имеются ссылки в map-файле (например, `8r.enc`);
7. и (возможно) сами шрифты, которые нужно загружать.

«А почему бы», спросит дотошный читатель, «нам просто не использовать имя Times-Roman в TEX'е, сделать все перекодировки либо с помощью такого средства, как LATEX'овский пакет inputenc, либо непосредственно в PostScript'е, и таким образом избегнуть всей этой возни с виртуальными шрифтами, map-файлами и т. п.?» Так на самом деле и работало первое поколение PostScript'овских драйверов TEX'а, и в простых случаях это продолжает хорошо и быстро работать и сейчас. Причин, по которым PSNFSS приняла более сложную схему, две. Во-первых, механизм виртуальных шрифтов обеспечивает существенно большую гибкость по сравнению с простой перекодировкой и обеспечивает дополнительный прозрачный функциональный слой, лежащий между TEX'ом и PostScript'ом. Хотя в ряде случаев желаемого результата можно было бы добиться прямой манипуляцией с самими PostScript'овскими шрифтами, это не всегда возможно или желательно в многопользовательской среде. Во-вторых, поскольку имена шрифтов не подчиняются какой-либо схеме, принятие рациональной схемы именования шрифтов в TEX'е существенно помогает при работе с большим числом шрифтов.

10.1.3 Установка PostScript-шрифтов

Для многих широко распространенных PostScript-шрифтов, разработанных фирмами Adobe, Monotype и др., в архивах CTAN имеются наборы метрических файлов и файлов описания шрифтов, готовые для непосредственного применения. Для решения задач перекодировки в них используются виртуальные шрифты. Чтобы установить новое семейство шрифтов, вам нужно сделать следующее:

1. Найти имя семейства шрифтов в директории `fonts/psfonts` архива CTAN. Эта директория организована по именам разработчиков шрифтов (например, `adobe`, `monotype`). Так, файлы поддержки шрифта Adobe Garamond лежат в `adobe/garamond`. Имена директорий выбраны в соответствии со схемой именования шрифтов (*Fontname*), которая описана в разд. 10.4 на с. 443.
2. В каждой из этих директорий есть поддиректории, в которых хранятся `tfm`, `vf` и `fd`-файлы; для Adobe Garamond вам понадобятся все файлы из этих поддиректорий, например `tfm/pgmb7t.tfm` и т. д. Эти файлы следует установить так, чтобы LATEX и ваш драйвер могли найти их; LATEX использует файлы `tfm` и `fd`, а драйвер — файлы `vf` (и, возможно, также файлы `tfm`).
3. Вам следует указать вашему драйверу, что вы собираетесь использовать эти новые PostScript-шрифты (чтобы драйвер не искал исходные файлы METAFONT и файлы pk). Для dvips это делается так: из директории се-

мейства шрифтов берется map-файл и его содержимое добавляется в конец стандартного файла psfonts.map драйвера dvips. То же можно сделать и иначе: установить map- и config-файлы[1] там, где хранятся рабочие файлы dvips, и использовать ключ для загрузки дополнительного map-файла на лету (в нашем примере dvips -Ppag).
Если вы пользуетесь другим драйвером (не dvips), посмотрите в описании драйвера, как информировать его о появлении новых PostScript-шрифтов.

4. Если шрифт не встроен в принтер, вам надо загрузить его. Вы можете сделать это с помощью программ, поставляемых со шрифтом, или это умеют делать на лету некоторые драйверы (в разд. 11.2.4 мы рассмотрим, как это делает dvips). Следует, однако, заметить, что имена в map-файлах на CTAN предполагают строгое соответствие со схемой *Fontname*, так что вам, возможно, понадобится переименовать имеющиеся у вас pfb или pfa-файлы.
5. Если вы захотите работать только с одним семейством шрифтов и вам нужно заменить этими шрифтами используемые по умолчанию шрифты с засечками, рубленый шрифт и шрифт пишущей машинки в вашем документе, то на этот случай в каждой из директорий шрифтов на CTAN есть простой стилевой файл LATEX, осуществляющий эту замену.

После того как метрики шрифтов скопированы и установлены, а ваш драйвер настроен, вы можете просто указать соответствующий стилевой файл и начать работать. Заметим, однако, что изменение семейства текстовых шрифтов не влияет на математику. Установка новых математических шрифтов является более сложной задачей и мы вернемся к ней в примере, рассмотренном на с. 466.

Того, что вы уже прочли, достаточно для решения такой задачи, как замена шрифта, используемого по умолчанию в вашем документе, на Times. Однако если вы готовите серьезную книгу или журнал, вам понадобится более глубокое понимание технологии шрифтов как со стороны TEX'а, так и со стороны PostScript'а. Поэтому мы немного отклонимся в сторону и рассмотрим основы этой технологии.

10.2 Технология шрифтов TEX'а

TEX использует довольно общее представление о шрифте, поэтому все, что ему нужно — это набор метрической информации для каждого из шрифтов, а реальное включение шрифтов в документ осуществляет dvi-драйвер. По этой причине рассуждения о TEX'овских шрифтах в действительности не имеют большого смысла, поскольку, как уже отмечалось во введении, любой шрифт

[1] Имя config-файла имеет общий вид config.*сокращенноеимясемейства*. Например, для Adobe Garamond сокращенное имя семейства в соответствии с *Fontname* есть pag, так что имя config-файла — config.pag.

может быть использован в TEX'е, если мы можем создать для него метрическую информацию (файл `tfm`, см. разд. 10.2.2), а программа драйвера может использовать шрифт, на который стоит ссылка в `dvi`-файле. Ниже мы опишем стандартную процедуру вызова шрифтов, используемую TEX'овскими шрифтами METAFONT, поскольку тот же подход верен и для других шрифтов. После того как мы опишем форматы TEX'овских шрифтов, мы рассмотрим формат метрических файлов для TEX'а и затем обсудим концепцию виртуальных шрифтов — то, что нам нужно для успешного подключения PostScript-шрифтов.

10.2.1 Форматы шрифтов TEX'а

Формат `gf`

Общий формат `gf` (generic font) получается при генерации шрифта программой METAFONT. Этот формат — разновидность растрового (*bitmap*) формата, который отличается от векторного формата исходного шрифта METAFONT. Шрифт в формате `gf` создается для конкретного устройства и имеет конкретный размер (например, 16 пунктов для лазерного принтера Canon). Формат `gf` не используется другими программами и был разработан для упрощения преобразования шрифтов в другие форматы, откуда и происходит его название «общий». Одновременно с генерацией `gf`-файла METAFONT порождает `tfm`-файл (метрический файл TEX'а), который будет использоваться вместе с этим шрифтом; мы рассмотрим этот тип файлов в разд. 10.2.2.

Система TEX обычно включает несколько программ преобразования из формата `gf`. Среди них наибольшее значение имеет `gftopk`, которая создает файлы `pk`, формат которых понимается всеми современными `dvi`-драйверами. Другими программами являются `gftodvi`, которая создает большие рисунки каждого глифа (например, те, которые приведены в Knuth (1986e)) с целью тестирования METAFONT-шрифтов при их разработке, и программа `gftype`, которая проверяет целостность `gf`-файла.

Формат `pxl`

Формат `pxl` представляет собой устаревший формат шрифта, который использовался первыми `dvi`-драйверами. Если в вашей системе еще используются какие-нибудь `pxl`-шрифты, их надо рассматривать как антиквариат и следует серьезно задуматься, не провести ли обновление системы. В принципе, существуют такие программы преобразования, как `gftopxl` и `pxltopk`, но их может оказаться трудно найти, поскольку современные дистрибутивы TEX'а редко поддерживают формат `pxl`.

Формат `pk`

Стандартным растровым форматом, используемым драйверами, является формат `pk`. Он представляет собой просто более эффективную форму хранения

шрифтов gf и генерируется программой gftopk, которая запускается сразу после METAFONT'а. На этой стадии уже нельзя изменить такие характеристики шрифта, как размер и тип устройства. Среди стандартных программ следует отметить программу pktogf для перевода pk-файла обратно в общий формат и программу pktype для проверки целостности pk-шрифта.

Пользователю следует помнить, что растровые шрифты нельзя масштабировать без потери качества и что METAFONT создает разные растровые шрифты для разных устройств. Очень тонкой настройкой, которая возможна при использовании METAFONT'а, можно создать высококачественные изображения глифов, но при этом появляется недостаток: шрифты нельзя использовать на другом выходном устройстве (хотя на практике многие устройства, например лазерные принтеры с разрешением 300 точек на дюйм, достаточно похожи и могут работать с одними и теми же файлами).

10.2.2 Метрические файлы TEX'а

Метрическая информация о глифах, которая необходима TEX'у, хранится в файлах с расширением tfm (что значит TEX font metric). Для каждого глифа в шрифте в этом файле записаны данные о четырех размерах: ширине, высоте выше базовой линии, глубине ниже базовой линии и так называемой «поправке на курсив», т. е. величине дополнительного сдвига, который добавляется командой \/ при переключении от наклонного к прямому начертанию.[1] Знания TEX'а о глифе ограничиваются только этими данными. По этой причине часто говорят, что TEX рассматривает глифы как «черные ящики».

Кроме этого, в tfm-файлах содержатся данные о некоторых соотношениях между символами — кернинге и лигатурах. Кернинг между двумя символами означает, что когда эти два символа оказываются рядом, они немного сдвигаются и между ними вставляется пустое пространство — «керн» (причем керн может быть и отрицательным), чтобы сделать внешний вид этой пары букв более гармоничным. Лигатура — это глиф, который TEX вставляет на место двух или большего числа букв, появляющихся в документе. Например, во многих шрифтах присутствует глиф «fi» (как единая литера), который используется всякий раз когда буква «i» появляется вслед за буквой «f». Издательские системы, которые не поддерживают лигатур, вместо этого печатают «fi».[2]

На самом деле механизм лигатур, используемый в TEX'е, намного мощнее, чем может показаться из рассмотренного примера. Можно, например, указать, что начертание буквы должно изменяться, если она стоит в начале или конце слова; можно вставить несколько глифов; символы, участвующие в формировании лигатуры, можно либо замещать (как в примере выше), либо частично или полностью сохранять. Так, правило формирования лигатуры в

[1] Пользователи LATEX'а обычно не должны беспокоиться о добавлении команд \/ в соответствующих местах, поскольку это автоматически берут на себя команды высокого уровня, такие, как \textit и \emph.

[2] В русских текстах для набора кавычек-елочек используются конструкции << и >>, которые дают лигатуры « и ». — *Прим. перев.*

некотором шрифте можно сформулировать так: если за буквой «a» следует «b», то заменить «a» на «c» и сохранить «b».

Наконец, файл `tfm` содержит информацию о шрифте в целом: кегель, ширину обычного пробела, его растяжимость и т. п. Из TEX'а эту информацию можно получить с помощью команды `\fontdimen` (Goossens et al., 1994, с. 234). Математические шрифты содержат еще несколько дополнительных размеров для специальных установок.

Информация в `tfm`-файле хранится в двоичном формате, удобном для быстрого доступа из TEX'а, однако поэтому ее практически невозможно ни прочесть, ни модифицировать без помощи компьютера. С помощью программы `tftopl` файл `tfm` можно преобразовать в `pl`-файл (property list), который имеет текстовый формат и при необходимости может быть отредактирован. Для преобразования `pl`-файла в `tfm`-файл служит программа `pltotf`.

Файл `pl` состоит из набора записей (строк), имеющих вид:

```
(СВОЙСТВО ЗНАЧЕНИЕ),
```

где *свойство* — одно из конечного множества имен, а *значение*, в зависимости от типа свойства, — строка, число или даже другой набор записей. Чтобы объяснить эту структуру лучше, рассмотрим в качестве примера фрагмент файла `cmr10.pl`; полное описание синтаксиса файла `pl` можно найти в Web-версии документации к `tftopl`.

Файл начинается со свойств, описывающих шрифт в целом (свойство с именем `COMMENT` игнорируется). Числа могут задаваться действительными (`R`), десятичными (`D`) или восьмеричными (`O`) числами.

```
(FAMILY CMR)
(FACE O 352)
(CODINGSCHEME TEX TEXT)
(DESIGNSIZE R 10.0)
(COMMENT DESIGNSIZE IS IN POINTS)
(COMMENT OTHER SIZES ARE MULTIPLES OF DESIGNSIZE)
(CHECKSUM O 11374260171)
(FONTDIMEN
   (SLANT R 0.0)
   (SPACE R 0.333334)
   (STRETCH R 0.166667)
   (SHRINK R 0.111112)
   (XHEIGHT R 0.430555)
   (QUAD R 1.000003)
   (EXTRASPACE R 0.111112)
   )
```

Свойство `FONTDIMEN` само имеет список свойств, значениями которых являются размеры шрифта. Поскольку они нужны TEX'у для верстки, программа `tftopl` (с помощью которой был получен этот листинг) знает о них и идентифицировала их по именам. Так, свойство `SPACE` указывает нормальный размер пробела, который следует использовать, когда выбран данный шрифт. Если в шрифте

указаны дополнительные размеры, то соответствующие им дополнительные записи (в примере ниже — для `\fontdimen8`) могут выглядеть так:

```
(PARAMETER D 8 R 0.1)
```

Следующий раздел файла `cmr10.pl` представляет собой свойство `LIGTABLE`, значением которого является набор записей, содержащих информацию о лигатурах и кернинге. Мы приведем здесь лишь небольшой фрагмент этой таблицы:

```
(LIGTABLE
   ...
   (LABEL C f)
   (LIG C i O 14)
   (LIG C f O 13)
   (LIG C l O 15)
   (KRN O 47 R 0.077779)
   (KRN O 77 R 0.077779)
   (KRN O 41 R 0.077779)
   (KRN O 51 R 0.077779)
   (KRN O 135 R 0.077779)
   (STOP)
   (LABEL O 13)
   (LIG C i O 16)
   ...
   (STOP)
   )
```

Эти записи следует интерпретировать так: если мы встретили букву «f» (`C f` обозначает букву, находящуюся в ASCII-таблице в позиции «f») и следующей за ней буквой является «i», заменить обе буквы глифом, находящимся в восьмеричной позиции 14 (в рассматриваемом шрифте там расположена лигатура «fi»). Аналогично, если за «f» следует другая буква «f», заменяем их на глиф с восьмеричным кодом 13 и т. д. для остальных строк `LIG`. Свойство `KRN`, указывает, какой ширины пробел должен быть добавлен (0.077779 × `DESIGNSIZE`), когда за «f» следуют определенные символы (указанные в примере восьмеричные коды отвечают знакам ', ?, !,) и]). Строка `STOP` завершает текущее правило лигатуры или кернинга, а `LABEL` начинает новое. Свойство `LIG` для глифа в восьмеричной позиции 13 ответственно за генерацию трехбуквенной лигатуры «ffi»: с помощью первого правила буквы «f» и «f» преобразуются в лигатуру «ff» (восьмеричное 13), а затем, если после нее следует буква «i», она дальше преобразуется в «ffi» (восьмеричное 16).

В действительности свойство `LIG` имеет ряд вариантных форм: например, непосредственно перед ним или сразу после него может располагаться наклонная черта, после которой следует некоторое число знаков > (не превышающее числа наклонных черт). Знаки черты обозначают удержание исходных символов слева или справа, а знаки > обозначают, что удержанный (или

вставленный) символ не должен участвовать в дальнейшей обработке лигатур. Например,

```
(LABEL C a)   (LIG/> C b C c)   (STOP)
(LABEL C c)   (KRN C b R 1.0)   (STOP)
```

преобразует «ab» в «cb», но второе правило игнорируется и между буквами «b» и «c» не вставляется никакого керна, поскольку знак > указывает, что обработка лигатур должна продолжаться начиная с буквы «b», а не со вставленной буквы «c» (разумеется, керн появится, если во входном файле встретится комбинация «cb»).

Возвратимся к нашему примеру. Файл pl завершается большим числом записей со свойством CHARACTER (по одному для каждого символа в шрифте). Мы приведем здесь записи, относящиеся к букве «g»:

```
(CHARACTER C g
   (CHARWD R 0.500002)
   (CHARHT R 0.430555)
   (CHARDP R 0.194445)
   (CHARIC R 0.013888)
   (COMMENT
      (KRN C j R 0.027779)
      )
   )
```

Первые четыре записи указывают значения ширины, высоты, глубины и курсивной коррекции для данного символа; если значение какого-то из свойств равно нулю, то запись может быть опущена. За ними следует свойство COMMENT, в котором перечисляется любая информация о кернинге или лигатурах для данной буквы. Эти строки автоматически генерируются программой tftopl из параметров раздела LIGTABLE просто для информации. Они говорят нам, что где-то в секции LIGTABLE есть правило добавления керна между литерами «g» и «j».

10.2.3 Виртуальные шрифты TEX'а

Виртуальные шрифты (Knuth, 1990) так же, как и метрические файлы, содержат общую информацию о шрифте и подробные инструкции для каждого символа. Однако эти инструкции остаются как бы на «высоком уровне»: виртуальные шрифты описывают «воображаемый» шрифт, который может состоять из фрагментов других шрифтов, низкоуровневого dvi-кода (например, для рисования линий) или даже команд \special. Таким образом, символ с кодом 47 может быть определен как «взять прописную T из шрифта Times Roman», символ 53 определен как «взять знак = размером 20 пунктов из Computer Modern», а символ 101 — «применить команду \special, чтобы включить рисунок этого логотипа». Виртуальные шрифты используются для того, чтобы заставить TEX думать, что другой шрифт устроен так же, как шрифт Computer Modern с тем, чтобы позволить всем внутренним TEX'овским макроопределениям работать

без проблем или чтобы комбинировать специальные символы из «экспертных шрифтов» с обычным текстовым шрифтом.

Любой виртуальный шрифт состоит из двух частей: метрического файла `tfm`, который используется TEX'ом, и файла `vf`, который используется драйвером и в котором указано, как создать каждый из глифов. Очень часто виртуальный шрифт применяют просто для перекомпоновки шрифта (например, «в качестве символа с кодом 212 взять символ 176 из другого шрифта»), но его также можно использовать для объединения двух шрифтов, добавления произвольных символов или даже изменения цвета, поворота символов и других операций.

Подобно файлам `tfm`, виртуальные шрифты имеют текстовую форму — файл `vpl`. Преобразование между форматами `vf` и `vpl` осуществляется программами `vftovp` и `vptovf`. Ниже в качестве примера мы рассмотрим фрагменты кода виртуального шрифта Baskerville Roman в кодировке `T1`.

Файл `vpl` может содержать те же типы свойств, что и файлы `pl`. Кроме того, появляются и некоторые другие свойства, в частности, описывающие заимствование символов из одного или нескольких шрифтов (для которых также должны существовать `tfm` или `vf`-файлы: заимствование символов может быть рекурсивным). Этими дополнительными свойствами являются `VTITLE`, которое идентифицирует виртуальный шрифт (его значение по умолчанию — пустая строка), свойство `MAPFONT`, в котором указываются шрифты, из которых выбираются символы, и свойство `MAP`, которое используется как часть свойства `CHARACTER` и указывает, откуда символ должен быть взят. Последние два свойства принимают в качестве значений списки свойств, относящихся только к `vf`-файлам.

В нашем примере после идентификации виртуального шрифта Baskerville Roman указываются два базовых шрифта, из которых будут заимствоваться символы:

```
(MAPFONT D 0
   (FONTNAME mbvr8x)
   (FONTCHECKSUM O 33005554065)
   (FONTAT R 1.0)
   (FONTDSIZE R 10.0)
   )
(MAPFONT D 1
   (FONTNAME mbvr8r)
   (FONTCHECKSUM O 7541561315)
   (FONTAT R 1.0)
   (FONTDSIZE R 10.0)
   )
```

Значением `FONTNAME` является имя соответствующего `tfm`-файла, а значение `FONTCHECKSUM` должно совпадать с контрольной суммой этого шрифта, чтобы обрабатывающая программа могла проверить, правильный ли файл найден. Значение `FONTDSIZE` должно совпадать с размером шрифта, а значение `FONTAT`

определяет масштабирующий множитель (т. е., изменяя этот параметр, вы можете создать шрифт, размер которого отличается от размера оригинала).

В простом случае (в приведенном ниже фрагменте — для цифры «4») в записи указываются размер и инструкция для извлечения литеры из одного из базовых шрифтов:

```
(CHARACTER C 4
   (CHARWD R 0.5)
   (CHARHT R 0.654999)
   (MAP
      (SELECTFONT D 1)
      (SETCHAR C 4)
      )
   )
```

А вот лигатура ff, например, может быть взята из другого базового шрифта, которым является экспертный набор для этого шрифта. Нестандартный акцентированный символ может быть создан с помощью dvi-примитивов (PUSH, POP и MOVERIGHT), поместив два базовых глифа один поверх другого:

```
(CHARACTER O 271
   (CHARWD R 0.4269905)
   (CHARHT R 0.6299925)
   (MAP
      (PUSH)
      (MOVERIGHT R 0.046997)
      (SETCHAR O 264)
      (POP)
      (SETCHAR C z)
      )
   )
```

Чтобы показать, что *все*, что допустимо в dvi-файле, может быть использовано в виртуальном шрифте, рассмотрим следующий пример из vpl-файла. Он создает глиф dotlessj (которого нет в большинстве PostScript'овских шрифтов) путем добавления чистого PostScript'овского кода, который берет символ j и маскирует точку. Отметим, однако, что этот пример скорее всего не будет работать на другой платформе, поскольку он основывается на свойстве SPECIAL, которое исполняет чистый PostScript'овский код, интерпретация которого зависит от драйвера.

```
(CHARACTER O 32 (comment dotlessj)
   (CHARWD R 278.00)
   (CHARHT R 458.00)
   (CHARDP R 217.00)
   (CHARIC R 0.00)
   (MAP (SELECTFONT D 0)
   (SPECIAL ps: gsave newpath 0 0 moveto (\31) true charpath
   flattenpath pathbbox /IHeight exch def pop pop pop grestore gsave
   newpath 0 0 moveto (\152) true charpath flattenpath pathbbox pop
```

```
exch /JDepth exch def /JRight exch def /JLeft exch def grestore
gsave newpath)
(PUSH)
(MOVEDOWN R 217.00)
(SPECIAL ps: JLeft JDepth rmoveto JLeft neg JRight add 0 rlineto
   0 JDepth neg IHeight add rlineto JLeft neg JRight add neg 0
   rlineto 0 JDepth neg IHeight add neg rlineto closepath clip)
(POP)
(SPECIAL ps: (\152) show grestore)
)   )
```

Виртуальные шрифты можно также использовать для манипуляции шрифтами, например для расчета метрик для растянутых и сжатых шрифтов, шрифтов для набора вразрядку и капители. В некоторых случаях такая измененная метрика должна дополняться неким PostScript'овским кодом, который изменяет сам шрифт. При использовании dvips это достигается с помощью соответствующих указаний в map-файле. Так, строка[1]

```
hlcbot8rn LucidaTypewriterBoldOblique " .850 ExtendFont
   TeXBase1Encoding ReEncodeFont " <8r.enc <hlcbot8a.pfb
```

означает, что шрифт, известный TEX'у под именем hlcbot8rn, должен быть создан с помощью двух операций над PostScript'овским шрифтом LucidaTypewriterBoldOblique. Сначала шрифт сжимается до 85% от его начальной ширины, а затем перекодируется в кодировку TeXBase1Encoding. Преобразования ExtendFont и ReEncodeFont описаны на языке PostScript в стандартных заголовочных файлах, которые загружает dvips. В команде используются два входных файла: файл определения кодировки (8r.enc) и сам шрифт в формате Type 1 (hlcbot8a.pfb) (в гл. 11 будет более детально описано, как работает dvips). Метрические файлы, которые TEX будет использовать для hlcbot8rn, создаются из обычного метрического файла для LucidaTypewriterBoldOblique при создании виртуального сжатого шрифта.

10.3 Технология PostScript-шрифтов

10.3.1 Типы PostScript-шрифтов

Язык описания страниц PostScript позволяет описывать сложные графические объекты достаточно эффективным способом. Систематическое изложение языка PostScript выходит за пределы данной книги; здесь нам достаточно лишь знать, что каждый глиф в PostScript'овском шрифте представляется небольшой PostScript'овской программой, описывающей *контуры глифа* набором прямых линий и кривых Безье. Чтобы напечатать изображение глифа на выходном устройстве, интерпретатор PostScript'а на лету растеризует эти векторные изображения с учетом разрешения выходного устройства и превра-

[1] Из-за ограничения в длине строки одна строка в конфигурационном файле представлена разбитой на две строки.

щает их в растровые изображения. Это отличает данные шрифты от шрифтов METAFONT, которые растеризуются только один раз при создании gf-файла.

Опишем основные способы представления шрифтов в PostScript'е.

Шрифты Type 1

Формат шрифта Type 1 представляет собой способ компактного описания векторного изображения шрифта с использованием хорошо определенного языка, допускающего быструю интерпретацию. Когда глиф описывается с помощью векторов, его разрешение бесконечно велико: если вы сделаете его в 10 раз больше, его пропорции будут точно такими же, как если бы он был в 10 раз меньше.

При растеризации векторного шрифта основная проблема состоит в том, что пиксел, имеющий конечный физический размер, может быть закрашен либо белым, либо черным. Возьмите лист разграфленной бумаги, строки и столбцы которой состоят из маленьких квадратиков. Нарисуйте большую букву «О» в середине этого листа. Закрасьте все квадратики, которые покрываются этой буквой. Насколько изображение закрашенных квадратиков похоже на букву, которую вы нарисовали? В различии этих изображений и состоит проблема, возникающая при низком разрешении (менее 300 точек на дюйм): какие пикселы надо «включить», а какие оставить «выключенными», чтобы наиболее точно воспроизвести глиф?

Чтобы решить эту проблему, в шрифтах Type 1 используются «хинты», которые подсказывают растеризатору, что делать при малых или больших размерах символов. Все методы хинтования стараются оптимально расположить контуры глифа на сетке из пикселов так, чтобы произвести наиболее радующее глаз и узнаваемое изображение глифа независимо от того, насколько груба сетка. Программа METAFONT, являющаяся частью TEX'а, также выполняет эту функцию.

Программа шрифта Type 1 написана на подмножестве операторов языка PostScript и организована как набор именованных процедур, описывающих контуры глифов. Когда запрашивается номер символа, программа-растеризатор сначала просматривает *вектор кодировки (encoding array)* в словаре шрифта и находит имя глифа, который необходимо построить (это позволяет, перекодируя этот вектор, получить шрифт, отличающийся расположением глифов). Найденное имя глифа далее используется как ключ в словаре `CharStrings` и в результате поиска находится искомая бинарная строка, содержащая векторное описание глифа. Затем этот глиф рисуется.

Перед основным телом шрифта располагается описательная часть. Ниже приведена структура этой части шрифта, которая преобразована в удобочитаемую форму (в качестве примера рассмотрим шрифт Times Roman):

```
%!PS-AdobeFont-1.0: Times-Roman 001.007
%%CreationDate: Tue Mar 20 12:15:38 1990
%%VMusage: 30487 37379
11 dict begin
```

```
/FontInfo 10 dict dup begin
/version (001.007) readonly def
/Notice (Copyright (c) 1985, 1987, 1989, 1990 Adobe Systems Incorporated.
/Copyright ( The digitally encoded machine readable outline data for
producing the Typefaces licensed to you is copyrighted
(c) 1981 Linotype AG and/or its subsidiaries. All Rights Reserved.
 ...
) readonly def
/FullName (Times Roman) readonly def
/FamilyName (Times) readonly def
/Weight (Roman) readonly def
/ItalicAngle 0 def
/isFixedPitch false def
/UnderlinePosition -100 def
/UnderlineThickness 50 def
end readonly def
/FontName /Times-Roman def
/Encoding StandardEncoding def
/PaintType 0 def
/FontType 1 def
/FontMatrix [0.001 0 0 0.001 0 0] readonly def
/UniqueID 28416 def
/FontBBox{-168 -218 1000 898}readonly def
currentdict end
currentfile eexec
```

Важными элементами здесь являются имя шрифта (`FullName`) и кодировка (`Encoding`). Имя нам нужно, чтобы получить доступ к шрифту; кроме того, нам нужно знать его кодировку. В рассматриваемом шрифте кодировкой является «`StandardEncoding`», хотя в других шрифтах может использоваться специфическое расположение глифов в векторе кодировки (см. разд. 10.3.3 на с. 439). За описательной частью шрифта следует закодированная часть, в которой определены сами контуры глифов. Ниже приведен пример раскодированного описания одного из глифов:

```
/asciicircum {
        24 469 hsbw
        0 422 vstem
        297 365 hstem
        68 297 rmoveto
        143 289 rlineto
        143 -289 rlineto
        68 hlineto
        -181 365 rlineto
        -60 hlineto
        -181 -365 rlineto
        closepath
        endchar
        } ND
```

Шрифты Type 1 поставляются разработчиком в формате, подготовленном для одной из операционных систем Macintosh, Windows или Unix. В системе Unix используется формат Printer Font Binary (`pfb`); чтобы читать и редактировать такие шрифты, их следует преобразовать в формат Printer Font ASCII (`pfa`), который используется системой Windows. Преобразования между этими форматами, равно как и преобразование в специальный формат, используемый в компьютерах Macintosh, можно сделать с помощью общедоступного (public-domain) пакета программ `t1utils`, написанных Ли Хетерингтоном, или с помощью пакета Font Manipulation Package, который продает компания Y&Y (см. разд. 10.5.5 на с. 452).

Композитные шрифты

Язык PostScript поддерживает два типа программ шрифтов: базовые шрифты и композитные шрифты. *Базовый шрифт*, как следует из гл. 5 документа *PostScript Language Reference Manual* (Adobe Systems, 1990), является программой шрифта, которая может содержать около 200 глифов, доступных при однобайтовом кодировании.

Программа *композитного шрифта*, которая по классификации фирмы Adobe также называется шрифтом Type 0, является «шрифтом шрифтов». Этот шрифт имеет иерархическую рекурсивную структуру, в которой «дерево» (основной шрифт Type 0, называемый *root (parent) font*) в качестве кодов использует ссылки на нижележащие программы шрифтов Type 1 (*descendant font*) или другие композитные шрифты, которые являются «листьями» этой структуры. Программы композитных шрифтов могут содержать несколько тысяч глифов, доступ к которым осуществляется с помощью многобайтовых кодов. Такие шрифты в особенности полезны для нелатинских письменностей, таких, как кандзи, для которых к тому же может быть характерно письмо в нескольких различных направлениях. Программа шрифта Type 0 может содержать несколько наборов метрики, а ключ в словаре шрифта выбирает тот набор, который используется для визуализации строки.

Шрифты Multiple Master

Шрифты Multiple Master являются расширением шрифтов Type 1, которые позволяют вам подстраивать внешний вид шрифта в создаваемом документе. Используя эти шрифты, вы можете создать сжатую версию заголовка, которая точно соответствует размеру выделенного для него пространства, или взять таблицу или блок текста, который не помещается точно на одной странице, набрать его шрифтами Multiple Master и затем, немного сжимая или растягивая литеры, точно загнать текст в заданные рамки.

Используя технологию шрифтов Multiple Master, улучшенная версия Adobe Type Manager (ATM) помогает обмену форматированными электронными до-

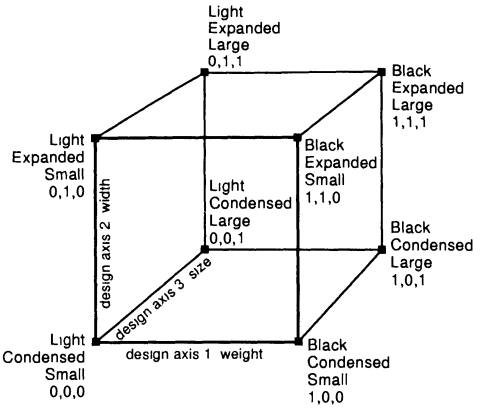


Рис. 10.4. Дизайнерское пространство в шрифтах Multiple Master.

кументами. Эта программа автоматически создает замещающие шрифты, которые используются вместо шрифтов, отсутствующих в вашем компьютере, так что вы можете видеть, редактировать и печатать страницы с сохранением того же расположения текста, переводов строк и других форматирующих элементов, как и в оригинальном тексте. Такая технология используется в Adobe Acrobat.

Любой шрифт Multiple Master состоит из двух или большего числа векторных шрифтов, называемых *основными начертаниями* (*master designs*), с помощью которых определяется одна или большее число *дизайнерских осей* (*design axes*). Основные начертания, лежащие на дизайнерских осях, указывают динамический диапазон изменения одного из типографских параметров, например, насыщенности шрифта или его высоты. Этот диапазон определяется в шрифте Multiple Master тем, что каждому концу дизайнерской оси приводится в соответствие какое-нибудь основное начертание, обозначающее, например, *светлую (light)* или *жирную (black)* насыщенность; при необходимости могут быть указаны и любые *промежуточные основные начертания*. Максимально допустимое число основных начертаний шрифта равно 16.

Основные начертания можно представить как начало координат и координатные оси в одно-, двух-, трех- или четырехмерном пространстве, в котором различным вариантам шрифта отвечают разные точки. Записи в словаре `FontInfo` указывают, что это за пространство и где именно внутри него лежат основные начертания. Эта информация необходима для интерактивных программ, позволяющих пользователю создавать новые образцы шрифтов, и находится в файле `amfm` (Adobe Multiple Font Metrics).

На рис. 10.4 представлено дизайнерское пространство в трехосном шрифте Multiple Master, осями которого являются *насыщенность*, *ширина* и *оптический размер*. На рис. 10.5 и 10.6 показано несколько примеров шрифтов, построенных из «двумерного» шрифта Myriad и «трехмерного» шрифта Minion.

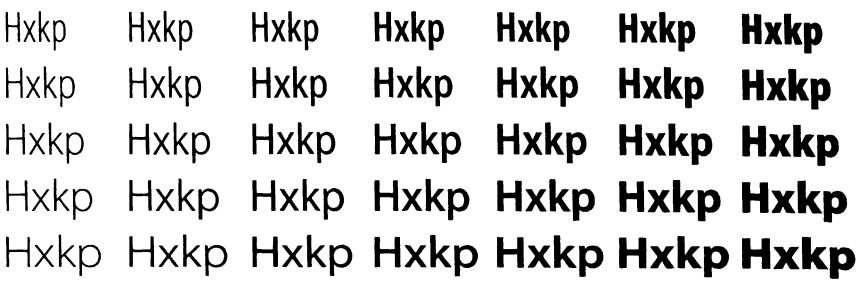


Рис. 10.5. Двумерная сетка с примерами различных вариантов шрифтов, получаемых из «двухосного» Multiple Master рубленого шрифта Myriad.

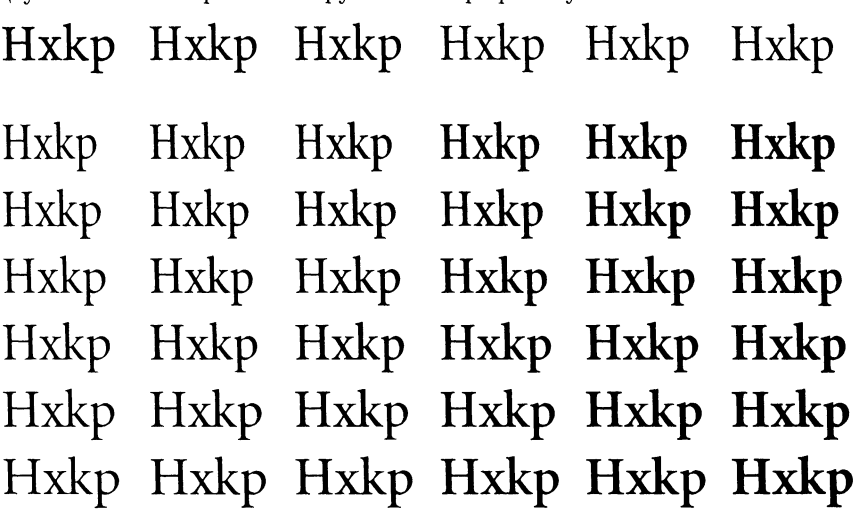


Рис. 10.6. «Трехосный» Multiple Master-шрифт с засечками Minion. В верхнем ряду показаны варианты шрифта с различными оптическими размерами (6pt, 8pt, 11pt, 18pt, 40pt и 72pt), нормализованные к 20pt. Матрица внизу получена путем изменения насыщенности (возрастает слева направо) и ширины (возрастает сверху вниз).

Подключение шрифтов Multiple Master к LaTeX'у, как будет показано в разд. 10.8, сравнительно просто. Книга, которую вы читаете, была набрана с использованием Multiple Master-шрифтов Minion и Myriad фирмы Adobe.[1]

Шрифты Type 3

Существует еще один формат PostScript'овских шрифтов — формат Type 3, в котором разрабатывать шрифт проще, поскольку вы можете использовать любые операторы PostScript'а для создания рисунка глифов. При этом, однако, невозможно использовать хинты и прорисовка (rendering) глифов оказывается

[1] Речь идет об английском оригинале книги. — *Прим. перев.*

менее эффективной, чем для шрифтов Type 1. При подготовке выходных файлов PostScript'а dvi-драйверы часто используют этот тип шрифтов, когда они вставляют растровые шрифты, созданные программой METAFONT из векторного описания: dvi-драйвер извлекает растровые изображения из упакованных pk-файлов для всех TEX'овских шрифтов, использованных в документе, и преобразует их в PostScript'овские шрифты в формате Type 3 для включения в выходной файл. Поскольку эти шрифты получены для данного разрешения и для данного принтера, они не очень хорошо выглядят при печати на другом принтере или при более высоком разрешении (по сравнению с устройством, для которого они создавались). Таким образом, для принтера с более высоким разрешением вам необходимо перегенерировать все pk-файлы для этого разрешения — огромная работа, результаты которой к тому же занимают многие мегабайты дискового пространства. В качестве альтернативы вы можете использовать представления TEX'овских шрифтов в формате Type 1; так, для шрифтов Computer Modern существуют как коммерческие, так и свободно распространяемые версии этих шрифтов. Похожие трудности возникают, когда вы хотите уменьшить размер страницы, чтобы напечатать две или более страниц на одном листе бумаги (используя средства, описанные в гл. 11), поскольку в этом случае вам также приходится масштабировать шрифты.

Шрифты в формате TrueType

Формат TrueType, разработанный фирмами Microsoft и Apple, появился позже формата PostScript. Как и в PostScript'е, изображения глифов в нем хранятся в виде контуров (при этом используются кривые второго порядка, в отличие от PostScript'а, в котором используются кривые третьего порядка), а растеризатор встроен в операционную систему (Mac System 7 или Microsoft Windows и NT). С точки зрения пользователя, это эквивалентно работе с программой Adobe Type Manager: любое приложение может получить доступ к шрифтам и построить их для любого размера и любого из поддерживаемых устройств (как дисплея, так и принтера). Как и PostScript'овские шрифты Type 1, эти шрифты имеют хинты.

Из-за широкого использования операционной системы Windows шрифты TrueType очень распространены, но на удивление мало используются в TEX'е. Частично это объясняется тем, что в настоящее время эти шрифты поддерживаются только коммерческими системами TEX, а частично и тем, что средства, позволяющие создавать метрические файлы TEX'а для шрифтов TrueType, мало известны![1] Пакет TrueTeX, выпущенный Kinch Computer Company, целиком основан на шрифтах TrueType (Kinch также поставляет шрифты семейства Computer Modern в этом формате); к этому пакету также прилагается программа Scientific Word, которая предоставляет дружеский интерфейс к LATEX'у. Вариант VTeX, разработанный и продаваемый фирмой MicroPress, имеет свой

[1] В настоящее время пробел в этой области восполняют программы `ttf2tfm` и `ttf2pk`, входящие в состав пакета ttf2pk (см. архив CTAN, `fonts/utilities/ttf2pk`). — *Прим. перев.*

собственный формат шрифтов, но содержит необходимые программы преобразования.

Формат шрифтов TrueType поддерживается Adobe Acrobat и бесплатно распространяемым интерпретатором PostScript'a — программой Ghostscript[1]; в последнем случае интерпретатор PostScript'a использует инкапсулированные шрифты TrueType в формате Type 42. В середине 1996 г. компании Adobe и Microsoft договорились о сотрудничестве в области объединения форматов шрифтов PostScript и TrueType и создании нового формата Open Type; одним из преимуществ последнего над распространенным в настоящее время форматом Type 1 (который в техническом отношении считается несколько выше TrueType) является то, что он будет представлять собой настоящую 16-битовую систему, что позволит создавать шрифты с более чем 65000 глифами, доступными одновременно.

10.3.2 Метрические файлы в стандарте Adobe

Метрические файлы в стандарте Adobe (Adobe Font Metric, AFM или Adobe Multiple Font Metrics, AMFM) являются стандартными средствами для обмена метрической информацией между программами. Эти чисто текстовые файлы машинно-независимы и допускают расширение. Файл `afm` предоставляет как общую информацию о метрике шрифта, так и информацию об индивидуальных глифах. Файл `amfm` для шрифтов Multiple Master указывает общую информацию обо всем шрифте, тогда как для каждого из основных дизайнов в шрифте имеется индивидуальный `afm`-файл.

Файл `afm` состоит из четырех частей. Первая часть содержит общую информацию о шрифте: его имя, насыщенность, высоту прописных и строчных букв, используемую по умолчанию кодировку и т. п.:

```
StartFontMetrics 2.0
Comment Copyright (c) 1991 Adobe Systems Incorporated.
Comment Creation Date: Fri Sep  6 17:58:27 1991
Comment UniqueID 301073
Comment VMusage 32532 39424
FontName BaskervilleMT
FullName Baskerville MT
FamilyName Baskerville MT
Weight Regular
ItalicAngle 0
IsFixedPitch false
FontBBox -160 -255 1006 902
UnderlinePosition -100
UnderlineThickness 50
Version 001.003
Notice Copyright (c) 1991 Adobe Systems Incorporated.
```

[1] Версия 4.03, которая была доступна во время написания книги не умела, однако, работать с хинтами.

```
EncodingScheme AdobeStandardEncoding
CapHeight 662
XHeight 400
Ascender 672
Descender -246
```

Вторая часть содержит метрическую информацию для каждого глифа. Глифы строятся на воображаемой квадратной сетке с размером стороны в 1000 единиц, относительно которой и рассчитываются метрические параметры. Для каждого глифа запись состоит из четырех полей:

1. численного кода, отвечающего позиции глифа в кодировке по умолчанию. Обратите внимание, что глифы могут иметь код, равный `-1`, который означает, что они недоступны без перекодировки;
2. ширины глифа;
3. имени глифа;
4. размеров бокса для глифа.

Ниже мы приводим лишь небольшой фрагмент этого раздела:

```
StartCharMetrics 228
C 32 ; WX 250 ; N space ; B 180 0 180 0 ;
C 33 ; WX 250 ; N exclam ; B 73 -16 178 656 ;
C 34 ; WX 408 ; N quotedbl ; B 66 388 342 674 ;
C 35 ; WX 668 ; N numbersign ; B 45 -23 622 669 ;
C 36 ; WX 490 ; N dollar ; B 46 -102 444 676 ;
....
C 118 ; WX 458 ; N v ; B -2 -16 460 400 ;
C 119 ; WX 677 ; N w ; B 1 -16 677 400 ;
C 120 ; WX 479 ; N x ; B 8 0 471 400 ;
C 121 ; WX 458 ; N y ; B -2 -246 461 400 ;
C 122 ; WX 427 ; N z ; B 33 0 393 400 ;
C 123 ; WX 480 ; N braceleft ; B 105 -249 378 669 ;
....
C 248 ; WX 240 ; N lslash ; B 1 0 238 672 ;
C 249 ; WX 500 ; N oslash ; B 37 -29 464 429 ;
C 250 ; WX 771 ; N oe ; B 37 -16 744 416 ;
C 251 ; WX 521 ; N germandbls ; B 15 -5 486 672 ;
C -1 ; WX 438 ; N ecircumflex ; B 37 -16 399 616 ;
C -1 ; WX 438 ; N edieresis ; B 37 -16 399 551 ;
C -1 ; WX 469 ; N aacute ; B 39 -10 459 626 ;
C -1 ; WX 760 ; N registered ; B 34 -18 726 674 ;
C -1 ; WX 250 ; N icircumflex ; B -15 0 267 616 ;
EndCharMetrics
```

Еще раз подчеркнем, что код `-1`, приписанный глифу `ecircumflex`, означает, что этот глиф недоступен по умолчанию, т. е. чтобы получить к нему доступ, шрифт должен быть перекодирован.

Третья часть файла содержит информацию о кернинге, т. е. размере невидимого керна (возможно, отрицательного по величине), который вставляется между парами глифов. О первой паре в примере ниже сказано, что если за буквой A следует C, то буквы должны быть немного придвинуты ближе друг к другу.

```
StartKernData
StartKernPairs 496
KPX A C -28
KPX A G -37
KPX A O -37
KPX A Q -37
KPX A T -74
KPX A U -46
KPX A V -120
KPX A W -66
KPX A Y -111
KPX A p 18
....
KPX quoteright v -37
KPX quoteright quoteright 0
KPX quotesinglbase quotesinglbase -28
EndKernPairs
EndKernData
```

Последним идет раздел с информацией о композитных глифах, в котором указывается, как следует строить эти глифы. Например, `Aacute` — это не полноценный глиф, а глиф, построенный наложением буквы A и диакритического знака (акцента) `acute`.

```
StartComposites 56
CC Aacute 2 ; PCC A 0 0 ; PCC acute 172 248 ;
CC Acircumflex 2 ; PCC A 0 0 ; PCC circumflex 172 248 ;
CC Adieresis 2 ; PCC A 0 0 ; PCC dieresis 172 248 ;
CC Agrave 2 ; PCC A 0 0 ; PCC grave 172 248 ;
CC Aring 2 ; PCC A 0 0 ; PCC ring 172 245 ;
....
EndComposites
EndFontMetrics
```

Третий и четвертый разделы являются необязательными. Так, в шрифте математических символов указываются только общая информация и метрика для индивидуальных глифов.

Как вы можете видеть, в отличие от формата `tfm`, формат `afm` содержит только информацию о кернинге, но не содержит лигатур, хотя последняя также нужна для высококачественного набора с помощью TEX'а. Это означает, что программы, преобразующие `afm` в `tfm`, должны получить дополнительную информацию, которая не поставляется вместе с PostScript-шрифтом. О том, как это делается, будет рассказано на с. 456.

10.3.3 Кодировка PostScript-шрифтов

Глифы в PostScript-шрифтах представлены в виде набора процедур, доступ к которым осуществляется путем просмотра таблицы *имен*. Кроме того, в шрифтах присутствует таблица соответствия между именами глифов и их порядковыми номерами (вектор кодировки), но этот способ доступа легко изменить. Таким образом, символ `exclamdown` (¡), который *часто* имеет десятичный код 161, всегда может быть вызван по имени или переотображен на другой код.

Вектор кодировки является обычным массивом PostScript'а, состоящим из 256 элементов:

```
/AnEncoding [          % далее следуют имена 256 глифов
  /grave /acute /circumflex /tilde /dieresis /hungarumlaut
  /ring /caron  /breve /macron /dotaccent /cedilla
  /ogonek /quotesinglbase /guilsinglleft /guilsinglright
  . . . . ]
```

При работе со шрифтом можно указать другой вектор кодировки и он будет использоваться вместо указанного в шрифте вектора кодировки (который используется по умолчанию). К сожалению, общепринятого стандарта расположения символов в кодировке нет; все договариваются о том, какие символы и где они должны быть расположены (речь идет прежде всего о кодах ASCII), но этого недостаточно для серьезной работы. В табл. А.3 и А.4 в приложении А.6 приведено положение латинских букв в следующих широко используемых кодировках:

EC LaTeX'овская кодировка `T1`, также известная как «Cork» (в честь города Корк в Ирландии, где проходила конференция пользователей TeX'а в 1990 г., на которой кодировка `T1` была принята);

TeXBase1 (8r) некая перекодировка, позволяющая получить доступ ко всем стандартным символам в PostScript'овских шрифтах. Используется виртуальными шрифтами в PSNFSS;

ISO Latin 1 расширенный набор ASCII, включающий в себя все символы, используемые в языках западной Европы;

Standard кодировка, разработанная фирмой Adobe и используемая по умолчанию в шрифтах PostScript. Она кодирует *не все* символы в шрифте;

Windows ANSI нестандартная расширенная кодировка ASCII, подобная ISO Latin 1, широко используемая благодаря ее поддержке продуктами фирмы Microsoft;

Mac Roman эквивалент Windows ANSI, используемый всеми приложениями Macintosh;

PDF еще одна кодировка фирмы Adobe, используемая в качестве внутренней стандартной кодировки в файлах PDF (Portable Document Format).

Для пользователей TeX'а, пишущих на русском и других языках на основе кириллицы, представляют интерес еще и кодировки `OT2`, `LCY` и `T2` (`X2`). Кодировка `OT2` («вашингтонская») была одной из первых предложенных кодировок,

использованных в шрифтах WNCYR. Кодировка LCY используется в семействе кириллических шрифтов LH, дополняющих кириллическими глифами шрифты Computer Modern. Кодировка X2 создавалась как кириллический аналог кодировки T1 с тем, чтобы включить в себя максимальное число глифов, используемых в живых языках на основе кириллицы. На основе фрагментов таблицы X2 и таблицы стандартной TEX'овской кодировки T1 были созданы кодировки T2A, T2B, T2C и T2D, позволяющие без переключения языков набирать тексты, содержащие одновременно полный комплект латинских букв и стандартных символов и буквы кириллических алфавитов. Таблица T2A включает в себя глифы из самых распространенных языков на основе кириллицы; она позволяет набирать тексты на русском, украинском, белорусском и еще многих языках Сибири, Средней Азии и Кавказа. В PostScript'овских кириллических шрифтах также используется кодировка Adobe Cyrillic, описанная в документе Adobe Standard Cyrillic Font Specification, Technical Note # 5013.

Все перечисленные выше кодировки, как и сами шрифты PostScript, являются представителями так называемого «восьмибитового» компьютерного мира, предоставляющего для номера кода любого глифа всего 256 различных возможностей. Операционные системы компьютеров, языки и шрифты в настоящее время переходят на 16- и даже 32-битовое представление, что позволяет иметь свыше 65000 (или еще большее число) различных глифов в каждой кодировке. Это значит, что мы получаем возможность однозначно закодировать все литеры из большинства известных в мире алфавитов в рамках одной кодировки. Стандарт Unicode (Unicode Consortium, 1996) описывает подобную 16-битовую кодировку, которая сейчас широко используется; в настоящее время разрабатывается версия TEX'а, называемая Omega (Ω), которая позволяет работать с этой кодировкой.

Однако, работая с *современными* (восьмибитовыми) версиями TEX'а, PostScript'а и шрифтов, важно иметь в виду два обстоятельства:

1. глифы в шрифте PostScript находятся *по имени*. Поэтому для них не существует «правильной» кодировки и никто не запрещает вводить новую кодировку;
2. в шрифте может быть много глифов, которые остаются *незакодированными* в принятой по умолчанию стандартной кодировке Adobe. Более того, в шрифте может быть даже больше 256 глифов, так что ни одна кодировка в принципе не сможет обеспечить доступ ко всем глифам одновременно. К счастью, шрифт может загружаться более чем один раз (в различных кодировках).

10.3.4 Оцифровка PostScript-шрифтов

Превращение векторных изображений в пикселы

При печати векторные контуры глифов переводятся в изображение, которое затем появится на листе бумаги, с помощью устройства, называемого растериза-

тором (raster image processor — RIP), которое строит изображение из большого числа маленьких элементов (пикселов). PostScript'овский код обычно растеризуется в принтере, в котором находится микросхема ПЗУ, содержащая программу интерпретатора PostScript'а и наиболее часто используемые шрифты.[1] С таким же успехом PostScript'овский код может быть растеризован в компьютере и использован для вывода на дисплей. Так, в компьютерах PC и Macintosh растровые изображения шрифтов *по запросу* других приложений (таких, как текстовые процессоры) генерируются программой Adobe Type Manager (см. ниже), а широко используемая бесплатная программа Ghostscript (см. гл. 11), которая доступна для различных компьютерных платформ, может служить примером полного интерпретатора PostScript'а. В операционной системе NeXT значительная часть работы выполняется системой Display PostScript, которая также используется и в системе OpenWindows фирмы Sun.

Оценивая сколько нужно сделать, чтобы преобразовать векторные изображения глифов в огромное число крошечных пикселов, становится ясно, что процессоры, создающие изображение как для принтера, так и для дисплея, требуют значительной производительности. Первый лазерный принтер Apple LaserWriter по-видимому имел более мощный процессор и больше памяти, чем компьютер Macintosh, управлявший им. Для черно-белого принтера с разрешением 300 точек на дюйм вам нужно примерно 1.1 Мбайт памяти, чтобы растеризовать страницу, заполненную текстом, а при разрешении 600 точек на дюйм понадобится примерно вчетверо больше памяти. Поскольку необходимо также оставить место для оперативной работы по растеризации, хранения словарей PostScript'а и распаковывающихся растровых изображений (которые могут присутствовать во входном потоке) и, конечно, самого PostScript'овского кода, отвечающего одной выводимой странице, вам требуется по крайней мере 4 Мбайт оперативной памяти для принтера с разрешением 300 точек на дюйм и минимум 8 Мбайт для принтера с разрешением 600 точек на дюйм.[2] Очевидно, что фотонаборные аппараты и цветные лазерные принтеры требуют еще больше памяти. Устройству для четырехцветной печати на бумаге формата A0 нужно около 70 Мбайт памяти только для того, чтобы хранить растеризованное изображение выводимой на печать страницы.

Чтобы продемонстрировать различие между шрифтами, полученными при разных разрешениях и разными методами, на рис. 10.7 и 10.8 показано слово «PostScript», которое набрано шрифтом Computer Modern и растеризовано с разрешением 85 и 300 точек на дюйм. (В обоих случаях результат показан с большим увеличением, чтобы различия были хорошо видны.) На рис. 10.9 показан тот же текст, полученный из PostScript'овской векторной версии того же шрифта, который был растеризован с разрешением 2400 точек на дюйм.

[1] Здесь речь идет о PostScript'овском лазерном принтере. — *Прим. перев.*

[2] Приведенные оценки объема памяти опять-таки относятся к PostScript'овским лазерным принтерам, в которых генерация изображения происходит в самом принтере; при работе с обычным лазерным принтером изображение создается в компьютере и требования к объему оперативной памяти принтера существенно ниже. — *Прим. перев.*

Рис. 10.7. Шрифт, сгенерированный при разрешении 85 точек на дюйм (типичное разрешение дисплея).

Рис. 10.8. Шрифт, сгенерированный при разрешения 300 точек на дюйм (для принтера LaserWriter).

Рис. 10.9. Векторный шрифт.

Программа Adobe Type Manager

Программа Adobe Type Manager (ATM) является растеризатором шрифтов и используется главным образом для вывода на дисплей в обычных компьютерах. Она перехватывает вызовы программ-приложений к операционной системе по выводу текста и формирует изображение текста вместо нее. Приложения могут также и непосредственно обращаться к ATM через его прикладной программный интерфейс (application program interface — API).

Когда ATM перехватывает вызов по выводу текста, в котором используется шрифт Type 1, программа либо отображает растеризованное изображение глифа на экране дисплея, либо передает их драйверу устройства. Если шрифт не является шрифтом Type 1, то управление возвращается к операционной системе, которая обрабатывает его так, как если бы ATM вообще не было.

ATM допускает, чтобы один и тот же векторный шрифт использовался и на экране дисплея, и на принтере, позволяя добиться наиболее точного соответствия между изображениями на экране и на принтере. Он создает экранные растровые изображения прямо из векторного шрифта, избегая таким образом зазубренных контуров глифов, которые появляются при обычном масштабировании шрифтов. ATM также позволяет печатать шрифтами Type 1 на большом числе не-PostScript'овских принтеров, растеризуя векторные шрифты и автоматически загружая их в принтер.

Оба формата — Type 1 и Type 3 — являются масштабируемыми и могут обрабатываться любым интерпретатором PostScript'а. Однако поскольку шрифт Type 3 требует *полного* интерпретатора PostScript'а для точного воспроизведения всех деталей шрифта, то шрифты Type 3 не следует использовать совместно с программой Adobe Type Manager.

10.4 Классификация PostScript-шрифтов

В этом разделе будет описана версия 2.0 *Fontname* — схемы наименования шрифтов для TEX'а, которая лежит в основе LATEX'овского пакета PSNFSS поддержки PostScript-шрифтов. Этот пакет поддерживается Карлом Берри; полное описание схемы в виде электронного документа можно найти в архиве CTAN, `info/fontname`.

Целью схемы *Fontname* является создание коротких имен файлов шрифтов, которые можно было бы использовать на разных компьютерных платформах. Чтобы обеспечить совместимость с файловой системой DOS и стандартом ISO 9660, используемым при создании CD-ROM, мы должны ограничить длину имени до 8 символов. Кроме того, имена должны состоять только из букв (в одном регистре), цифр и знака подчеркивания. Примером имен, удовлетворяющим этим требованиям, могут служить короткие имена 35 стандартных PostScript'овских шрифтов, приведенные в табл. 10.3.

Основой этой схемы является использование в имени восьми символов, разбитых на следующие семь групп:

S TT W `[`*V...*`]` `[`*N*`]` `[`*E*`]` `[`*DD*`]`,

где

S кодирует *поставщика (supplier)* шрифта,

TT является *сокращенным именем гарнитуры (typeface name)*,

W указывает *насыщенность (weight)*,

Таблица 10.3. Имена стандартных PostScript'овских шрифтов в схеме *Fontname*.

`pagk`	AvantGarde-Book	`phvrrn`	Helvetica-Narrow
`pagko`	AvantGarde-BookOblique	`phvron`	Helvetica-NarrowOblique
`pagd`	AvantGarde-Demi	`pncb`	NewCenturySchlbk-Bold
`pagdo`	AvantGarde-DemiOblique	`pncbi`	NewCenturySchlbk-BoldItalic
`pbkd`	Bookman-Demi	`pncri`	NewCenturySchlbk-Italic
`pbkdi`	Bookman-DemiItalic	`pncr`	NewCenturySchlbk-Roman
`pbkl`	Bookman-Light	`pplb`	Palatino-Bold
`pbkli`	Bookman-LightItalic	`pplbi`	Palatino-BoldItalic
`pcrb`	Courier-Bold	`pplri`	Palatino-Italic
`pcrbo`	Courier-BoldOblique	`pplr`	Palatino-Roman
`pcrr`	Courier	`psyr`	Symbol
`pcrro`	Courier-Oblique	`ptmb`	Times-Bold
`phvb`	Helvetica-Bold	`ptmbi`	Times-BoldItalic
`phvbo`	Helvetica-BoldOblique	`ptmri`	Times-Italic
`phvbrn`	Helvetica-NarrowBold	`ptmr`	Times-Roman
`phvbon`	Helvetica-NarrowBoldOblique	`pzcmi`	ZapfChancery-MediumItalic
`phvr`	Helvetica	`pzdr`	ZapfDingbats
`phvro`	Helvetica-Oblique		

V... обозначает *вариант (variant)* шрифта[1] и опускается, если насыщенность и ширина нормальные. Многие шрифты имеют более чем один вариант;

N указывает *кодировку (encoding)* и опускается, если она нестандартна. Кодировки описаны в разделе, рассматривающем варианты;

E указывает ширину (*expansion*) и опускается, если она нормальная;

DD указывает размер шрифта (в десятичной системе) и опускается, если шрифт масштабируемый.

Насыщенность, начертание и ширину шрифта, по-видимому, лучше всего заимствовать из оригинального полного имени шрифта, чем пытаться определить их, сравнивая с какими-то внешними стандартами. Некоторые семейства шрифтов (в частности, Univers) имеют большое число вариантов, имена которых следуют некоторой рациональной схеме, но это скорее исключение, чем правило.

«Поставщик» — это источник шрифта, обычно указывается (цифровая) словолитня. Вам следует использовать код поставщика, от которого вы получили шрифт, а не оригинального разработчика, поскольку реселлеры обычно вносят изменения в оригинальный дизайн. Например, шрифт AvantGarde был разработан Гербом Любалином для ITC, но Adobe также продает его. Поэтому имя шрифта, который вы получили от Adobe, должно начинаться с буквы p.

[1] Понятие варианта включает в себя и начертание, и кодировку, поэтому это поле используется совместно с полем *N*. — *Прим. перев.*

Таблица 10.4. Поставщики шрифтов, определенные в *Fontname*.

9	неизвестен	
a	autologi	Autologic
b	bitstrea	Bitstream
c	cg	Compugraphic
d	dtc	Digital Typeface Corporation
e	apple	Apple
f	public	свободно распространяемый (например, public domain)
g	gnu	Free Software Foundation
h	bh	Bigelow & Holmes
i	itc	International Typeface Corporation
k	softkey	SoftKey
l	linotype	Linotype
m	monotype	Monotype
n	ibm	IBM
o	corel	Corel
p	adobe	Adobe (p означает PostScript)
r	—	«сырой» (для [вышедшего из употребления] использования с виртуальными шрифтами)
s	sun	Sun
u	urw	URW
z	—	странный (имя шрифта нестандартно)

Полный список известных поставщиков шрифтов представлен в табл. 10.4; во второй колонке указаны предлагаемые аббревиатуры, которые можно использовать для именования директорий. Отметим коды для необычных «поставщиков»:

`f` Шрифты, распространяемые индивидуалами или без специального упоминания их разработчика.

`r` Устаревший код поставщика (указывает на «сырые» шрифты из старых версий dvips). Новые шрифты никогда не должны использовать код `r`. (На самом деле надо указать правильные кодировку, начертание и другие характеристики шрифта.)

`z` Код для шрифтов, которые не подходят ни к одной другой группе в схеме наименования. За буквой `z` должна быть указана буква, отвечающая действительному поставщику.

В схеме *Fontname* понятие «гарнитура» означает коллекцию объединенных единым дизайном шрифтов. Иногда одна и та же аббревиатура гарнитуры используется для шрифтов с разными именами поставщиков, если известно (обычно из сопоставления), что шрифты действительно одни и те же. В приложении A.5 перечислены гарнитуры, известные на момент написания книги; обратите внимание на вариантные имена для шрифтов, созданных компаниями Monotype (m) и Bitstream (b).

Таблица 10.5. Коды насыщенности в *Fontname*.

a	Thin Hairline	l	Light
b	Bold	m	Medium
c	Black	p	Poster
d	Demi	r	Regular Roman
h	Heavy Heavyface	s	Semibold
j	ExtraLight	u	Ultra UltraBlack
k	Book	x	ExtraBold ExtraBlack

«Насыщенность» шрифта указывает степень его жирности. Располагая шрифты по порядку от самого светлого до самого жирного, они образуют следующий ряд: *hairline*, *extra light*, *light*, *book*, *regular*, *medium*, *demibold*, *semibold*, *bold*, *extra bold*, *heavy*, *black*, *ultra* и, наконец, *poster*. В табл. 10.5 представлен список упорядоченных по алфавиту аббревиатур, соответствующих шрифтам разной насыщенности. Это те имена, которые обычно используются в полных названиях шрифтов.

«Начертание» включает в себя вариант гарнитуры (например, наклонный (italic), без засечек (sans-serif)), а также кодировку шрифта (например, Adobe standard, TEX text). Имя шрифта может потребовать нескольких вариантов. Чтобы избежать недоразумений, любой цифровой код кодировки (7, 8 или 9, см. ниже) должен идти последним, а любой цифровой код варианта идет первым (чтобы не перепутать его с кеглем). Рекомендуется (хотя и не требуется), чтобы все другие варианты указывались в алфавитном порядке.

Возможность изменения межбуквенного расстояния, вводимая с помощью fontinst, и виртуальные шрифты еще не стали достаточно распространенными, чтобы сделать подобную стандартизацию полезной и для разных способов генерации фальш-капители. Возможные варианты перечислены в табл. 10.6, которую надо использовать со следующими предостережениями:

1, 2, 4, 6, b Эти варианты (с односторонними засечками, bright, контрастный гротеск и факсовый) использовались только для одной гарнитуры; чтобы сэкономить аббревиатуры вариантов, сейчас эти шрифты используют аббревиатуру начертания.

g, h, j Поскольку эти коды в действительности не использовались нигде, они рассматриваются как устаревшие и в будущем им могут быть присвоены новые значения.

5, 7, 8, 9 Этими цифрами начинаются коды двухсимвольных «вариантов кодировки». В целом, 7 используется для 7-битовых кодировок, 8 — для 8-битовых, а 9 — для «экспертных» кодировок (см. код x ниже), однако это не является абсолютным правилом. Чтобы присвоить шрифту некоторый код кодировки, не нужно, чтобы все глифы в нем находились на местах, предусмотренных кодировкой: достаточно, чтобы работали макроопределения TEX'a. Практически это означает, что шрифт может быть помечен как 7t, если его единственным отличием от Computer Modern является то, что лигатуры и глифы lslash и Lslash в нем другие, поскольку из TEX'a ни

Таблица 10.6. Коды вариантов в *Fontname*.

Во второй колонке указаны коды, использовавшиеся в более ранних версиях схемы наименования шрифтов.

4		факсовый; в настоящее время — `lx`, Lucida Fax	c		капитель
5		префикс для фонетических кодировок	d		шрифт для заголовков
5a		фонетический альтернативный	e		изысканный, элитный
5i		фонетический IPA	f		готический, староанглийский
5s		SIL-IPA	i		курсивный
7		префикс для (главным образом) 7-битовых кодировок	l		контурный
7a		только дополнительные символы	m		математический курсив TEX'a
7c	*2*	Dfr (только измененные символы)	n		неформальный (и шрифты со слегка измененными начертаниями символов)
7d	*9*	цифры в старинном стиле; `OsF` фирмы Adobe	o		наклонный
7f	*3*	дроби	p		орнаменты
7m		TeXMathItalicEncoding	r		с засечками или без (часто опускается)
7t		TeX'овская кодировка текста	s		рубленый
7v		TeXMathExtensionEncoding	t		шрифт пишущей машинки, моноширинный
7y		TeXMathSymbolEncoding	u		выпрямленный курсив
8		префикс для (главным образом) 8-битовых кодировок	v		расширенный математический шрифт TEX'a
8a	*0*	стандартная кодировка (Adobe)	w		рукописный, каллиграфический
8m		стандартная кодировка Macintosh	x		шрифт, построенный с помощью экспертного набора Adobe
8r		кодировка TeXBase1	y		математические символы TEX'a
8t	*q*	кодировка Cork		*1*	с односторонними засечками (устаревший)
8u		XT2Encoding (cmtt + Latin 2)		*6*	контрастный гротеск (устаревший)
8w		кодировка Windows ANSI		*b*	bright; в настоящее время lh
8x		экспертный набор		*g*	с обводкой (устаревший)
8z		кодировка XL2 (cmr + Latin 2)		*h*	оттененный (устаревший)
9		префикс для экспертных наборов		*j*	невидимый (устаревший)
9e		экспертный + Cork		*k*	греческий (устаревший)
9t		экспертный + TEX'овская кодировка текста		*z*	кириллический (устаревший)
a		дополнительный со стрелками			

получить доступ к таблице лигатур, ни изменить лигатуры нельзя. Если в имени шрифта не указан вариант кодировки, кодировка шрифта считается неопределенной.

9 Этот вариант нужен только потому, что код `x`, за которым следует двухбуквенный код варианта, делает некоторые имена слишком длинными. Таким образом, `9t` эквивалентно `x7t`.

k, z Эти коды (указывающие, соответственно, на греческие и кириллические шрифты) являются устаревшими. В будущем шрифты для национальных алфавитов будут маркироваться аббревиатурой кодировки (в настоящее время последние пока не стандартизованы).

m, v, y Эти коды, относящиеся к математическим шрифтам, сохраняются для специальных кодировок математики, как, например, в Lucida. Шрифты, использующие кодировку математики, принятую в Computer Modern, должны использовать коды `7m`, `7v` и `7y`.

r Используется только тогда, когда шрифт не имеет никаких других вариантов (включая кодировку) и если либо код ширины не равен `r`, либо указан кегль; таким образом, код `r` используется просто как указатель позиции. Когда нормальным начертанием гарнитуры является шрифт без засечек (например, Helvetica), при необходимости надо использовать `r`, а не `s`. Код `s` используется только тогда, когда семейство шрифтов имеет как шрифты с засечками, так и шрифты без засечек.

x, 8x `8x` указывает на то, что сам шрифт экспертного набора, а `x` — на то, что шрифт композитный (например, виртуальный), т. е. включает глифы из шрифта с кодом `8x`.

«Ширина» шрифта указывает на сжатие или растяжение шрифта. Упорядоченные по ширине от самого узкого до самого широкого шрифты образуют следующий ряд: *ultra compressed*, *extra condensed*, *compressed*, *condensed*, *narrow*, *regular*, *extended*, *expanded* и, наконец, *wide*.

Растяжение или сжатие шрифтов иногда делается автоматически (например, с помощью PostScript'овского оператора `scale`), а иногда и самими художниками-дизайнерами. В последнем случае дизайнер сам добавляет в имя шрифта слова «Extended», «Expanded», «Narrow» или что-то подобное в соответствии со своими предпочтениями, и в аббревиатуре следует следовать этим указаниям. При создании синтетического расширенного или сжатого шрифта для использования с TEX'ом (например, с помощью afm2tfm или fontinst), используйте коды `n` и `e`. В табл. 10.7 приведены возможные типы кодов ширины.

Бóльшая часть реализаций TEX'а ищет метрический файл `tfm`, просматривая указанное пользователем имя в одной или нескольких директориях. Однако можно было бы поступить иначе и сначала просматривать список имен `tfm`-файлов в *другом* файле (или ряде файлов), в котором были бы указаны настоящие имена файлов. В этом случае именами файлов, используемых в TEX'е, могли бы быть имена, содержащие любые символы и имеющие любую длину. Карл Берри включил такой файл-указатель в свою ориентированную на

Таблица 10.7. Коды ширины в *Fontname*.

c	Condensed, Cond	q	ExtraCompressed, ExtraCondensed
e	Expanded	r	Regular, Normal, Medium
n	Narrow, Thin	u	UltraCompressed
o	UltraCondensed	w	Wide
p	Compressed, Compact	x	Extended, Elongated

Unix реализацию TEX'a — Web2c. Были предложены и другие схемы, включая варианты базы данных шрифтов для X Windows и базы данных шрифтовых ресурсов PostScript. Ни одна из этих схем не нашла пока широкого применения.

10.5 Установка новых PostScript-шрифтов

10.5.1 TEX и кодировки

Сам TEX работает не с именами символов, а с абсолютными числовыми кодами, и, таким образом, переводит запрос на набор глифа в запрос на набор символа, расположенного в определенной позиции в определенном шрифте. Однако когда мы формируем символ с помощью макрокоманды типа `\'{c}`, мы ожидаем, что в результате получим ć; таким образом, TEX'у надо знать, где найти этот символ в шрифте, что является нетривиальной задачей, если положение символов может меняться от шрифта к шрифту. Мы можем встретиться по крайней мере с тремя основными TEX'овскими раскладками символов в шрифте или даже пятью, если мы используем PostScript-шрифты:[1]

1. кодировка, принятая Кнутом в его оригинальных шрифтах Computer Modern, которая в действительности состоит из нескольких чуть-чуть различающихся схем, порой отличающихся всего на одну литеру;
2. расширенная («Cork») кодировка TEX'a;
3. одна из расширенных ASCII-кодировок, например тех, которые используются в программах Windows или Macintosh;
4. стандартная кодировка фирмы Adobe (принятая по умолчанию в PostScript-шрифтах);
5. измененная кодировка для PostScript'a, такая, как описанная ниже кодировка «TeXBase1».

Поэтому, чтобы избежать абсолютного хаоса, либо все макроопределения типа `\'` должны изменяться в соответствии со шрифтом, который мы используем, либо все шрифты должны иметь одну и ту же кодировку.

LATEX называет эти команды «кодозависимыми» и обеспечивает интерфейс для выбора разных определений, который активизируется при изменении кодировки шрифта. Это позволяет нам так определить команду `\'`, чтобы в

[1] Читатели наверняка столкнутся и с другими кодировками, которые осуществляют поддержку кириллических букв (см. с. 440 и русскую документацию на TEX Live 6). — *Прим. перев.*

одной кодировке она создавала композитный символ из акцента и базового символа, а в другой кодировке выбирала один глиф ć (если он присутствует в шрифте). Этот первый и пока единственный макро, который поддерживает эту концепцию; все остальные пакеты макро определяют эти команды статически и, таким образом, сталкиваются с серьезными проблемами при использовании новых шрифтов.

10.5.2 Кодировка TeXBase1

Имена шрифтов в пакете PSNFSS, о котором мы говорили в разд. 10.1.1, соответствуют схеме *Fontname* и включают в себя используемую кодировку. Так, шрифт Times Roman в кодировке OT1 называется `ptmr7t`, а в кодировке T1 — `ptmr8t`. Однако `map`-файлы, сопровождающие PSNFSS (для dvips), используют не эти имена, а ссылаются на промежуточную форму, в которой Times Roman называется `ptmr8r`. При этом каждый шрифт перекодируется в новую стандартную кодировку TeXBase1, используемую для построения TEX'овских виртуальных шрифтов для различных кодировок пользователя; так, оба имени, `ptmr7t` и `ptmr8t`, являются виртуальными шрифтами, которые ссылаются на один и тот же исходный шрифт. Поскольку в типичном PostScript'овском шрифте присутствуют не все глифы кодировки Cork, невозможно просто перекодировать шрифт прямо в конечную форму.

Кодировка исходных шрифтов TeXBase1 была создана специальной группой экспертов в области шрифтов TEX'а, в первую очередь Карлом Берри, Бертольдом Хорном, Аланом Джеффри, Пьером Макеем и Себастианом Ратцем. Ее целью было сделать доступными для набора все глифы, обычно входящие в состав шрифтов Type 1. Это — глифы из стандартной кодировки Adobe плюс кодировки ISO Latin 1 и еще несколько символов, доступных в Lucida Bright. Кодировка создавалась на основе следующих правил:

1. Символы, входящие в Windows ANSI, остаются почти все в тех же позициях, поскольку пользователи Windows не могут просто перекодировать шрифты, а для других систем это не имеет значения. Несколько символов из Windows ANSI, которые становятся недоступными, не имеют значения при наборе документов, это — rubout (код 127), неразрывный пробел (код 160), и мягкий перенос (код 173). Глифы «quotesingle» и «grave» передвинуты только из-за того, что то, что они находятся не на своих привычных TEX'овских позициях, очень раздражает.
2. Остальные символы произвольно распределены по нижней части таблицы за исключением кодов в позициях 0, 10 и 13, которые не изменяются и позволяют программам нормально работать, считая, что эти символы используются для выполнения приписанных им действий.
3. Lucida Bright имеет несколько дополнительных текстовых символов. В надежде на то, что и другие PostScript'овские шрифты, возможно, с целью удовлетворения запросов пользователей, будут включать их в будущем, эти символы включены начиная с позиции 18.

	'0	'1	'2	'3	'4	'5	'6	'7	
'00x		˙	fi	fl	⁄	˝	Ł	ł	"0x
'01x	˛	°		˘	–		Ž	ž	
'02x	ˇ	ı	ȷ	ff	ffi	ffl			"1x
'03x							`	'	
'04x		!	"	#	$	%	&	’	"2x
'05x	(	)	*	+	,	-	.	/	
'06x	0	1	2	3	4	5	6	7	"3x
'07x	8	9	:	;	<	=	>	?	
'10x	@	A	B	C	D	E	F	G	"4x
'11x	H	I	J	K	L	M	N	O	
'12x	P	Q	R	S	T	U	V	W	"5x
'13x	X	Y	Z	[	\	]	^	_	
'14x	‘	a	b	c	d	e	f	g	"6x
'15x	h	i	j	k	l	m	n	o	
'16x	p	q	r	s	t	u	v	w	"7x
'17x	x	y	z	{	|	}	~		
'20x			‚	*f*	„	…	†	‡	"8x
'21x	ˆ	‰	Š	‹	Œ				
'22x				“	”	•	–	—	"9x
'23x	˜	™	š	›	œ			Ÿ	
'24x		¡	¢	£	¤	¥	¦	§	"Ax
'25x	¨	©	ª	«	¬	-	®	¯	
'26x	°	±	²	³	´	µ	¶	·	"Bx
'27x	¸	¹	º	»	¼	½	¾	¿	
'30x	À	Á	Â	Ã	Ä	Å	Æ	Ç	"Cx
'31x	È	É	Ê	Ë	Ì	Í	Î	Ï	
'32x	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	"Dx
'33x	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	
'34x	à	á	â	ã	ä	å	æ	ç	"Ex
'35x	è	é	ê	ë	ì	í	î	ï	
'36x	ð	ñ	ò	ó	ô	õ	ö	÷	"Fx
'37x	ø	ù	ú	û	ü	ý	þ	ÿ	
	"8	"9	"A	"B	"C	"D	"E	"F	

Рис. 10.10. Раскладка глифов в кодировке TeXBase1 (шрифт Lucida Bright).

4. Оставшиеся позиции оставлены неопределенными для использования в (мы надеемся) последующих версиях, если еще какие-нибудь глифы станут общеупотребительными.
5. Глиф «hyphen» появляется в таблице дважды, для совместимости при работе с ASCII и с Windows.

На рис. 10.10 показано расположение глифов в кодировке TeXBase1 в привычном TEX'овском виде. В табл. А.3 и А.4 из приложения А.6 эта кодировка сопоставляется с другими широко используемыми кодировками.

10.5.3 Создание TEX'овской метрики для PostScript-шрифтов

Хотя LATEX может работать с различными кодировками, нам необходимо решить, какую из них использовать в конкретном шрифте, и создать для него TEX'овскую (`tfm`) метрику. Каждый PostScript-шрифт обычно выпускается с файлом `afm` (см. разд. 10.3.2 выше), в котором содержится вся необходимая для этого информация. Как будет показано, существует несколько программ, позволяющих преобразовать эту информацию в метрические файлы системы TEX.

10.5.4 От AFM к TFM и VF: средства преобразования

Существует пять общераспространенных программ, которые могут преобразовать формат `afm` в `tfm` и по возможности одновременно создавать дополнительные виртуальные шрифты:

1. утилиты Font Manipulation Tools фирмы Y&Y;
2. программа afm2tfm, распространяемая вместе с dvips;
3. пакет fontinst, созданный Аланом Джеффри;
4. программа Edmetrics, распространяемая с версией TEX'а Textures (для Macintosh);
5. TEX'овская программа просмотра dviwindo фирмы Y&Y, имеющая развитый интерфейс с Adobe Type Manager и поддерживающая прямую генерацию метрических файлов для TEX'а из самих шрифтов.

В следующих двух разделах мы кратко рассмотрим первые два варианта и уделим несколько больше внимания третьему. Детали, касающиеся шрифтов для Macintosh, лежат вне круга интересов этой книги, и пользователям системы Textures лучше обратиться к документации, поставляемой вместе с ее программным обеспечением.

10.5.5 Средства манипуляции шрифтами фирмы Y&Y

Средства манипуляции шрифтами (Font Manipulation Tools), которые работают только в DOS, предоставляют ряд очень полезных программ, позволяющих сделать много больше, чем просто перевести `afm` в `tfm`. Вот некоторые из них:

afmtopfm преобразование `afm` в метрический формат системы Windows (`pfm`);

afmtotfm преобразование `afm` в метрику шрифта системы TEX;

composit добавление новых композитных глифов в шрифт формата Type 1;

mactopfa преобразование формата Type 1 для Macintosh в формат общего вида `pfa`;

mergepfa слияние шрифтов формата Type 1;

pfatoafm извлечение метрики `afm` из шрифта в формате `pfa`;

pfatopfb преобразование формата `pfa` в двоичный формат `pfb`;

pfbtomac преобразование формата `pfb` в формат Macintosh;

pfbtopfa преобразование формата `pfb` в формат `pfa`;

pfmtoafm преобразование метрического файла Windows формата `pfm` в `afm`-формат;

reencode изменение кодировки по умолчанию в шрифте формата Type 1;

sidebear изменение размеров апрошей в шрифте формата Type 1;

subfont извлечение подмножества глифов из шрифта формата Type 1;

tfmtoafm преобразование метрического файла системы TEX в `afm`.

Часть этих программ нужна потому, что на различных компьютерных платформах используются шрифты и метрические файлы в своем собственном формате. Большинство указанных преобразований являются «точными» (в том смысле, что потери информации при преобразовании не происходит), хотя следует заметить, что ни одно из преобразований *в* `afm`-формат не позволяет получить столько информации, сколько содержится в настоящем `afm`-файле. Наиболее важными программами являются те, которые манипулируют с самими шрифтами формата Type 1; пользователи TEX'а могут применять их для практически любых действий, описанных выше в отношении виртуальных шрифтов. При этом вместо манипуляций с промежуточными шрифтами и осуществления изменений на уровне виртуальных шрифтов они изменяют сам шрифт формата Type 1, который затем используется непосредственно с TEX'ом. Это позволяет нам:

1. перекодировать шрифт;
2. добавить новые композитные (акцентированные) символы;
3. объединить шрифты, чтобы добавить в них капитель или дополнительные символы из экспертных шрифтов.

Полученные шрифты могут непосредственно использоваться такими системами, как Adobe Type Manager. При этом нет необходимости в «слое» виртуальных шрифтов, но, с другой стороны, это также означает, что пользователь должен иметь доступ к файлам самих шрифтов формата Type 1 (у многих они встроены в принтер) и вынужден использовать их нестандартные копии. Поэтому система виртуальных шрифтов хотя и создает впечатление громоздкости, в целом оказывается более гибкой и может работать на различных компьютерных платформах.

10.5.6 Программа afm2tfm

Программу afm2tfm Тома Рокицки можно использовать для преобразования файла `afm` в TEX'овский `tfm`-файл. По умолчанию он сохраняет ту же кодировку, что и `afm`-файл, но может также создавать и виртуальный шрифт, в котором производится что-то похожее на перекодировку, а также осуществлять такие преобразования, как растяжение и наклон глифов. Это — программа для

Unix или DOS с управляющей командной строкой, в которой управляющие ключи задаются следующим образом (опции указаны в квадратных скобках):

afm2tfm *inputfile* [-O] [-u] [-v | -V *vplfile*] [-e *expansion*]
[-s *slant*] [-c *capheight*] [-p | -t | -T *encodingfile*] *outputfile*

Обычно в результате работы программа создает tfm-файл, однако некоторые ключи позволяют также создать и vpl-файл (который следует затем обработать программой vptovf, чтобы получить другой tfm-файл и vf-файл). Одновременно программа выводит на стандартное устройство вывода соответствующие команды для map-файла программы dvips.

Первые две опции позволяют растянуть или наклонить шрифт; такие шрифты требуют добавления дополнительной информации в соответствующей строке в map-файле, и afm2tfm сделает для вас это автоматически:

-e *ratio*
Растягивает глифы по горизонтали в *ratio* раз; если *ratio* меньше 1.0, то глифы сжимаются.

-s *slant*
Наклоняет глифы вправо на величину *slant*; отрицательное значение аргумента позволяет наклонять глифы *влево*. Эта опция позволяет получить наклонный шрифт или (забавный эффект!) выпрямленный курсив.

Следующие четыре опции определяют кодировку шрифта и то, необходимо ли создавать виртуальный шрифт для этой кодировки:

-p *encodingfile*
Указывает, что шрифт PostScript должен быть перекодирован с использованием вектора кодировки *encodingfile* (при этом лигатуры и кернинг игнорируются). afm2tfm считает, что соответствующий вектор кодировки имеется. Файл с этой кодировкой (enc-файл) должен быть указан в строке map-файла для этого шрифта.

-t *encodingfile*
Использует TEX'овскую кодировку *encodingfile* при создании tfm-файла; информация о лигатурах и кернинге берется из файла кодировки.[1]

-T *encodingfile*
Использует TEX'овскую кодировку *encodingfile* и в шрифте PostScript, и в создаваемом tfm-файле.

-v *file*
Создает список параметров, используемый для получения виртуального шрифта (virtual property list, vpl-файл), и tfm-файл на основе указанной кодировки.

[1] Речь идет об использовании информации, записанной в строках LIGKERN в enc-файле. — *Прим. перев.*

Если вам нужна фальш-капитель, то afm2tfm использует технологию виртуальных шрифтов[1]:

`-V` *file*
: То же, что и `-v`, но полученный виртуальный шрифт отвечает капители, которая получается масштабированием прописных букв (по умолчанию, до 0.8 от их начального размера) и помещается в позиции строчных букв. Этот подход правильно работает с акцентированными буквами и кернингом.

`-c` *fraction*
: При создании капители (с помощью опции `-V`) параметр масштабирования капительных букв изменяется со значения, принятого по умолчанию (0.8), на *fraction*.

Наконец, имеются две общие опции, указываемые без аргументов:

`-O` Заставляет код глифа в `vpl`-файле указывать в восьмеричной системе счисления. Это полезно при обработке шрифтов, содержащих глифы символов, чтобы глифы, располагающиеся в позициях букв A–Z и a-z, не помечались как буквы и не смущали читателя.

`-u` Использует только глифы, присутствующие в выходной (TEX'овской) кодировке. Без этой опции программа включает не только все глифы в выходной кодировке, но и любые другие глифы, которые присутствуют в шрифте.

Каково же действительное назначение опций `-p`, `-t` и `-T`? Программа afm2tfm может изменять кодировку выходного файла двумя различными способами. Во-первых, вы можете просить PostScript настроить кодировку так, чтобы она соответствовала расположению глифов в TEX'е и просто использовала его стандартную кодировку. Для этого вы используете опцию `-p`. При этом виртуальный шрифт *не требуется*, но нужно, чтобы вы загружали необходимый вектор кодировки вместе со шрифтом. Во-вторых, вы можете решить оставить PostScript'овский шрифт неизменным и создать виртуальный шрифт. Это позволяет TEX'у работать в его обычной кодировке, но заставляет `dvi`-драйвер перекодировать символы с тем, чтобы находить их правильное положение в PostScript-шрифте. Для этой цели вы используете опцию `-t`. При этом в файле кодировки вы можете указать лигатуры и информацию о кернинге, которая будет использована совместно с информацией, содержащейся в `afm`-файле.

Чтобы скомбинировать действия, производимые `-p` и `-t`, вы можете использовать опцию `-T`. Это означает, что вы указываете желаемую выходную кодировку и afm2tfm создает виртуальный шрифт, который соответствует ей, и вы также говорите, что PostScript'овский шрифт закодирован таким же способом (при этом вам все равно нужно будет загружать вектор кодировки вместе со шрифтом). Различие между этой опцией и `-p` состоит в том, что в виртуаль-

[1] К сожалению, создание фальш-капители для кириллических букв невозможно без заметного изменения исходного текста программы afm2tfm. — *Прим. перев.*

ный шрифт добавляется информация о лигатурах и кернинге. Таким образом, эта опция придает наибольшую гибкость программе и является наиболее часто используемой опцией для текстовых шрифтов.

Файлы кодировки afm2tfm имеют совершенно такой же формат, что и вектор кодировки в любом PostScript-шрифте. Вот его общий вид:

```
/MyEncoding [ % точно 256 имен, каждое из которых начинается с /
    /Alpha /Beta /Gamma /Delta ...
    ... /A /B ... /Z
    /aacute /logicalnot /eth ] def
```

Комментарии в файле кодировки, которые начинаются со знака процента и идут до конца строки, игнорируются до тех пор, пока они не начинаются со слова LIGKERN. Первое «слово» файла, которое должно начинаться с косой черты (PostScript'овское имя литерала), определяет имя кодировки. Следующим словом должна быть открывающаяся квадратная скобка [. После этого должно следовать точно 256 имен глифов; при этом все неопределенные символы обозначаются /.notdef. В конце списка должна стоять закрывающаяся квадратная скобка]. Завершающий токен def не проверяется, но обычно необходим для интерпретаторов PostScript'а. Обратите внимание на то, что прописные и строчные буквы в именах глифов считаются различными.

Любая информация, касающаяся лигатур и кернинга, передается через комментарии[1]. Если первым словом после знака процента является LIGKERN, то вся оставшаяся часть строки разбирается в поиске информации о лигатурах и кернинге. Эта информация записывается в виде группы слов, каждая из которых завершается точкой с запятой, с пробелом перед ней и пробелом после нее (если это не конец строки).

В строках, помеченных словом LIGKERN, могут присутствовать три типа информации: пары лигатур; керны, которые следует удалить или игнорировать; и символ, используемый в TEX в качестве указателя границы слова. Какой из трех типов информации указан в строке, автоматически определяется по ее синтаксису. Символ-указатель границы слова определяется как | | и устанавливается командой, подобной | | = 39 ;.

Чтобы указать на удаление керна, задают имена двух глифов (без начальной косой черты), разделенные знаком {}, как в one {} one ;. Это напоминает использование {} в TEX'овских файлах для отключения лигатур или кернинга в определенном месте. Любой из этих глифов или сразу оба могут быть помечены символом *, означающим, что в этом месте может стоять любой символ. Таким образом, кернинг может быть полностью отменен с помощью конструкции * {} * ;.

Чтобы указать лигатуру, задают имена двух символов, за которыми следуют «операция» лигатуры (как в METAFONT'е) и имя замещающего глифа. Любым из первых двух символов (но не обоими вместе) может быть символ

[1] В частности, в кириллических шрифтах в комментариях указывается информация о дополнительных лигатурах для кавычек, которые используются в русских правилах типографского набора. — *Прим. перев.*

|, указывающий на границу слова. Обычно знаком «операции» является =:, который означает, что оба символа убираются и заменяются третьим символом, однако, добавляя символы | с любой стороны от =:, вы можете указать, что один из двух символов должен быть сохранен. Кроме того, добавляя справа к оператору лигатуры один или два знака >, вы можете указать, что при операции сканирования лигатур должно быть пропущено указанное число символов перед продолжением обработки. Это работает так же, как и в METAFONT'е. Примером простой лигатуры может быть лигатура `ff i =: ffi ;`. Более сложным примером может быть лигатура `one one |=:|>> exclam ;`, которая предписывает, чтобы каждая пара соседствующих цифр 1 была разделена восклицательным знаком и затем два первых полученных символа пропущены перед продолжением поиска лигатур и кернов. В операциях лигатуры не допускается большее число знаков >, чем знаков |, так что всего возможно восемь операций с лигатурами:

```
=: |=: |=:> =:| =:|> |=:| |=:|> |=:|>>
```

которым отвечают восемь вариантов параметров LIG в файлах pl, описанных на с. 425.

Набор лигатур и кернов, используемых по умолчанию в afm2tfm, был рассчитан так, чтобы произвести те же действия, какие использовал Кнут в шрифтах Computer Modern. Это обеспечивает эквивалентный вид TEX'овских документов, набранных разными шрифтами. Ниже приведены соответствующие этим лигатурам и кернам строки в файле кодировки:

```
% LIGKERN question quoteleft =: questiondown ;
% LIGKERN exclam quoteleft =: exclamdown ;
% LIGKERN hyphen hyphen =: endash ; endash hyphen =: emdash ;
% LIGKERN quoteleft quoteleft =: quotedblleft ;
% LIGKERN quoteright quoteright =: quotedblright ;
% LIGKERN space {} * ; * {} space ; 0 {} * ; * {} 0 ;
% LIGKERN 1 {} * ; * {} 1 ; 2 {} * ; * {} 2 ; 3 {} * ; * {} 3 ;
% LIGKERN 4 {} * ; * {} 4 ; 5 {} * ; * {} 5 ; 6 {} * ; * {} 6 ;
% LIGKERN 7 {} * ; * {} 7 ; 8 {} * ; * {} 8 ; 9 {} * ; * {} 9 ;
```

Пример использования программы afm2tfm со шрифтом cheq был рассмотрен в разд. 8.1.5.

10.5.7 Пакет fontinst

Наиболее гибким способом создания виртуальных шрифтов для TEX'а является использование пакета fontinst Алана Джеффри. Этот пакет написан на TEX'е и позволяет пользователю задавать входную метрику шрифта, процедуры, которые надо исполнить, и формат желаемого вывода. TEX исполняет ваши указания и, например, читает afm-файлы (он также умеет читать и файлы pl) и записывает виртуальный шрифт в ASCII-формате (vpl-файл). Последние преобразуются в vf-файлы с помощью стандартной программы vptovf. В этом

разделе мы опишем только команды высокого уровня fontinst и приведем примеры некоторых других команд; детали, касающиеся команд низкого уровня пакета fontinst, описаны в приложении A.7.

Пакет fontinst использует три типа файлов:

- *управляющие файлы*, в которых вы пишите команды TEX'а для установки шрифтов;
- *файлы кодировок* (например, `t1.etx`), в которых содержится информация о кодировке, включая кодовую таблицу, лигатуры и размеры шрифтов;
- *метрические файлы* (файлы описания глифов), в которых содержится информация о наборах глифов (шрифтов), включая размеры глифов, композитные символы и кернинг.

Так, файл `latin.mtx` является *метрическим файлом*, описывающим *совокупность* латинских букв, которые могут встретиться в шрифте. Файл `t1.etx` задает *порядок*, в котором они должны появляться, а файл `ptmr8r.mtx` содержит действительную совокупность доступных символов.

Макрокоманды пакета fontinst загружаются с помощью команды

```
\input fontinst.sty
```

а файл заканчивается командой `\bye`. Управляющий файл fontinst обычно состоит либо из вызова высокоуровневых макрокоманд, как, например, для простого случая обработки целого семейства латинских шрифтов:

`\latinfamily{`*family name*`}{`*extra .fd code*`}`

либо из вызова одной или нескольких структур общего вида:

```
\installfonts
  команды установки
\endinstallfonts
```

Интерфейс `\latinfamily`

Многие пользователи не требуют от fontinst ничего более сложного, чем просто создать совместимую с PSNFSS метрику для нового семейства шрифтов. Команда `\latinfamily` как раз и осуществляет эту функцию. Она рассчитана для работы с набором `afm`-файлов, имена которых следуют схеме *Fontname*, и этой команде надо указать лишь код изготовителя и имя семейства. Используя эту информацию, команда пытается исполнить команду `\installfont` для каждой комбинации насыщенности и начертания, для которой она может найти соответствующий `afm`-файл. Она перекодирует шрифт в TeXBase1 (`8r`) и генерирует метрику для двух кодировок — `OT1` и `T1`. Экспертные шрифты, если они доступны (они опознаются по дополнительной букве «x», добавленной к названию семейства), также могут быть использованы (для создания капители, добавления лигатур и т. п.). Если к имени семейства добавлена цифра «9»,[1] то цифры в шрифтах будут представлены в старинном стиле.

[1] В описании fontinst указано, что для этого используется буква «j». — *Прим. перев.*

В простейшем случае необходимый для fontinst командный файл оказывается не сложнее приведенного ниже примера, который делает все необходимое по установке шрифтов семейств Times (ptm), Helvetica (phv) и Courier (pcr):

```
\input fontinst.sty
\latinfamily{ptm}{}
\latinfamily{phv}{}
\latinfamily{pcr}{\hyphenchar\font=-1}
\bye
```

Интерфейс `\installfont`

Теперь рассмотрим группу *команд установки*, в которых указываются шрифты, глифы и кодировки, которые надо использовать при построении шрифтов.

`\installfamily{`*encoding*`}{`*family*`}{`*fd commands*`}`

Эта команда подключает семейство шрифтов *family* к LaTeX'у в кодировке *encoding*. Например, для установки семейства Optima в кодировке Cork, надо написать:

```
\installfamily{T1}{pop}{}
```

В поле *fd commands* указываются команды LaTeX'а, которые записываются в файл описания для создаваемого семейства шрифтов и исполняются каждый раз, когда шрифт этого семейства загружается. Например, чтобы запретить переносы в семействе Courier, надо сказать:

```
\installfamily{T1}{pcr}{\hyphenchar\font=-1}
```

`\substitutesilent{`*old series or shape*`}{`*new series or shape*`}`
`\substitutenoisy{`*old series or shape*`}{`*new series or shape*`}`

Эти две команды указывают LaTeX'у какие замещения насыщенности или начертания в шрифте могут быть сделаны если запрашиваемое начертание или насыщенность шрифта отсутствуют.

`\transformfont{`*target_font*`}{`*generation_command*`}`

Эта команда создает для внутренних целей файл *target_font* с расширением mtx путем исполнения команды *generation_command*, которой может быть одна из следующих команд:

`\scalefont{`*integer_expression*`}{`*font*`}`

Отмасштабировать шрифт *font* в горизонтальном и вертикальном направлениях в *integer_expression* раз.

`\xscalefont{`*integer_expression*`}{`*font*`}`

Отмасштабировать шрифт в горизонтальном направлении в *integer_expression* раз.

`\yscalefont{`*integer_expression*`}{`*font*`}`

Отмасштабировать шрифт в вертикальном направлении в *integer_expression* раз.

`\slantfont{`*integer_expression*`}{`*font*`}`

Наклонить шрифт на величину *integer_expression*.

`\reencodefont{`*etx_file*`}{`*font*`}`

Перекодировать шрифт в кодировку, указанную в *etx_file*.

В поле *font* в каждой из описанных команд может быть указан вызов любого другого преобразования, однако чаще всего в этом поле указывают

`\fromafm{`*afm*`}`

при котором из `afm`-файла создается промежуточный файл, который в дальнейшем может использоваться программой fontinst.

Примером вызова команды преобразования является следующая строка:

```
\transformfont{pplr8r}{\reencodefont{8r}{\fromafm{pplr8a}}}
```

Она означает, что из оригинального `afm`-файла для шрифта Palatino (`pplr8a.afm`) путем перестановки символов с тем, чтобы следовать кодировке 8r, создается файл метрического описания шрифта `pplr8r.mtx`. После этого `pplr8r` может быть использован в командах `\installfont`. Важно иметь в виду, что эта перекодировка (как и другие преобразования) относится только к *метрике*; сам шрифт должен быть изменен точно так же при его загрузке (в разд. 11.2.4 мы рассмотрим, как это делает dvips).

`\installfont{`*font*`}{`*files*`}{`*etx_file*`}{`*encoding*`}{`*family*`}{`*series*`}{`*shape*`}{`*size*`}`

Эта команда создает TEX'овский шрифт с именем *font* из списка (разделенных запятыми) файлов *files*, которыми могут быть файлы `mtx`, `afm` и `pl`. Шрифт TEX'а кодируется с помощью файла *etx_file*`.etx` и становится доступным в LATEX'е как шрифт семейства *family* в кодировке *encoding* с насыщенностью *series*, начертанием *shape* и размером *size*. Например, чтобы установить шрифт Times Roman в кодировке Cork, надо сказать:

```
\installfont{ptmr8t}{ptmr8r,latin}{T1}{T1}{ptm}{m}{n}{}
```

При этом программа возьмет список *доступных* метрик из файла `ptmr8r.mtx`, посмотрит в `latin.mtx` описание, как строить глифы, и создаст записи в выходном файле для всех глифов, перечисленных в файле кодировки `t1.etx` (заметим, что, хотя название кодировки задается прописными буквами, в имени файла оно записывается строчными буквами). Отсутствие последнего аргумента в примере означает, что один и тот же шрифт будет использоваться для всех размеров.

Метрические файлы

Метрический файл `latin.mtx` имеет смысл рассмотреть более подробно. Он определяет все характеристики широко используемых шрифтов и иллюстрирует возможности fontinst. Метрические файлы могут содержать любую информацию (например, документацию) вплоть до первой команды `\metrics`, которая указывает, что мы готовы начать:

```
\metrics
\needsfontinstversion{1.334}
```

Одной из важных особенностей пакета fontinst является то, что он написан на знакомом нам языке макропрограммирования TEX. Вот, например, макро, описывающий, как создать капительную букву, добавляя небольшие пробелы с обеих сторон уменьшенного глифа:

```
\setcommand\cscglyph#1{
   \movert{\int{smallcapsextraspace}}
   \glyph{#1}{\int{smallcapsscale}}
   \movert{\int{smallcapsextraspace}}
}
```

Все размеры шрифта TEX'а и другие параметры могут быть заданы в абсолютных величинах или рассчитаны. Разумеется, это можно изменить с помощью загружаемых управляющих файлов. Все размеры задаются по отношению к сетке 1000×1000.

```
\setint{italicslant}{0}
\setint{xheight}{\height{x}}
\setint{capheight}{\height{A}}
\setint{ascender}{\height{d}}
\setint{descender}{\depth{g}}
\setint{underlinethickness}{40}
\setint{visiblespacedepth}{200}
\setint{visiblespacewidth}{400}
\setint{visiblespacesurround}{50}
\setint{smallcapsscale}{800}
\setint{smallcapskerning}{900}
```

Сложность используемого кернинга остается на усмотрение автора управляющего файла. Ниже приведены некоторые установки fontinst, определенные по умолчанию в `latin.mtx`:

```
\setleftrightkerning{Aacute}{A}{1000}
\setleftrightkerning{Abreve}{A}{1000}
...
\setleftkerning{ffi}{f}{1000}
\setleftkerning{ffl}{f}{1000}
...
\setleftrightkerning{Aacutesmall}{Aacute}{\int{smallcapskerning}}
\setleftrightkerning{Abrevesmall}{Abreve}{\int{smallcapskerning}}
\setleftrightkerning{Acircumflexsmall}{Acircumflex}{\int{smallcapskerning
```

Далее идет список символов, используемых в латинском алфавите. Первый набор определяется с помощью макро `\unfakable`, который вызывает генерацию сообщения-предупреждения, если какой-нибудь символ в шрифте отсутствует. Порядок символов не имеет значения.

```
\unfakable{Gamma}
\unfakable{Delta}
\unfakable{Theta}
...
\unfakable{guillemotleft}
\unfakable{guillemotright}
\unfakable{endash}
\unfakable{emdash}
...
\unfakable{germandbls}
\unfakable{lslashslash}
```

Остальные символы задаются с помощью определения, которое используется, когда в шрифте нет соответствующего реального глифа:

```
\setglyph{fi}
   \glyph{f}{1000}
   \movert{\kerning{f}{i}}
   \glyph{i}{1000}
\endsetglyph
\setglyph{ff}
   \glyph{f}{1000}
   \movert{\kerning{f}{f}}
   \glyph{f}{1000}
\endsetglyph
...
\setglyph{Abreve}
   \topaccent{A}{breve}{500}
\endsetglyph
\setglyph{Aogonek}
   \botaccent{A}{ogonek}{900}
\endsetglyph
```

```
56  ...
57  \setglyph{Qsmall}
58     \cscglyph{Q}
59  \endsetglyph
```

Наконец, мы вновь возвращаемся к плавающим акцентам с тем, чтобы установить для них нулевую глубину, которую требует TEX. Следующие команды показывают, как могут быть переопределены существующие определения:

```
60  \setcommand\zerodepth#1{
61     \resetglyph{#1}
62        \glyph{#1}{1000}
63        \resetdepth{0}
64     \endresetglyph
65  }
66  \zerodepth{acute}
67  \zerodepth{grave}
68  \zerodepth{caron}
69  \zerodepth{dieresis}
70  ....
71  \endmetrics
```

Все, что следует после `\endmetrics`, программой fontinst игнорируется, т. е. после этой команды может быть добавлена любая текстовая документация.

Файлы кодировки

Основное содержание файла кодировки — совокупность команд `\setslot`, указывающих, какие символы должны быть включены в кодировку. Эти команды обычно задают соответствие строчных и прописных букв для слота и создают лигатуры (см. примеры ниже). Отметим важность *порядка* следования символов в файле, поскольку, например, номер кода буквы A определяется ее положением в файле кодировки. (Номера слотов в файле кодировки автоматически увеличиваются при переходе к следующему слоту; если ряд слотов пропускается, то номер следующего слота может быть явно установлен командой `\nextslot`.)

```
\setslot{\uc{A}{a}}
   \comment{Буква `{A}'.}
\endsetslot
\setslot{hyphen}
   \ligature{LIG}{hyphen}{rangedash}
   \ligature{LIG}{hyphenchar}{hyphenchar}
   \comment{Дефис `-'.}
\endsetslot
\setslot{\lclig{FL}{fl}}
   \comment{Лигатура `fl'.  В моноширинном шрифте ширина равна
      двум литерам.}
\endsetslot
\setslot{dollar}
```

```
    \comment{Знак доллара '\$'.}
\endsetslot
\setslot{\lc{Breve}{breve}}
    \comment{Акцент breve '\u a'.}
\endsetslot
```

Так как большинству пользователей fontinst не нужно понимать все детали `latin.mtx` и файлов кодировки, приведенные фрагменты дают лишь общее представление об их устройстве.

Примеры командных файлов `\installfont`

Проанализируем несколько примеров реальных управляющих файлов fontinst. Наш первый пример взят из системы поддержки шрифтов семейства Adobe Minion. Сначала загружаем сам макро, устанавливаем группу `\installfonts` и инициализируем кодировку и семейство:

```
\input fontinst.sty
\needsfontinstversion{1.317}
\installfonts
\installfamily{T1}{pmn}{}
```

Далее определяем некоторые общие замещения, указывая, что у нас нет «bold extended» и поэтому вместо него надо использовать «bold» и что у нас нет «upright italic» и вместо него надо использовать «italic»:

```
\substitutesilent{bx}{b}
\substitutenoisy{ui}{it}
```

Теперь нам надо создать три совокупности шрифтов: прямой, заголовочный, и сверхжирный. В каждой группе мы сначала перекодируем базовый шрифт, а затем используем его в `\installfont`:

```
\transformfont{pmnr8r}{\reencodefont{8r}{\fromafm{pmnr8a}}}
\transformfont{pmnri8r}{\reencodefont{8r}{\fromafm{pmnri8a}}}
\installfont{pmnr8t}{pmnr8a,pmnr8x,latin}{T19}{T1}{pmn}{m}{n}{<-14>}
\installfont{pmnri8t}{pmnri8a,pmnri8x,latin}{T19}{T1}{pmn}{m}{it}{<-14>}
\installfont{pmnrc8t}{pmnr8a,pmnr8x,latin}{T1c}{T1}{pmn}{m}{sc}{<-14>}
```

Можно видеть, что каждый шрифт производится из трех метрических файлов: базового шрифта, сопровождающего его экспертного шрифта (который содержит дополнительные лигатуры, цифры в старинном стиле и т. п.) и набора латинских букв. Порядок, в котором они вызываются, является критичным; команды в `latin.mtx` зависят от предшествующих определений совокупности доступных символов, так как он может анализировать значения и переопределять их в случае необходимости. Кодировка создаваемого файла имеет суффикс 9, который означает, что по умолчанию должны будут использоваться цифры в старинном стиле.

Первая группа указанных шрифтов используется только до размера символов в 14 пунктов. Для бóльших размеров используется специальное подмно-

жество заголовочных шрифтов (Display), которое подключается следующими командами:

```
\transformfont{pmnrd8r}{\reencodefont{8r}{\fromafm{pmnrd8a}}}
\transformfont{pmnrdi8r}{\reencodefont{8r}{\fromafm{pmnrdi8a}}}
\installfont{pmnrd8t}{pmnrd8a,pmnrd8x,latin}{T19}{T1}{pmn}{m}{n}{<14->}
\installfont{pmnrdi8t}{pmnrdi8a,pmnrdi8x,latin}{T19}{T1}{pmn}{m}{it}{<14-
\installfont{pmnrdc8t}{pmnrd8a,pmnrd8x,latin}{T1c}{T1}{pmn}{m}{sc}{<14->}
```

Наконец, обеспечиваем поддержку сверхжирного шрифта:

```
\transformfont{pmnc8r}{\reencodefont{8r}{\fromafm{pmnc8a}}}
\installfont{pmnc8t}{pmnc8a,pmnc8x,latin}{T19}{T1}{pmn}{eb}{n}{}
\endinstallfonts
\bye
```

В результате обработки указанного выше командного файла с помощью TEX'а получаем совокупность файлов `pl`, виртуальных шрифтов `vpl` и `fd`-файл для LATEX'а. Чтобы создать `tfm`- и `vf`-файлы, файлы `vpl` остается обработать программой `vptovf`, а файлы `pl` — программой `pltotf`.

В нашем втором примере строятся несколько кириллических шрифтов в кодировке `OT2` с использованием Monotype Times. Как и раньше, начнем с загрузки `fontinst` и определения версии, которая нам нужна:

```
\input fontinst.sty
\needsfontinstversion{1.317}
```

Теперь воспользуемся преимуществами TEX'а и включим в макро то, что мы хотели бы сделать для каждого компонента семейства шрифтов. Для заданной комбинации начертания и насыщенности мы хотим сначала загрузить кириллический шрифт (суффикс z), а затем соответствующий латинский шрифт (для цифр, знаков пунктуации и т. п.); любые символы, которые нужны нам для кодировки и которые отсутствуют в кириллическом шрифте, будут заимствованы из латинского шрифта. Последний перед этим перекодируется в стандарт `8r`:

```
\def\TimesCyrillicFont#1#2#3#4{% Fontname name, series, shape,
                                % alternate name
   \transformfont{#48r}{\reencodefont{8r}{\fromafm{#48a}}}
   \installfont{#17k}
      {#1z,#48r,monocyr}
      {monoot2}{OT2}{mnt}{#2}{#3}{}
}
```

Метрический файл `monoot2.mtx` определяет порядок, в котором должны появляться символы, например:

```
\nextslot{060}\setslot{guillemotleft}\endsetslot
\nextslot{061}\setslot{dotlessi}\endsetslot
\nextslot{062}\setslot{guillemotright}\endsetslot
\nextslot{063}\setslot{question}\endsetslot
\nextslot{064}\setslot{breve}\endsetslot
\nextslot{065}\setslot{As}\endsetslot
```

```
\nextslot{066}\setslot{Buki}\endsetslot
\nextslot{067}\setslot{Zui}\endsetslot
```

а `monocyr.mtx` определяет совокупность доступных символов, имена которых соответствуют используемым в Monotype'овских `afm`-файлах, например:

```
\unfakable{guillemotright}
\unfakable{question}
\unfakable{cyrbreve}
\unfakable{As}
\unfakable{Buki}
\unfakable{Zui}
\unfakable{Dobro}
```

Определив макро, теперь мы можем выполнить его для нормального, курсивного, полужирного и курсивного полужирного начертаний и завершить работу:

```
\installfonts
   \installfamily{OT2}{mnt}{}
   \TimesCyrillicFont{mntr}{m}{n}{mntr}
   \TimesCyrillicFont{mntro}{m}{it}{mntri}
   \TimesCyrillicFont{mntb}{b}{n}{mntb}
   \TimesCyrillicFont{mntbo}{b}{it}{mntbi}
\endinstallfonts
\bye
```

Наш третий пример — командный файл, предназначенный для построения математических шрифтов с использованием Multiple Master-шрифта Minion и шрифта Lucida New Math (который изначально был разработан для английского оригинала этой книги, но в конечном счете не был использован). В разд. 10.1.1 на с. 413 мы уже описывали пакет `mathptm`. Он состоит из стилевого файла, переопределяющего небольшие части математического шрифта и определяющего новые семейства шрифта, однако основная работа выполняется на уровне виртуальных шрифтов. Последние были созданы с помощью управляющего файла `fontinst`, в котором особое внимание было уделено тому, чтобы все глифы имели правильные размеры и характеристики и пользователю `mathptm` не нужно было беспокоиться об этом. Этот же подход можно достаточно просто использовать и для создания других смешанных шрифтов, например сочетания математического шрифта Computer Modern с Garamond, или, как в нашем примере, подход к созданию которого во многом заимствован из пакета `mathptm` Алана Джеффри.

Как и ранее, после загрузки `fontinst` определим макро, который будет выполнять бóльшую часть работы:

```
\input fontinst.sty
\needsfontinstversion{1.5}
\def\MathSize#1#2#3#4{% start of size range,
                      % end of size range,
                      % font size, scaling
```

Что этот макро делает? Мы хотим создать шрифт математических символов (в кодировке OML) и математический шрифт для букв и цифр (в кодировке OT1) для нового семейства, которое мы назовем zmnluc. Для обоих шрифтов мы будем использовать символы из уменьшенного шрифта Lucida New Math и одного из вариантов Multiple Master-шрифта Minion. Поскольку у нас есть варианты шрифта Minion шести различных размеров, мы будем использовать для каждого из размеров самый лучший из них, а шрифт Lucida просто отмасштабируем, чтобы подогнать его под необходимый размер.

Как обычно, начинаем с построения версий текстовых шрифтов в кодировке 8r:

```
\transformfont{zmnm8r#3}{\reencodefont{8r}{\fromafm{zmnm8a#3}}}
\transformfont{zmnmi8r#3}{\reencodefont{8r}{\fromafm{zmnmi8a#3}}}
\transformfont{zmnbi8r#3}{\reencodefont{8r}{\fromafm{zmnbi8a#3}}}
\transformfont{zmnb8r#3}{\reencodefont{8r}{\fromafm{zmnb8a#3}}}
```

и вызываем \installfont:

```
  \installfont{zmnlucrm#3}
    {kernoff,hlcrima scaled #3,kernon,unsetluc,zmnmi8r#3
        scaled 1075,lmathit,lhax}
     {LOML}{OML}{zmnluc}{m}{it}{<#1-#2>}
```

Важным здесь является список метрических файлов, вызываемых для построения шрифта в кодировке OML. По порядку это:

1. kernoff.mtx, который просто содержит команды, запрещающие кернинг, который не нужен нам для символов, заимствованных из математического шрифта;
2. hlcrima.mtx, определяющий совокупность символов в LucidaNewMath-AltItalic;
3. kernon.mtx, восстанавливающий кернинг;
4. unsetluc.mtx, *удаляющий* символы, заимствованные ранее из шрифта Lucida, которые мы хотим взять из Minion. Он состоит из строк такого вида:

   ```
   \unsetint{xheight}
   \unsetint{ascender}
   \unsetint{descender}
   \unsetint{capheight}
   \unsetint{italicslant}
   \unsetglyph{A}
   \unsetglyph{B}
   ...
   ```

5. Сам шрифт Minion, символы из которого заполняют слоты, освобожденные нами на предыдущем этапе;
6. lmathit.mtx, который определяет расположение символов в создаваемом шрифте;

7. `lhax.mtx`, который производит некоторые дополнительные манипуляции с символами (меняя кернинг, переставляя их и т. п.). Эти изменения имеют целью заставить символы из обычного текстового шрифта иметь характеристики математического шрифта. Текст этого модуля целиком принадлежит Алану Джеффри.

Полные тексты tex из упомянутых файлов, которые не являются частью стандартного пакета fontinst, можно найти в архиве CTAN, `info/lgc`.

Описанные действия повторяются для полужирного шрифта:

```
    \installfont{zmnlucbf#3}
       {kernoff,hlcdima scaled #3,kernon,unsetluc,zmnbi8r#3
        scaled 1075,lmathit,lhax}
       {LOML}{OML}{zmnluc}{b}{it}{<#1-#2>}
```

После этого то же делается и для кодировки OT1 для прямого, курсивного и полужирного начертаний. Это проще, поскольку на выходе здесь получается совокупность латинских букв, хотя мы можем заимствовать несколько полезных символов (таких, как греческие буквы) из Lucida:

```
    \installfont{zmnlucmi#3}
       {hlcrima scaled #3,unsetluc,zmnmi8r#3 scaled 1075,latin}
       {OT1I}{OT1}{zmnluc}{m}{it}{<#1-#2>}
    \installfont{zmnlucmn#3}
       {hlcrm scaled #3,unsetluc,zmnm8r#3 scaled 1075,latin}
       {OT1}{OT1}{zmnluc}{m}{n}{<#1-#2>}
    \installfont{zmnlucbn#3}
       {hlcdm scaled #3,unsetluc,zmnb8r#3 scaled 1075,latin}
       {OT1}{OT1}{zmnluc}{b}{n}{<#1-#2>}
}
```

Теперь мы готовы строить шрифты. Определим два множества выходных кодировок (OT1 и OML) и вызовем основной макро для каждого из строящихся диапазонов размеров:

```
\installfonts
\installfamily{OT1}{zmnluc}{}
\installfamily{OML}{zmnluc}{\skewchar\font=127}
\MathSize{}{6.5}{6}{990}
\MathSize{6.5}{7.5}{7}{970}
\MathSize{7.5}{8.5}{8}{970}
\MathSize{8.5}{10.5}{10}{940}
\MathSize{10.5}{12.5}{12}{930}
\MathSize{12.5}{14.5}{14}{930}
\MathSize{14.5}{}{17}{920}
\endinstallfonts
\bye
```

Остается еще одна вещь: информировать LaTeX о новых математических шрифтах. Это немного сложнее, чем просто переопределить `\rmdefault`. Необходимый минимум включает определение математических групп «letters»,

«operators», «symbols» и «largesymbols» (буквы, операторы, символы, большие символы) и установку полужирной и нормальной версий операторов и букв. При этом считается, что семейство обычных текстовых шрифтов под именем zmn уже установлено:

```
\DeclareSymbolFont{letters}{OML}{zmnluc}{m}{it}
\DeclareSymbolFont{operators}{OT1}{zmn}{m}{n}
\DeclareSymbolFont{symbols}{OMS}{hlcy}{m}{n}
\DeclareSymbolFont{largesymbols}{OMX}{hlcv}{m}{n}
\SetSymbolFont{letters}{normal}{OML}{zmnluc}{m}{it}
\SetSymbolFont{letters}{bold}{OML}{zmnluc}{b}{it}
\SetSymbolFont{operators}{bold}{OT1}{zmn}{b}{n}
\SetSymbolFont{operators}{normal}{OT1}{zmn}{m}{n}
```

В действительности мы можем использовать еще больше математических шрифтов Lucida, создав несколько дополнительных шрифтов, например, прямой математический (upright math) и дополнительный шрифт математических символов (в котором много всяких стрелок, откуда и происходит его название «arrows»):

```
\DeclareSymbolFont{mathupright}{OML}{hlcm}{m}{n}
\DeclareFontEncoding{LMR}{}{}
\DeclareFontSubstitution{LMR}{hlcm}{m}{n}
\DeclareSymbolFont{arrows}{LMR}{hlcm}{m}{n}
\DeclareSymbolFont{boldarrows}{LMR}{hlcm}{b}{n}
```

Поскольку шрифт Lucida со стрелками содержит полный шрифт «blackboard bold», мы можем связать его с командой \mathbb, чтобы использовать его для набора ажурных букв:

```
\DeclareSymbolFontAlphabet{\mathbb}{arrows}
```

Кроме того, мы можем переопределить некоторые распространенные имена символов, чтобы выбирать их из шрифта «arrows»:

```
\DeclareMathSymbol{\rightleftharpoons}{3}{arrows}{"7A}
\DeclareMathSymbol{\leftrightharpoons}{3}{arrows}{"79}
```

и определить несколько новых стрелок:

```
\DeclareMathSymbol{\dashuparrow}{0}{arrows}{"39}
\DeclareMathSymbol{\dashdownarrow}{0}{arrows}{"3B}
```

Прямой математический (math upright) шрифт Lucida содержит ряд полезных прямых греческих символов, к которым мы также можем получить доступ:

```
\DeclareMathSymbol{\upalpha}{\mathalpha}{mathupright}{11}
\DeclareMathSymbol{\upbeta}{\mathalpha}{mathupright}{12}
```

На рис. 10.11 показан пример текста, набранного таким смешанным шрифтом.

The default math mode font is *math italic*. This should not be confused with ordinary *text italic*. `\mathbf` produces **bold-face roman** letters. If you wish to have ***bold-face math italic*** letters and bold-face Greek letters and mathematical symbols, use the `\boldmath` command *before* going into math mode. This changes the default math fonts to bold. Greek is available in upper and lower case: $\alpha, \beta, \Gamma, \Delta \ldots \omega, \Omega$

normal: $x = 2\pi \Rightarrow x \simeq 6.28$

mathbf: $\mathbf{x = 2\pi \Rightarrow x \simeq 6.28}$

boldmath: $\boldsymbol{x = 2\pi \Rightarrow x \simeq 6.28}$

There is also a calligraphic font for upper-case letters; these are produced by the `\mathcal` command: $\mathcal{ABCDE}$

$$\phi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx \tag{1}$$

$$\prod_{j\geq 0} \left(\sum_{k\geq 0} a_{jk} z^k \right) = \sum_{k\geq 0} z^n \left(\sum_{\substack{k_0, k_1, \ldots \geq 0 \\ k_0 - k_1 + \cdots = n}} a_0 k_0 a_{1k_1} \ldots \right) \tag{2}$$

$$\pi(n) = \sum_{m=2}^{n} \left\lfloor \left(\sum_{k=1}^{m-1} \lfloor (m/k) / \lceil m/k \rceil \rfloor \right)^{-1} \right\rfloor \tag{3}$$

$$\underbrace{\{\overbrace{a, \ldots, a}^{k\ a's}, \overbrace{b, \ldots, b}^{l\ b's}\}}_{k+1 \text{ elements}} \tag{4}$$

$$\mathrm{W}^+ \begin{cases} \nearrow \mu^+ + \nu_\mu \\ \rightarrow \pi^+ + \pi^0 \\ \rightarrow \kappa^+ + \pi^0 \\ \searrow \mathrm{e}^+ + \nu_\mathrm{e} \end{cases}$$

$$\pm \begin{vmatrix} x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \end{vmatrix}$$

$$\begin{vmatrix} l_1 & m_1 \\ l_2 & m_2 \end{vmatrix}^2 + \begin{vmatrix} m_1 & n_1 \\ n_1 & l_1 \end{vmatrix}^2 + \begin{vmatrix} m_2 & n_2 \\ n_2 & l_2 \end{vmatrix}^2$$

10-5-1

Рис. 10.11. Пример математического текста, набранного шрифтом, полученным при смешении Multiple Master-шрифта Minion и шрифта Lucida New Math.

10.6 Шрифты Multiple Master

Шрифты Multiple Master можно использовать одним из трех способов:

1. совершенно автономно с помощью таких программ, как Adobe Acrobat, которая генерирует вариант шрифта, удовлетворяющий определенным требованиям;
2. в рамках улучшенной версии Adobe Type Manager (ATM), которая позволяет пользователю интерактивно поэкспериментировать с установками, чтобы получить желаемый шрифт;
3. прямой манипуляцией с метрическими файлами и записью соответствующих команд на PostScript'е.

Для пользователей LaTeX'а второй способ оказывается самым простым, поскольку в результате экспериментирования создаются `afm`-файл и некий фрагмент PostScript'овского кода («корень»), который загружается с самим шрифтом. Этот `afm`-файл может быть далее преобразован в TeX'овскую метрику обычным образом. Если LaTeX работает вместе с программой просмотра (previewer) и/или драйвера принтера, имеющего доступ к ATM, то этот подход очень удобен. Коммерческие системы TeX'а для Windows от Y&Y полностью поддерживают и описывают эту процедуру, так что единственной заботой пользователя такой системы остается выбрать параметры для вариантов шрифта в рамках ATM.

Пользователи, работающие с шрифтами Multiple Master на компьютерах без ATM (пользователи Unix и DOS), также могут генерировать варианты шрифтов с помощью метрических файлов шрифта и бесплатной программы Ghostscript. Ниже в этом разделе мы опишем подробности, которые представляют интерес для тех, кто просто хочет понять, как это делается, или тех, кто хочет рассчитывать варианты шрифтов автоматически, полагаясь на алгоритмический, а не визуальный подход.

Шрифты Multiple Master приходят с набором основных `afm`-файлов, называемым AMFM (Adobe Master Font Metrics)[1] Эти файлы содержат информацию о числе основных дизайнов, числе дизайнерских осей, массивы `BlendDesignPositions` и `BlendDesignMap`, а также имена и вектор весов (`weightvector`) для основных дизайнов, из которых строятся любые варианты шрифтов.

Чтобы получить истинную метрическую информацию о глифах в шрифте, нам нужно скомбинировать метрическую информацию из основных дизайнов (в случае шрифта Minion их восемь). Для этого нам нужно рассчитать значения вектора весов `weightvector` для заданного варианта шрифта. Стартуя из дизайнерского пространства, мы можем использовать оператор `NormalizeDesignVector`, чтобы перейти в нормализованное координатное про-

[1] К сожалению, производители шрифтов в настоящее время перестали поставлять `amfm`-файлы со своими шрифтами, объясняя это тем, что для системы ATM, работающей под Windows, эти файлы не нужны. — *Прим. ред.*

странство, откуда с помощью оператора `ConvertDesignVector` мы рассчитываем вектор весов. Эти два оператора являются характеристиками шрифта (поскольку они зависят от основных дизайнов) и хранятся в словаре шрифта Multiple Master. Мы можем декодировать этот PostScript'овский код, используемый для расчета вектора весов, и перевести его на другой компьютерный язык, чтобы затем, комбинируя значения параметров в `afm`-файлах основных дизайнов, рассчитывать их значения для конкретного варианта шрифта. Например, в случае шрифта MinionMM, код PostScript определяет восемь компонент вектора весов следующим образом:

$$\begin{array}{lcllcl} w_1 & = & xyz & w_2 & = & (1-x)yz \\ w_3 & = & x(1-y)z & w_4 & = & (1-x)(1-y)z \\ w_5 & = & xy(1-z) & w_6 & = & (1-x)y(1-z) \\ w_7 & = & x(1-y)(1-z) & w_8 & = & 1-\sum_{n=1}^{7} w_n \end{array}$$

где x — нормализованная насыщенность, y — нормализованная ширина, а z — нормализованный оптический размер. Эти восемь чисел w_i позволяют рассчитать все параметры, необходимые для `afm`-файла заданного варианта шрифта. Для этого мы по очереди считываем значение каждого параметра из восьми `afm`-файлов для основных дизайнов и умножаем их на соответствующий вес; полученная таким образом взвешенная сумма и является искомым интерполированным значением рассчитываемого параметра в варианте шрифта.

Myriad является рубленым шрифтом, сочетающимся с Minion, который имеет две дизайнерские оси и состоит из четырех основных дизайнов. Веса для расчета параметров производных шрифтов в нормализованном координатном

Рис. 10.12. Варианты шрифта Minion, лежащие на противоположных концах оси оптического размера, после приведения к одному и тому же размеру (с большим увеличением).

пространстве определяются следующим образом:

$$\begin{array}{rclrcl} w_1 & = & (1-x)(1-y) & w_2 & = & (1-x)y \\ w_3 & = & x(1-y) & w_4 & = & xy \end{array}$$

где x — нормализованная насыщенность, а y — нормализованная ширина. Соответствующие параметры, используемые для перехода из дизайнерского пространства к нормализованным координатам, равны:

```
BlendDesignPositions [ [0 0] [1 0] [0 1] [1 1] ]
BlendDesignMap [[[215 0][830 1]][[300 0][700 1]]]
BlendAxisTypes [/Weight /Width ]
```

Теперь мы можем рассчитать любой из размеров обрамляющего бокса или параметры кернинга для заданного варианта шрифта, используя значения искомых параметров в восьми (или четырех в случае Myriad) основных файлах и вычисляя их интерполированное значение.

После того как afm-файл для производного шрифта создан, соответствующие метрические файлы для LaTeX'а могут быть построены обычным способом.

10.6.1 Генерация производных файлов с помощью Ghostscript'а

Описанные выше идеи были реализованы в виде Unix'овского скрипта (makeinstance) и программы разбора afm-файлов фирмы Adobe (mmafm), которые можно найти в архиве CTAN, `fonts/utilities/mm`. Основной скрипт делает следующее:

1. создает небольшой PostScript'овский файл, чтобы вызвать операторы Multiple Master со значениями, передаваемыми скрипту;
2. исполняет этот файл с помощью Ghostscript'а, чтобы получить нормализованные веса и записать их во временный файл. Заметим, для этого необходимо, чтобы версия Aladdin Ghostscript была 3.33 или старше, поскольку более ранние версии программы не умеют работать со шрифтами Multiple Master;
3. запускает программу mmafm, которая читает основные afm-файлы и записывает afm-файл для производного шрифта.

В результате работы скрипта получаются файл afm и заголовочный файл pro. Этот afm-файл теперь можно использовать для создания TeX'овской метрики для кодировки 8r и виртуальных шрифтов в кодировке EC для действительной работы. Кроме этого, необходимо добавить запись в map-файл dvips, в которой следует указать имя файла заголовка и реальное имя шрифта Multiple Master.

Поскольку скрипт достаточно короткий, мы приведем его здесь полностью:

```
#!/bin/sh
#
#Usage: makeinstance fontname instancename design-space-variables 1 to 4
X=$2.control
```

```
cat > $$.instance <<CLOSE
/mmfile (mmfile) (w) file def
/str 20 string def
/MASTERFONT findfont dup begin [
V1 V2 V3 V4 NormalizeDesignVector ConvertDesignVector
counttomark 1 sub -1 0 {
index str cvs mmfile exch writestring
mmfile (\n) writestring} for
mmfile closefile
] end
CLOSE
#-----------------------------------
MASTERFONT="$1"
V1="$3"
V2="$4"
V3="$5"
V4="$6"
sed -e "s/MASTERFONT/$MASTERFONT/" \
    -e "s/V1/$V1/" \
    -e "s/V2/$V2/" \
    -e "s/V3/$V3/" \
    -e "s/V4/$V4/" \
    < $$.instance > $$.ps
rm $$.instance
gs -dNODISPLAY `kpsewhich dvips_header $1.PFA` $$.ps quit.ps
rm $$.ps
#-----------------------------------
# build control file for mmafm
#-----------------------------------
# number of weights
wc -l mmfile | awk '{print $1}'  > $N
cat $1.afmnames >> $N
echo $2 >> $X
cat mmfile >> $X
#-----------------------------------
rm mmfile
mmafm $X | \
 sed -e "/^FontName/s/^FontName .*$/FontName $1-$3-$4-$5-$6/" \
     -e "/^FontName/s/-*$//" > $2.afm
#-----------------------------------
echo "/$2 /$1 findfont dup begin [" > $2.pro
echo "   $V1 $V2 $V3 $V4  NormalizeDesignVector ConvertDesignVector" >>$2.pro
echo "       ] end makeblendedfont definefont pop " >>$2.pro
#-----------------------------------
exit
```

Вызов этого скрипта с параметрами:

```
MinionMM zmnl8ac6 360 460 6
```

создаст метрический файл с именем zmnl8ac6, которому, согласно схеме *Fontname*, соответствует шрифт «Minion, light weight, 8a-encoded, condensed, at 6pt design size». Соответствующая ему строка в map-файле dvips имеет вид:

```
zmnl8rc6 zmnl8ac6 "TeXBase1Encoding ReEncodeFont
       <8r.enc <MinionMM.pfb <zmnl8ac6.pro
```

где пролог-файл zmnl8ac6.pro содержит инструкции для интерпретатора PostScript'а о том, как заданный вариант шрифта должен генерироваться из кода Multiple Master-шрифта MinionMM.pfb. Файл zmnl8ac6.pro содержит следующий код:

```
/zmnl8ac6 /MinionMM findfont
dup begin [
  360 460 6 NormalizeDesignVector
  ConvertDesignVector
  ] end makeblendedfont definefont pop
```

Обратите внимание на присутствие PostScript'овских операторов NormalizeDesignVector и ConvertDesignVector, о которых мы говорили выше.

Кроме этого, нам потребуются файлы описания шрифтов, чтобы сообщить LaTeX'у, как соотносятся шрифты с различной насыщенностью и шириной с LaTeX'овскими определениями насыщенности и начертания. Единственным осложнением здесь является то, что шрифт Minion имеет ось оптического размера, и нам нужно, чтобы LaTeX использовал различные варианты шрифта при различном размере букв:

```
\DeclareFontShape{T1}{zmn}{lc}{n}{
        <-7>zmnl8tc6
        <7-10>zmnl8tc8
        <10-15>zmnl8tc11
        <15->zmnl8tc18}{}
```

Влияние оптического размера иллюстрирует рис. 10.12 на с. 472, на котором сопоставлены шрифты 6pt и 18pt, приведенные к одному размеру. Различия в дизайне здесь так же заметны, как и в соответствующих примерах шрифтов Computer Modern.

Продвинутый TeX-эксперт, используя Unix'овскую систему Web2c, может написать скрипт MakeTeXTFM;[1] который позволяет обращаться к программам на лету прямо из TeX'а; это может позволить макрокомандам TeX'а управлять параметрами при создании вариантов шрифтов. Основной проблемой здесь является то, что при использовании пакета fontinst, описанного в этой главе, работа происходит слишком медленно (несколько минут, чтобы создать каждый шрифт), однако ситуацию можно улучшить, используя другие средства.

[1] Когда TeX'у не удается загрузить шрифт, система Web2c может быть сконфигурирована так, что будет вызывать внешнюю программу для создания шрифта, загружать его и затем возвращать управление TeX'у.

10.6.2 Применение шрифтов Multiple Master в этой книге[1]

При верстке этой книги мы пытались следовать стандартному набору начертаний и насыщенностей в LaTeX'овской схеме выбора шрифтов (Goossens et al., 1994, с. 220) с тем, чтобы построить соответствующие варианты шрифтов Minion и Myriad и назвать их в соответствии со схемой *Fontname*.

Шрифт Minion имеет три оси: насыщенность, которая изменяется от 345 до 620, ширину, которая изменяется от 450 до 600, и размер, который меняется от 6 до 72. В шрифте Myriad только две оси: насыщенность, которая меняется от 215 до 830, и ширина, которая меняется от 300 до 750. Если взять диапазон насыщенностей *Fontname* как стартовую точку и просто разбить его на равные интервалы, то результат получится неестественно жирным, поскольку точка «medium», которую мы используем чаще всего, лежит не в середине диапазона. Аналогично, простое приписывание ширин приводит к слишком приплюснутой форме букв, используемых по умолчанию. Ясно, что значения параметров должны выбираться в процессе тщательного дизайна. В табл. 10.8 в качестве рекомендации мы привели ряд значений параметров для шрифтов Minion и Myriad, который был использован в этой книге. Соответствующие шрифты показаны в табл. 10.9.

Таблица 10.8. Соответствие параметров шрифтов Multiple Master схеме *Fontname* и LaTeX'овским кодам NFSS.

Код *Fontname*	Насыщенность LaTeX	Параметр Minion	Параметр Myriad	Имя *Fontname*
h	ul	345	215	Hairline
j	el	360	280	Extralight
l	l	380	360	Light
k	sl	400	400	Book
m	m	415	443	Medium
s	sb	500	540	Semibold
b	b	580	640	Bold
x	eb	600	740	ExtraBold
u	ub	620	830	UltraBlack
o	uc	450	300	UltraCondensed
q	ec	468	350	ExtraCondensed
c	c	490	400	Condensed
n	sc	510	450	Narrow
r	m	535	500	Regular
x	sx	550	550	Extended
e	x	565	600	Expanded
v	ex	584	650	Extra Expanded
w	ux	600	700	Wide

[1] В данном разделе речь идет об английском оригинале книги. — *Прим. ред.*

Таблица 10.9. Multiple Master-шрифты Minion, отвечающие комбинациям *Fontname*.

	UltraCondensed	ExtraCondensed	Condensed	Narrow	Regular	Extended	Expanded	Wide
Extra Light	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
Light	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
Book	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
Medium	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
Semibold	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
Bold	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
ExtraBold	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn
UltraBlack	Majn	Majn	Majn	Majn	Majn	Majn	Majn	Majn

Важно иметь в виду, что TEX имеет фиксированный объем памяти, выделенный для шрифтов, и что каждый из размеров одного и того же шрифта занимает один слот. Варианты шрифта Multiple Master, созданные для каждого из обычно используемых в LATEX'е размеров (5, 6, 7, 8, 9, 10, 11, 12, 14, 17, 20 и 25pt), занимают не больше места, чем шрифты всех размеров, полученные масштабированием шрифта 10pt. Однако более важно то, что для каждого размера текстового шрифта LATEX может загружать соответствующие математические шрифты![1] В зависимости от того, как установлены математические шрифты (например, декларирован ли алфавит «math typewriter» или, как обычно, загружается другое семейство для символов AMS), это может означать, что переключение на новый размер текста может стимулировать загрузку до 37 новых шрифтов. Вдобавок, шрифты занимают не один и тот же объем памяти; множество лигатур и кернов приводит к существенному увеличению необходимого им объема. Поскольку мы строим наши шрифты с 256 символами в кодировке T1 при помощи fontinst, мы используем очень большие таблицы лигатур. Поэтому можно легко выйти за границы TEX'а, и при наборе книги (в которой, очевидно, используется много разнообразных шрифтов) нам нужно принять особые меры, чтобы быть в состоянии обработать их все за один проход TEX'а. Иногда нам требовалось собрать отдельную копию TEX'а с бо́льшим (по сравнению с обычным) объемом памяти, зарезервированным для шрифтов (размер пула памяти в большинстве реализаций TEX'а фиксирован и может быть увеличен только до определенного предела).

[1] LATEX делает это всякий раз, когда он сталкивается с математической формулой; к сожалению, многие макро и окружения (например, `tabular`) используют математический режим внутри себя и поэтому стимулируют загрузку математических шрифтов.

Глава 11

Драйверы и утилиты PostScript'а

Бóльшая часть этой книги так или иначе имеет дело с использованием языка PostScript, идет ли речь о наборе текста, подготовке рисунков или включении цвета. В этой главе мы взглянем на эти задачи «сверху» и обсудим, как сгенерировать PostScript-файл из dvi-файла, как манипулировать полученным файлом, как его просматривать или модифицировать. Среди огромного числа пакетов для генерации, манипуляции и просмотра PostScript-файлов мы сосредоточимся в первую очередь на общедоступных (public-domain) программах, которые могут быть получены большинством пользователей LaTeX'а. Особенно подробно будут рассмотрены драйвер dvips, пакет для манипуляции файлами psutils, программа просмотра Ghostscript и программы, которые создают TeX'овские растровые шрифты pk из PostScript-шрифтов.

Чтобы показать, как некоторые из описываемых средств могут использоваться совместно, в конце главы мы рассмотрим два примера, иллюстрирующие подход к решению общих задач. Это — преобразование фрагментов LaTeX'овского выходного файла в растровую форму для представления в Сети и добавление текста в помеченные места ранее созданных PostScript'овских рисунков средствами LaTeX'а.

11.1 Драйверы dvi

Мы уже подчеркивали, что, если это возможно, вам следует работать со стандартными пакетами графики и цвета, и тогда в определенной степени неважно, какой dvi-драйвер вы используете. В табл. 2.1 на с. 49 перечислены функции, поддерживаемые различными драйверами, и эта таблица может служить хорошей отправной точкой для выбора системы и драйвера, которыми следует пользоваться.

Почти все реализации TeX'а включают драйвер, преобразующий dvi-файл в формат PostScript; несколько известных коммерческих и общедоступных (public-domain) программ рассмотрены ниже.

- Драйвер в Textures для Macintosh поддерживает виртуальные шрифты, прямые вставки на PostScript'е и все, что функционально необходимо для LaTeX'овского пакета графики. Textures с особой тщательностью выполняет функции высококачественного отображения документов в процессе их подготовки, точно следуя предложенным фирмой Adobe соглашениям *Document Structuring Conventions* (см. Appendix G в Adobe Systems (1990)) и требованиям работы с цветом.
- Драйверы TeX'а компании Y&Y (согласованная пара dvipsone и визуализатора для Windows dviwindo) уникальны в том отношении, что используют Adobe Type Manager (ATM) для вывода на дисплей и поддерживают *только* PostScript-шрифты (т. е. не поддерживают файлы pk, которые генерирует METAFONT, хотя и включает утилиту для преобразования этих файлов в PostScript'овские шрифты Type 3). Как и драйвер в Textures, эти драйверы тщательно поддерживаются, чтобы обеспечить максимальную совместимость со всеми PostScript'овскими устройствами и интеграцию с другими коммерческими продуктами, включая ATM (для работы, например, со шрифтами Multiple Master). Эти драйверы хорошо совместимы с программами перекодировки шрифтов, однако *не поддерживают* виртуальных шрифтов. Программа dvipsone является единственной коммерческой программой, поддерживающей частичную загрузку PostScript'овских шрифтов Type 1, что позволяет включать в выходной файл только то подмножество символов, которое используется в документе. Такая возможность в настоящее время имеется и в общедоступной программе dvips, однако dvipsone остается лучшей реализацией.
- Драйверы в выпущенной Майклом Вулисом версии TeX'а VTeX (Vulis, 1996) интересны прежде всего в том отношении, что они поддерживают способность VTeX'а манипулировать шрифтами внутри TeX'а.
- Бесплатный (shareware) пакет TeX'а для Macintosh OzTeX (автор — Эндрю Треворов) включает мощный драйвер PostScript'а, который сортирует текст страницы по шрифтам. Это позволяет ему один раз загрузить шрифт, набрать им те части текста, где он используется, а затем перейти к следующему шрифту, при этом постоянно не переключаясь между шрифтами. Недостатком этого подхода является то, что добавление чистого PostScript'овского кода в выходной файл (например, для включения и выключения цвета) становится практически невозможным. В дистрибутив OzTeX также включен и вариант программы dvips.
- Unix'овские драйверы, преобразующие dvi-файл в PostScript-файл восходят к истокам PostScript'а и драйвер dvi2ps (созданный многими руками) иногда все еще используется, равно как и программа dvitops Джеймса Кларка, которая в конце 80-х годов сыграла революционную роль благодаря возможности работать на разных платформах, возможностям перекодировки и поддержке включения файлов. К сожалению, она никогда не модифицировалась с целью, например, поддержки виртуальных шрифтов.

- Наиболее широко используемым драйвером `dvi` в настоящее время, несомненно, является dvips Тома Рокицки. Он доступен практически для всех компьютерных платформ (включая DOS, Macintosh, Unix, VMS и VM/CMS) и является основной программой, на которой тестируются все нововведения (Дональд Кнут сам добавил в него первую поддержку виртуальных шрифтов). Драйвер dvips является стандартом, по отношению к которому оцениваются другие драйверы.

У нас нет возможности дать оценку здесь всем известным драйверам и мы решили сосредоточить внимание только на одной программе — dvips. Мы надеемся, что обсуждение ее особенностей позволит читателям конструктивно взглянуть на возможности используемых ими драйверов. Существует две жизненно важные характеристики драйвера, связанные с обработкой PostScript'а:

- *«Мобильность»* выходного файла. Если файл удовлетворяет стандартам для Encapsulated PostScript, то вы можете использовать огромное число программ для последующей обработки и включения полученных файлов в другие приложения.
- *Возможность включения чистого кода PostScript* в выходной файл. Эта лазейка имеет ключевое значение для работы таких полезных утилит, как весь пакет PSTricks (рассмотренный нами в гл. 4), и для обеспечения возможности включения команды `pdfmark` для последующей ее интерпретации программой Acrobat Distiller (см. Haralambous and Rahtz, 1995).

11.2 PostScript'овский драйвер dvips

Программа dvips разрабатывалась многие годы и, следовательно, является зрелым продуктом с множеством важных и полезных функций. Она создает стандартный PostScript, соответствующий соглашениям Adobe Document Structuring Conventions практически во всех отношениях.[1] Программа специально заботится о том, чтобы создаваемый код PostScript занимал в памяти принтера как можно меньше места с тем, чтобы сложные документы с большим числом шрифтов могли распечатываться даже на таких старых устройствах, как Apple LaserWriter. (Сложные документы автоматически разрезаются на части и память принтера очищается между загрузкой этих частей.)

Программа dvips характеризуется мощной поддержкой конструкций типа `\special`. Это — единственный драйвер, который поддерживает все соглашения METAPOST'а, emTEX'а и программы tpic и задает современный стандарт поддержки цвета (хотя в этом направлении еще есть куда развиваться, см. Rokicki (1994)).

[1] В настоящее время dvips некорректно идентифицирует шрифтовые ресурсы, которые ей требуются или она включает, а также не создает столь же много структурированных комментариев, как это делает Textures.

Программа dvips первая ввела две очень полезные функции, представляющие общий интерес для драйверов. Это:

- Автоматическая генерация отсутствующих шрифтов. Если программа dvips не находит каких-либо шрифтов, она временно останавливается и вызывает другую программу, чтобы построить эти шрифты. Эта функция конфигурируема и не ограничивается только запуском METAFONT'а.
- Именованные конфигурационные файлы. Почти все параметры настройки dvips могут быть указаны в конфигурационных файлах, имена которых передаются программе в командной строке. Это позволяет одной программе работать с несколькими устройствами, имеющими различные характеристики.

Возможность автоматической генерации шрифтов сильно зависит от используемой операционной системы. В системе emTEX, работающей под DOS, например, для генерации шрифтов может быть использована emTEX'овская программа dvidrv; под Unix обычно вызывается скрипт MakeTeXPK, который может довольно сильно различаться в разных реализациях TEX'а.

11.2.1 Командная строка и опции конфигурационного файла

Работой dvips можно управлять двумя способами: с помощью ключей, указываемых в командной строке для каждого конкретного задания, или с помощью команд, записываемых в один или несколько конфигурационных файлов. Используя конфигурационные файлы, вы можете установить параметры глобально (для всей системы), ориентируясь либо на конкретные принтеры, либо на конкретных пользователей.

Когда программа dvips запускается, она ищет глобальный файл `config.ps`, который должен существовать (пути, в которых ищутся конфигурационные файлы, зависят от того, как эта программа была скомпилирована). После того как этот основной файл загружен, Unix'овская версия dvips пытается загрузить конфигурационный файл с именем `.dvipsrc` из домашней директории пользователя, а версии для других операционных систем ищут файл с именем `dvips.ini`. После этого читается командная строка. Если при запуске в ней был указан ключ `-P`, то в этот момент читается указанный конфигурационный файл. Каждый конфигурационный файл (ключей `-P` может быть несколько) может изменять любые установки в глобальном или пользовательском конфигурационных файлах, а также изменять все, что было указано в командной строке до того момента, когда была считана текущая опция `-P`.

Если вся командная строка просканирована и в ней не найдено ни одного ключа `-P`, а также не найдено ключей `-o` и `-f`, то ищется переменная окружения `PRINTER`. Если эта переменная определена и для соответствующего принтера существует конфигурационный файл, в конце концов загружается этот конфигурационный файл. Поскольку в путях к конфигурационным файлам обычно указывается и текущая директория (пути могут включать также и другие директории пользователя), то пользователи могут иметь свои версии

конфигурационных файлов для каждого принтера, чтобы вносить изменения в глобальные установки.

Число опций конфигурации и ключей в командной строке dvips достаточно велико, и практически все настройки могут быть осуществлены одним из двух указанных способов. В последующем перечне мы проклассифицировали их по достигаемому результату и указали ключ, используемый в командной строке, и/или букву, указываемую в конфигурационном файле. Большинство опций, не требующих задания параметров, могут быть выключены, если сразу вслед за опцией указать нуль. Например, чтобы запретить печать в обратном порядке (если она разрешена по умолчанию), надо использовать `-r0` в командной строке или `r0` в конфигурационном файле. Вот список опций, которые можно выключить таким способом: `-a`, `-f`, `-j`, `-k`, `-i`, `-m`, `-q`, `-r`, `-s`, `-E`, `-F`, `-K`, `-M`, `-N`, `-U` и `-Z`.

В конфигурационном файле любая пустая строка или строка, начинающаяся с пробела, звездочки, знаков равенства или фунта,[1] игнорируется. Все остальные строки должны иметь вид: управляющая буква, пробел, значение параметра.

В приведенных ниже таблицах каждая строка показывает ключ в командной строке, опцию в конфигурационном файле и тип ожидаемого аргумента (если он требуется). Заметим, что этот набор опций предназначен для программы dvips версии 5.60 и выше; вам следует убедиться, что вы работаете с самой последней версией.

Выбор страниц

`-p ` *number* *Начать* печать со страницы, равной *number*. По умолчанию печать начинается с первой страницы документа. Если перед числом *number* стоит знак =, то он интерпретируется как абсолютное значение номера, а не номер страницы в LaTeX'е (технически, значение `\count0`). Таким образом, в примере `-p =9` печать начнется с девятой страницы документа независимо от того, какой номер страницы будет указан в распечатке.

`-l ` *number* *Завершить* печать после страницы с номером *number*. По умолчанию используется последняя страница документа. Знак = перед числом означает, что указанное число относится к физическому числу страниц, а не к LaTeX'овским номерам страниц.

`-pp ` *numbers* Напечатать страницы с номерами в указанном *интервале*, который обозначается списком номеров *numbers*, разделенных запятыми. Этот список может состоять из номеров отдельных страниц или диапазонов страниц (диапазон обозначается двумя числами, разделенными двоеточием). Так, чтобы напечатать страницы с 3 по 10, 21 и с 73 по 92, надо указать `-pp 3:10,21,73:92`. Заметим, что с помощью этой опции вы не можете заставить принтер печатать страницы в обратном порядке: так, если вы укажете `-pp 3,2,1`, то страницы 3, 2 и 1 будут напечатаны в обычном порядке.

[1] А также знака процента. — *Прим. перев.*

`-n` *number* Напечатать число страниц, равное *number*. По умолчанию число страниц равно 100000. Иногда использовать этот ключ удобнее, чем ключ `-l`. Например, чтобы напечатать 20 страниц начиная со страницы 53, можно указать `-p 53 -n 20`.

`-A` Печатать только *нечетные номера* страниц.

`-B` Печатать только *четные номера* страниц.

`-r` Печатать страницы в *обратном порядке*. По умолчанию страница 1 печатается первой. Эта опция полезна для таких принтеров, как HP Deskjets, которые складывают напечатанные страницы «лицом вверх», так что страница, напечатанная первой, оказывается в самом низу пачки.

Управление выводом

`-o` *name* Направить выходной PostScript'овский код в файл с именем *name*. Если имя файла не указано, то для `dvi`-файла с именем *file*`.dvi` выходному файлу присваивается имя *file*`.ps`. Если первый символ в *name* — восклицательный знак[1], то следующие за ним литеры интерпретируются как команда, которой передается выходной поток данных. Так, если вы указали `-o !lpr`, выходной поток передается стандартной команде печати Unix (в некоторых операционных системах, таких, как DOS и Macintosh, этого сделать нельзя). Опция `-o` автоматически запрещает использование переменной окружения `PRINTER` и отключает автоматическое добавление символа control-D (если оно было включено опцией `-F` или предусмотрено в конфигурационном файле). Используйте `-F` *после* опции `-o`, если вы хотите, чтобы этого не происходило.

`-f` Перенаправить PostScript'овский код на *стандартное устройство вывода*. Заметим, между прочим, что `dvips` требует, чтобы `dvi`-файл допускал возможность многократного чтения, т. е. входной поток для `dvips` не может поступать через канал (характерно только для Unix).

`-m` Установить *ручную подачу бумаги* для принтера.

`-t` *papertype* Установить *формат бумаги*, соответствующий *papertype*. Значение *papertype* должно быть определено в одном из конфигурационных файлов вместе с соответствующим кодом для его выбора. Кроме того, вы можете указать `-t landscape` для того, чтобы повернуть документ на 90 градусов. Чтобы повернуть документ, формат которого не является «letter», вам следует указать опцию `-t` дважды: один раз для формата бумаги и еще раз для `landscape`. Верхний левый угол каждой страницы в `dvi`-файле располагается на расстоянии одного дюйма слева и одного дюйма сверху от соответствующего угла листа. Использование этой опции сильно зависит от конфигурационного файла, поскольку форматы бумаги определяются обычно там.

[1] Или знак |. — *Прим. перев.*

`  ` `@` Эта опция определяет размер бумаги в конфигурационном файле. Она подробно описана в разд. 11.2.2.

`-O` `O` *offset* Сдвинуть *начало отсчета* страницы. Значения *offset* — это разделенная запятой пара чисел x, y типа `.1in,-.3cm`. Величина x указывает на величину дополнительного сдвига страницы вправо, а величина y — на величину сдвига вниз (по умолчанию начало отсчета располагается на расстоянии одного дюйма вниз и одного дюйма вправо от верхнего левого угла листа бумаги).

`-T` `  ` *offset* Установить *размер бумаги* равным указанной паре размеров. Аргументы у этой опции интерпретируются так же, как и в случае `-O`. Эти значения изменяют любые размеры бумаги, которые могут быть указаны с помощью конструкции `\special` в `dvi`-файле.

`-x` `  ` *number* Установить *масштабный множитель*, равный *number*/1000. Опция изменяет любой масштабный множитель, указанный в `dvi`-файле. Величина *number* должна быть между 10 и 100000. Рекомендуется использовать стандартные значения множителей magstep (1095, 1200, 1440, 1728, 2074, 2488, 2986 и т. д.), чтобы ограничить общее число создаваемых `pk`-файлов.

`-D` `D` *number* Установить горизонтальное и вертикальное *разрешение* в dpi (точках на дюйм), равное *number*. Эта опция определяет, какой из растровых шрифтов будет загружен, и также влияет на позиционирование глифов в резидентных PostScript'овских шрифтах. Значение должно лежать между 10 и 10000.

`-X` `X` *number* Установить только *горизонтальное разрешение* (в точках на дюйм), равное *number*.

`-Y` `Y` *number* Установить только *вертикальное разрешение* (в точках на дюйм), равное *number*.

Конфигурация вывода

`-E` `  ` Попытаться *создать Encapsulated PostScript* с правильным размером обрамляющего бокса. Опция может работать только с одностраничными документами (или одностраничными выборками из более длинных документов) и принимает во внимание положение только символов и линий, игнорируя любые графические вставки (обсуждение того, как этим можно воспользоваться, см. на с. 531). Кроме того, поскольку метрика глифов берется из `tfm`-файла, то глифы, выступающие за пределы указанного `tfm`-бокса, могут сбивать программу; бокс также может оказаться несколько «просторным», если размеры левого и правого апрошей у глифа велики. Заметим, что если в документе используются `pk`-шрифты, то вывод `dvips` может оказаться зависящим от разрешения и качество `eps`-файлов будет не очень хорошим, особенно если рисунки предстоит мас-

штабировать. Поэтому используйте такие eps-файлы с большой осторожностью.

`-e` `e` *number* *Разместить каждый символ* так, чтобы его смещение относительно «истинного» положения на странице не превышало *number* пикселов. Значение по умолчанию для этого параметра зависит от разрешения (оно выбирается из списка значений 100, 200, 300, 400, 500, 600, 800, 1000, 1200, 1600, 2000, 2400, 2800, 3200, ... и не должно превышать разрешения в точках на дюйм). Позволяя индивидуальным глифам «дрейфовать» из их правильно округленных положений на несколько пикселов и затем восстанавливая их истинное положение в начале каждого нового слова, можно улучшить расположение букв внутри слов. Однако эта функция может оказаться рискованной при работе с таблицами (например, с пакетом hhline), поскольку сложные линейки не будут выравниваться надлежащим способом. Если у вас появляются проблемы, установите этот параметр в 0.

`-h` `h` *name* Добавить файл *name* в качестве дополнительного заголовочного файла *PostScript*. Если имя файла просто «-», то эта опция *подавляет* добавление всех заголовочных файлов в выходной файл. Содержание заголовочного файла вставляется в PostScript'овский словарь `userdict`.

`-k` Напечатать *границы обрезки бумаги*. Эта опция увеличивает размер бумаги (который указывается либо с помощью конструкции `\special`, относящейся к размеру бумаги, либо с помощью опции `-T`) на полдюйма в каждом направлении. Страница при этом сдвигается на четверть дюйма и в каждом ее углу рисуются границы полей.

Работа с файлами

`-mode` `M` *mode* Указать программе METAFONT параметр *mode* для любого pk-файла, который, возможно, потребуется создать. Если разрешение, отвечающее параметру mode, отличается от указанного опцией `-D` разрешения, оно игнорируется.

`-M` *Отключить автоматическую генерацию шрифтов*. Если какие-нибудь из шрифтов отсутствуют, команды для генерации этих шрифтов добавляются в файл `missfont.log` в текущей директории.

`-V` *Загрузить* нерезидентные PostScript'овские шрифты в растровом формате. Это требует использования таких программ, как ps2pk или gsftopk для генерации необходимых растровых файлов (см. разд. 11.5).

`-Z` `Z` Скомпрессировать растровые файлы перед загрузкой в принтер для уменьшения размера PostScript'овского файла. Эта опция полезна при высоких разрешениях или когда используется большое число шрифтов, однако она замедляет печать, в особенности на старых принтерах.

`-j` `j` *Частичная загрузка* PostScript'овских шрифтов Type 1. Для каждого шрифта создается подмножество шрифта, содержащее только глифы,

используемые в документе[1] Эта функция впервые появилась только в версии 5.60 dvips.

`R` *numbers* Установить список значений разрешения, которые используются по умолчанию (список *numbers*, разделенных пробелами). Эти значения используются при поиске других размеров pk-шрифта, если требуемый шрифт недоступен. При этом в выходном файле средствами PostScript'a производится масштабирование для того, чтобы получить шрифт требуемого размера. В полученном файле эти шрифты обычно выглядят уродливо, поэтому выдается сообщение-предупреждение. Чтобы отключить эту опцию, укажите в конфигурационном файле строку `R` без численных параметров.

Конфигурационный файл и принтер

`-P` *configname* Создать выходной файл для указанной конфигурации или принтера. Для определения конфигурации читается файл `config.`*configname* (*configname*`.cfg` в операционной системе DOS), в котором можно указать большинство описываемых опций. Поскольку `config.ps` читается раньше, чем `config.`*configname*, рекомендуется поместить все стандартные параметры (используемые по умолчанию) в один основной файл `config.ps`, а в конфигурационные файлы записывать только параметры, которые меняются от принтера к принтеру. Если ни одна опция `-P` не задана, то проверяется переменная окружения `PRINTER`. Если такая переменная существует и соответствующий конфигурационный файл найден, то используется последний. В операционной системе Unix указание опции `-P` в отсутствие соответствующего конфигурационного файла интерпретируется как указание направить вывод на системный принтер с именем *configname*. Таким образом, если файл `config.lp1` не существует, опция `-Plp1` направит выходной файл в очередь на системный принтер `lp1`.

`m` *number* Установить значение *number* (по умолчанию — 180000) в качестве объема *памяти принтера*, который может быть выделен для шрифтов. Назначение этой опции — экономия памяти и автоматическое разбиение выходного файла на секции. Размер доступной памяти можно определить, послав на принтер файл следующего содержания:

```
%!
/Times-Roman findfont 30 scalefont setfont 144 432 moveto
vmstatus exch sub 40 string cvs show pop showpage
```

Неплохо указать драйверу dvips, что принтер имеет даже несколько *меньше* памяти, чем число, которое получено этим способом, поскольку память может понадобиться и для других целей. Для типичных документов TEX'a

[1]Заметим, что в dvips эта функция включена по умолчанию. При работе с рядом неправильно сконструированных шрифтов Type 1 при печати документов могут появиться такие дефекты, как печать неправильных глифов, или их выпадание. В этом случае эту функцию следует отключить, указав `-j0`. — *Прим. перев.*

вам надо беспокоиться в том случае, если возвращаемый объем меньше, чем примерно 300000. Если у вас «неограниченная» виртуальная память, используйте очень большое число, скажем миллион![1]

`I` Игнорировать переменную окружения PRINTER.

`p` *name* Указывает имя файла (по умолчанию, `psfonts.map`), в котором указан список доступных PostScript'овских шрифтов. Эта опция позволяет учесть возможное различие наборов резидентных шрифтов в принтерах. Если имя начинается с символа «+», то остальные буквы интерпретируются как имя *дополнительного* `map`-файла, а не имя файла, целиком изменяющего текущие установки.

Управление памятью

`-a` `a` Позволяет *сэкономить память*, обрабатывая `dvi`-файл за три прохода вместо двух (чтобы загрузить только те символы, которые требуются).

`-S` `i` *number* Указать *максимальное число страниц* в каждой «секции». Эта опция обычно применяют вместе с `-i`; при использовании в конфигурационном файле опция `i` эквивалентна `-i -S`.

`-i` Вывести каждую секцию в отдельный файл. В некоторых обстоятельствах dvips разрезает документ на «секции», которые обрабатываются независимо. Эта опция заставляет dvips записывать каждую секцию в отдельный файл, имена которых конструируются путем замещения суффикса выходного файла трехзначным последовательным номером. Эта опция чаще всего используется совместно с опцией `-S`, которая устанавливает максимальную длину секции в страницах. Если, например, ваше устройство не может работать более чем с 10 страницами за один раз в одном задании, вы можете указать `-i -S 10`, чтобы автоматически разрезать выходной файл на 10-страничные секции, которые будут записываться в отдельные файлы.

Разное

`-b` `b` *number* Создать *number* копий каждой страницы путем повторения тела страницы. При добавлении заголовочного файла `bop-hook` эта опция может быть использована для решения задачи цветоделения (см. разд. 9.5 на с. 401).

`-c` *number* Последовательно напечатать *number* копий каждой страницы с использованием PostScript'овской функции `#copies`. По умолчанию создается одна копия. (Для печати «отсортированных» копий документов см. опцию `-C` ниже.)

`-C` *number* Создать *number «отсортированных»* копий документа (путем повторения данных в PostScript'овском файле). Такой файл печатается

[1] Например, при использовании интерпретатора Ghostscript'а. — *Прим. перев.*

дольше, чем при использовании опции `-c`, но это удобнее и проще, чем посылка на печать одного и того же PostScript'овского файла снова и снова.

`-d` | | *number* Включить режим *отладки*. Эта опция используется только в случае крайней необходимости или для детального тестирования экспертами. Она будет работать, если dvips был скомпилирован с опцией `DEBUG`. Выбор значений *number* обсуждается ниже в разд. 11.2.6; для вывода максимально подробной информации используйте значение `-1`.

`-q` | `q` Запретить выдачу сообщений о ходе выполнения программы. По умолчанию dvips выводит номера страниц и номера включаемых файлов; эта опция запрещает выводить что-либо кроме сообщений об ошибках.

`-R` | | Запустить dvips в *безопасном режиме*. Опция запрещает исполнение команд, указанных посредством ‘ в `\special` и с помощью опции `E` в конфигурационных файлах, а также передавать данные в канал (pipe) и открывать любые файлы с абсолютными путями.

| `E` *command* Выполнить системную команду *command* немедленно, пока читается конфигурационный файл. Во многих реализациях эта функция запрещена и приводит к сообщению об ошибке, поскольку эта функция очевидно небезопасна.

| `W` *text* Вывести сообщение *text* на стандартный вывод ошибок. Опция без аргумента подавляет вывод любого указанного ранее сообщения. Эту опцию полезно добавить в основной конфигурационный файл, если вы хотите, чтобы пользователь всегда указывал, например, тип принтера, или если вы хотите информировать пользователя о том, что получаемый PostScript-файл имеет какие-то особые свойства.

Пути

| `H` *path* Искать заголовочные PostScript-файлы в директории *path* (для подключения PostScript'овских шрифтов Type 1).

| `P` *path* Искать растровые шрифты `pk` в директории *path*. Если в аргументе *path* найдены символы `%`, в зависимости от следующей буквы делаются подстановки, указанные в таблице ниже, и поиск для полученного имени файла продолжается.

`%f`	имя файла
`%b`	разрешение устройства вывода по горизонтали (в точках на дюйм)
`%d`	размер шрифта (в точках на дюйм)
`%p`	тип шрифта (это всегда `pk`)
`%m`	режим генерации шрифта METAFONT'ом (задается опцией `M`)

Если в имени пути *действительно* есть символ `%`, должно быть задано полное имя файла (включая путь), а не просто имя директории; например, при поиске файла `cmr10.329pk` и указанном пути типа `/fonts/%b` будет открыта

директория `/fonts/300`, что, по-видимому, не то, что хотелось бы; правильнее будет указать `/fonts/%b/%f.%dpk`.

`S` *path* Искать иллюстрации (такие, как файлы Encapsulated PostScript) в пути *path*.

`T` *path* Искать `tfm`-файлы в пути *path*.

`V` *path* Искать файлы виртуальных шрифтов `vf` в пути *path*.

Вопросы совместимости

`-N N` Отключить *структурированные комментарии*. Это может понадобиться для некоторых систем, неправильно интерпретирующих комментарии PostScript'а, или же для некоторых старых PostScript'овских принтеров и программ.

`-F` Добавить символ Control-D (десятичный код 4) в самый конец PostScript'овского файла. Это может потребоваться, если `dvips` работает в операционной системе типа DOS и общается с PostScript'овским принтером непосредственно (не используя спулинга). Символ Control-D указывает некоторым интерпретаторам PostScript'а, что задание завершено.

`-s s` Заключить выходной файл целиком между двумя PostScript'овскими командами *save* and *restore*. Старайтесь этого избегать, поскольку результат не является истинным `EPS`, однако это может понадобиться, если вы управляете принтером напрямую и вас не очень беспокоят проблемы совместимости выходного файла.

`-K K` Удалить комментарии из включенной PostScript'овской графики, шрифтовых файлов и заголовков. Это иногда необходимо, чтобы обойти ошибки, встречающиеся в спулерах или программах последующей обработки PostScript'а. Например, чаще всего вызывают трудности комментарии `%%Page` внутри `eps`-файлов. Однако использование этой опции может привести к пропаданию части графики, поскольку макро PostScript'овских заголовков в некоторых программах читают входной поток строка за строкой в поисках определенного комментария. По умолчанию эта опция *включена*, поскольку некоторые визуализаторы PostScript'а и спулеры не в ладах с соглашениями о структурировании.

`-U U` Запретить оптимизацию виртуальной памяти PostScript'а, при которой метрическая информация для глифов хранится в той же строке, где хранится его растровое изображение. Это необходимо только при работе с PostScript-интерпретатором Xerox 4045, в котором имеется эта неприятная ошибка.

Вся конфигурация, касающаяся путей поиска, может быть изменена во время исполнения путем установки переменных окружения, которые мы не будем здесь рассматривать.

11.2.2 Размеры бумаги

При создании документов обычно предполагается, что бумага имеет определенный размер и может быть в двух ориентациях — вертикальной (portrait) или горизонтальной (landscape). В идеале, информация о размере бумаги должна находиться в dvi-файле, а не выбираться во время печати, и поэтому dvips поддерживает команду `\special papersize`, хотя размер также может быть указан в конфигурационных файлах или в командной строке. Формат команды `\special` (которая должна присутствовать на первой странице документа[1]) таков:

```
\special{papersize=dimension,dimension}
```

Первый *dimension* соответствует размеру бумаги по горизонтали, а второй — по вертикали. Размеры могут быть даны в любых обычных TEX'овских единицах: *in* (дюймах), *cm* (сантиметрах), *mm* (миллиметрах), *pt* (пунктах), *sp* (приведенных пунктах), *bp* (больших пунктах, иначе говоря, PostScript'овской единице измерения по умолчанию), *pc* (пиках), *dd* (пунктах Дидо) и *cc* (цицеро). Например, для выбора американского стандартного размера бумаги в документе LATEX (или, лучше, в классе документа) следует указать:

```
\special{papersize=8.5in,11in}
```

а формат бумаги A4 в горизонтальной ориентации выбирается так:

```
\special{papersize=297mm,210mm}
```

Разумеется, подобные команды лишь информируют dvips о желательном размере бумаги; вам еще надо указать размеры полосы набора `\textwidth` и `\textheight` (и соответствующие поля) в вашем LATEX'овском классе документа, чтобы действительно использовать указанный размер бумаги.

Когда dvips обрабатывает файл, он пытается удовлетворить запрос на размер бумаги с помощью размеров, предопределенных в конфигурационном файле, после чего решает, что выводить в выходной PostScript'овский файл. Размеры бумаги в конфигурационном файле определяются строками, начинающимися с `@`.

Существуют три формата для строк, начинающихся с `@`. Первый формат — опция, указанная в строке сама по себе (без параметров), — приказывает dvips игнорировать все определенные ранее (например, в другом конфигурационном файле) размеры бумаги и начать с начала. Второй формат — опция с тремя параметрами (ключевым словом и двумя размерами) — интерпретируется как начало нового описания размера, где ключевое слово является именем, а численные параметры — размерами листа бумаги по горизонтали и вертикали. В третьем формате, в котором оба размера являются нулевыми (заметьте, что и в этом случае нужно указывать единицы измерений, например, `0mm`), будет использован размер, указанный в dvi-файле. Если за символом `@` сразу же следует знак `+`, то оставшаяся часть строки (после пропуска пустого пространства) интерпретируется как PostScript'овский код, который посылается на принтер,

[1] В LATEX'е имеется удобный способ сделать это с помощью макро `\AtBeginDvi`.

чтобы выбрать определенный размер бумаги. Если на такой строке со знаком + первым символом является восклицательный знак, то следующая за ним строка помещается в начальную секцию *комментариев* выходного файла, в противном случае она помещается в раздел *установки* выходного файла. Так, описание бумаги формата A4 в стандартном конфигурационном файле имеет следующий вид:

```
@ A4 210mm 297mm
@+ ! %%DocumentPaperSizes: A4
@+ %%BeginPaperSize: A4
@+ a4
@+ %%EndPaperSize
```

Это означает, что структурированный комментарий `%%DocumentPaperSizes: A4` должен быть помещен в раздел начального описания документа, а строки `PaperSize` идут в секцию установки. В этом примере единственная PostScript'овская команда `a4` (определенная в заголовочных файлах dvips) является тем, что инструктирует интерпретатор PostScript'а, однако код может быть произвольным образом усложнен. Если вы уверены, что все ваши принтеры понимают PostScript второго уровня, вы можете использовать такую команду:

```
@+ << /PageSize [ 595 842 ] >> setpagedevice
```

Когда программа dvips встречает формат бумаги в командной строке, она ищет соответствующее *имя*; когда программа встречает команду `\special papersize`, она ищет подходящий размер (который должен совпадать с указанным с точностью до четверти дюйма). Программа будет использовать первый подходящий формат из той последовательности, в которой размеры бумаги перечислены в конфигурационном файле. Если ничего подходящего в списке не находится, то выдается сообщение об ошибке и используется размер бумаги, указанный первым. Поэтому первым всегда должен идти размер, используемый по умолчанию.

Ориентация бумаги landscape для всех размеров поддерживается автоматически; после попытки подогнать пару размеров dvips пытается подогнать их к размерам, взятым в обратном порядке. Если это удается, программа вставляет в выходной PostScript'овский файл код, который поворачивает страницы при печати. Заметим, что этот код отличается от кода, инструктирующего принтер переключиться на другой лоток бумаги.

Например, если в нашем LaTeX'овском файле написано

```
\special{papersize=297mm,210mm}
```

то dvips находит, что размер бумаги 210mm × 297mm есть в конфигурационном файле, но соответствует горизонтальной ориентации landscape. По умолчанию, dvips действует так, как если бы бумага была повернута на 90 градусов против часовой стрелки. Если же вам нужна ориентация landscape, повернутая по часовой стрелке, то в классе документа или самом документе должна быть следующая команда:

```
\special{! /landplus90 true store}
```

(команды \special, подобные этой, обсуждаются в разд. 11.2.3 на следующей странице). Однако если принтер поддерживает ориентацию landscape для формата A4 *напрямую*, то в конфигурационный файл следует добавить дополнительную запись для бумаги размера 297mm × 210mm и указать правильный PostScript'овский код для переключения принтера.

Если ваш принтер имеет команду установки произвольного размера бумаги, тогда укажите нулевые размеры (например, `0in 0in`); PostScript'овская программа, которая устанавливает размер бумаги, может использовать указанные пользователем размеры как `hsize` и `vsize`, которые являются макрокомандами на PostScript'е, возвращающими указанный размер в единицах, принятых PostScript'ом по умолчанию. Например, при выводе на фотонаборный аппарат Linotronic, печатающий на рулоне, вы можете следующим образом определить имя размера страницы, чтобы заставить текст печататься поперек ширины рулона:

```
@ film1 0in 0in
@+ statusdict /setpageparams known
@+ { hsize vsize 0 1 statusdict begin
@+ { setpageparams } stopped end }
@+ { true }
@+ ifelse
@+ { statusdict /setpage known
@+  { hsize vsize 1 statusdict begin
@+  { setpage } stopped pop end } if
@+ }
@+if
```

Чтобы расположить страницы формата A3 длинной стороной вдоль края ленты, этот формат следует вызвать из командной строки с опциями `-t film1 -T 297mm,420mm`. Вышеприведенные установки определены в конфигурационном файле `config.lino`, который распространяется в составе программы dvips.

Заметим, что команды PostScript, определяющие размер бумаги, могут быть зависимыми от устройства, что делает выходной файл dvips менее мобильным.

11.2.3 Поддержка команд \special

Большой набор команд \special, поддерживаемых dvips, можно разделить на четыре группы:

1. указание ресурсов, общая конфигурация и включение PostScript'овских файлов;
2. вставка фрагментов PostScript'овского кода;
3. определение и использование цвета;
4. поддержка соглашений пакетов emTEX, tpic и METAPOST.

Мы *не рекомендуем* вам пользоваться этими командами `\special` непосредственно в документах LaTeX'а. Пользуйтесь вместо этого пакетами высокоуровневых макро, которые могут работать более чем с одним драйвером. В частности, мы очень рекомендуем, чтобы все включения графики, повороты, масштабирование и работа с цветом проводились с применением стандартных LaTeX'овских пакетов graphics и color. Приводимое ниже описание команд предназначено для авторов новых макропакетов и тех, кому просто интересно, как эти пакеты работают.

Включение PostScript'овских файлов и фрагментов PostScript'овского кода

Ниже приведены команды `\special` для включения ресурсов и графики:

Имя	*Пример*	*Замечания*
`papersize`	`\special{papersize=5in,8in}`	устанавливает размер бумаги
`landscape`	`\special{landscape}`	специальный вид `papersize` для выбора горизонтальной ориентации листа стандартного размера
`header`	`\special{header=duplex.pro}`	включить файл PostScript'овского заголовка в выходной файл
`PSfile`	`\special{PSfile="pot.ps"}`	включить графический файл
`psfile`	`\special{psfile=pot.ps}`	включить графический файл (альтернативный синтаксис)

Чтобы иметь возможность использовать некоторые типы графических файлов в выходном файле dvips, а также обеспечить поддержку таких усложненных пакетов, как PSTricks, которые для рисования отдельных элементов включают фрагменты чистого кода PostScript, иногда требуется добавить в выходной файл заголовочный файл PostScript; dvips обеспечивает поддержку этого с помощью `\special`-команды `header`. При включении заголовочных файлов стек словаря находится на уровне `userdict`. Важно представлять, какой объем памяти в принтере может занять заголовочный файл. Заголовочный файл в своих первых 1024 байтах может содержать строку следующего вида:

`%%VMusage:` *minimum maximum*

В этом случае для расчета того, сколько памяти остается для внутренних целей, dvips использует значение *maximum*. Если строки `%%VMusage` нет, то в качестве оценки берется общий размер заголовочного файла в байтах.

В команде `\special{PSfile=...}` для указания размеров рисунка и других параметров используется набор пар *keyword*=*value*, причем пары отделяются друг от друга пробелами. Допустимые ключевые слова приведены в таблице

ниже (все размеры выражены в пунктах PostScript'а, которые эквивалентны TEX'овской единице bp):

Ключевое слово	*Значение по умолчанию*	*Комментарии*
hoffset	0	смещение по горизонтали
voffset	0	смещение по вертикали
hsize	612	размер обрезки по горизонтали
vsize	792	размер обрезки по вертикали
hscale	100	масштаб по горизонтали
vscale	100	масштаб по вертикали
angle	0	угол поворота
clip		обрезать по размеру бокса
llx		координата x нижнего левого угла
lly		координата y нижнего левого угла
urx		координата x верхнего правого угла
ury		координата y верхнего правого угла
rwi		окончательная ширина рисунка
rhi		окончательная высота рисунка

Важно иметь в виду, что для определения размера бокса dvips *не читает* включаемый графический файл; это делает файл LATEX'а, а соответствующие значения передаются через ключи в команде `\special`. В случае стандартного LATEX'овского графического пакета драйвер dvips использует ключи *llx*, *lly*, *urx* и *ury*, чтобы установить размеры обрамляющего бокса рисунка, ключ *clip*, если он требуется, и ключи *rwi* и *rhi*, если рисунок необходимо масштабировать.

Есть два способа прямого включения кода PostScript в выходной файл. В первом, «безопасном» методе код в выходном файле заключается в команды PostScript'а `save` и `restore`, чтобы ограничить их действие. Во втором методе код не защищен и может оказывать влияние на текущее состояние. Последний метод необходим для такой операции, как поворот текста, когда нам *действительно надо* изменить текущее состояние; первый метод можно использовать для того, чтобы очертить границы страницы независимо от основного текста.

«Безопасными» командами `\special` являются:

Имя	*Пример*	*Комментарии*
"	`\special{"newpath 0 0 moveto 100 100 lineto stroke}`	вставить фрагмент PostScript-кода
!	`\special{! /magscale false def}`	вставить фрагмент PostScript-кода в словарь пользователя

Вы можете определить ваши собственные макро для вызова их из добавляемых вами фрагментов PostScript'овского кода; они определяются точно так же, как и сами фрагменты кода за исключением того, что команда `\special` начинается с восклицательного знака вместо кавычек. Эти макро включаются

в раздел заголовка выходного файла в специальный словарь `SDict`, который оказывается первым словарем в стеке словарей PostScript'а при наборе фрагментов PostScript'овского кода.

Незащищенными командами `\special` для включения фрагментов PostScript'овского кода являются:

Имя	*Пример*	*Комментарии*
ps:	`\special{ps: .5 setgray}`	вставить фрагмент PostScript-кода без защиты
ps::	`\special{ps:: 0.5 setgray}`	вставить фрагмент PostScript-кода
ps::[begin]	`\special{ps::[begin]gsave .5 setgray}`	начало вставки PostScript-кода
ps::[end]	`\special{ps::[end]grestore}`	конец вставки PostScript-кода
ps: plotfile	`\special{ps: plotfile foo.ps}`	включить содержимое файла в выходной файл без защиты

Команды `::` могут использоваться для формирования в выходном файле «неразрывных» последовательностей команд PostScript'а, построенных из блоков команд `ps::`, которые начинаются с `ps::[begin]` и заканчиваются `ps::[end]`. Это может быть полезно, когда вам нужно написать большой фрагмент PostScript'овского кода. В общем случае, однако, используйте простые конструкции `ps:` там, где это возможно.

Когда вы добавляете незащищенный PostScript'овский код, вам следует помнить, что dvips работает в своей собственной системе координат, которая зависит от разрешения. Последнее хранится в PostScript'овском макро, называемом `Resolution`.

Цвет

Программа dvips понимает две группы команд `\special`, служащих для работы с цветом. Первая из них устанавливает цвет фона страницы:

`\special{background` *colorname*`}`

Если на одной странице было указано несколько цветов фона, то используется последний из определенных цветов, причем он оказывает влияние на всю страницу. Вы не можете установить фон для части страницы; для решения этой задачи следует использовать `\colorbox`.

Чтобы установить текущий цвет текста, есть три команды `\special`, которые правильно работают, даже если страницы повторяются или следуют в обратном порядке:

`\special{color` *colorname*`}`

Эта команда устанавливает текущий цвет в *colorname*.

`\special{color push` *colorname*`}`

Эта команда сохраняет текущий цвет в специальном стеке и устанавливает новый цвет в *colorname*.

`\special{color pop}`

Эта команда считывает цвет, находящийся на вершине стека, и назначает его текущим цветом.

Имена цветов в командах `\special` могут быть заданы одним из следующих способов:

1. Именем PostScript'овской процедуры, определенной в заголовочном файле. Файл `color.pro`, поставляемый с dvips, определяет в виде процедур 64 имени цветов (они показаны на цветной вклейке XIII).
2. Именем модели цвета (*rgb*, *hsb*, *cmyk* или *gray*), за которым следует соответствующее число параметров, например `color cmyk 0.1 0.2 0.4 0.9`.
3. Конструкцией, состоящей из двойных кавычек, за которыми следует произвольный PostScript'овский код. Этот способ может быть, в частности, использован для того, чтобы получить доступ к PostScript'овским patterns[1], или для того, чтобы задать цвет в системе представления цвета, которую dvips прямо не поддерживает.

Хотя мы можем использовать этот механизм для доступа к PostScript'овским patterns или screens[2], в каждый момент активным может быть только один цвет. Это означает, что мы не можем переключать цвета и patterns независимо.

Поддержка других команд `\special`

Поддержка команд `\special` в emTEX'е и tpic обсуждается в приложениях A.1 на с. 537 (emTEX) и A.2 на с. 538 (tpic), а поддержка в METAPOST'е — в гл. 3.

[1] Речь идет о возможности «закрашивать» выделенную область рисунком из заданных геометрических фигур, которая входит в группу специальных эффектов (Special Color Space) в PostScript'е второго уровня. Аналогичный подход также может использоваться и для градиентной заливки цветом, которая появилась в PostScript'е третьего уровня. — *Прим. перев.*

[2] Halftone screens — механизм, реализованный в PostScript'е, в котором за счет объединения пикселов в ячейки и их последующей обработки можно формировать полутоновые изображения, в том числе и цветные. — *Прим. перев.*

11.2.4 Поддержка шрифтов

Традиционно TEX использует для печати на принтере растровые (pk) шрифты. Однако когда устройство печати является PostScript'овским, мы можем захотеть использовать встроенные в него PostScript'овские шрифты, а также другие PostScript'овские шрифты, которые мы можем загрузить. Для dvips список встроенных шрифтов хранится в файле, по умолчанию называемом psfonts.map. Он состоит из строк, которые отображают имя TEX'овского шрифта в полное имя PostScript'овского шрифта. Так, в простейшем случае эта строка выглядит так:

```
boldtimes Times-Bold
```

которая указывает драйверу dvips отобразить TEX'овское имя boldtimes в полное имя Times-Bold. Когда dvips встречает имя шрифта в dvi-файле, программа просматривает этот список. В более приближенном к реальности случае, когда мы следуем схеме именования шрифтов, описанной в разд. 10.4 на с. 443, наша строка будет выглядеть примерно так:

```
ptmb8r Times-Bold "TeXBase1Encoding ReEncodeFont" <8r.enc
```

Здесь мы используем две дополнительные возможности, предусмотренные в файле отображения (map-файле). Во-первых, любой PostScript'овский код, заключенный в двойные кавычки и следующий за именем шрифта PostScript, исполняется при загрузке шрифта. Строка TeXBase1Encoding ReEncodeFont перекодирует шрифт, чтобы сделать все глифы в нем доступными (команда ReEncodeFont определяется в заголовочном файле, который загружает dvips). Во-вторых, все, что следует за знаком <, интерпретируется как имя файла, содержимое которого должно быть добавлено в выходной файл. В нашем случае 8r.enc определяет кодировку TeXBase1Encoding, о которой мы только что говорили. Перекодировка шрифта обсуждалась в разд. 10.2.3 на с. 426.

Программа dvips поддерживает еще две команды PostScript'а, которые могут использоваться в map-файле так же, как ReEncodeFont: это — ExtendFont и SlantFont. Использование этих команд проиллюстрировано в следующем отрывке из map-файла:

```
purro8x  CentaurMT-Expert " .167 SlantFont "
hlcrt8rn LucidaTypewriter " .850 ExtendFont TeXBase1Encoding ReEncodeFont " <8r.enc
```

Первая строка говорит, что TEX'овский шрифт purro8x получается из PostScript'овского шрифта CentaurMT-Expert путем наклона его вправо на величину .167. Вторая строка указывает, что hlcrt8rn является PostScript'овским шрифтом LucidaTypewriter, сжатым по горизонтали до 85% от его начальной ширины (значение 1.5 сделает его в полтора раза шире по сравнению с начальной шириной) и перекодированным.

Наконец, map-файл программы dvips позволяет указать, чтобы PostScript'овский шрифт был загружен вместе с документом в принтер и после

этого интерпретировался как встроенный. Для этого используется та же самая синтаксическая конструкция (<), как и для файлов кодировки:

```
mzdr MZapfDingbatsPS <mzdr.pfb
```

которая означает, что шрифт `mzdr.pfb` должен быть загружен, чтобы удовлетворить запросам к `MZapfDingbatsPS`.

Все доступные возможности использованы в следующем примере:

```
mbbb8r Bembo-Bold "TeXBase1Encoding ReEncodeFont" <8r.enc <mbbb8a.pfb
```

в котором перекодированная версия `Bembo-Bold` (загружаемая вместе с документом) отображается на TEX'овское имя `mbbb8r`. Отметим, что файлы шрифтов и файлы кодировки загружаются только один раз, хотя map-файлы могут ссылаться на них чаще.

Хотя многие пользователи просто дописывают все строки из сопровождающего шрифт map-файла в используемый по умолчанию файл `psfonts.map`, dvips допускает более тонкую организацию путем использования нескольких конфигурационных и map-файлов. Дополнительные map-файлы могут быть подключены с помощью команды `p` в конфигурационном файле. Так,

```
p new.map
```

указывает, что `new.map` является *единственным* набором строк-отображений, в то время как

```
p +new.map
```

указывает, что содержимое `new.map` должно быть добавлено к текущему списку.

11.2.5 Специальные приемы

Программа dvips предоставляет набор приемов (hooks), позволяющих пользователю вставить код, который необходимо выполнить в нескольких местах. Это делается путем определения четырех PostScript'овских процедур (`start-hook`, `end-hook`, `bop-hook` и `eop-hook`) в PostScript'овском словаре `userdict`, которые исполняются, соответственно, в начале и в конце документа и в начале и в конце каждой страницы. Эти процедуры могут быть *переопределены* в заголовках, добавляемых с помощью опции `-h` или командой `\special{header=...}`. Этот код исполняется вне содержимого страницы, заключенного в операторы `save` или `restore`, так что он может оперировать с информацией о документе в целом (если последний не разбит на секции из-за ограничения памяти). Когда исполняется код этих процедур, то используемые по умолчанию координатная система и начало отсчета в PostScript'е могут также изменяться.

Когда исполняется `bop-hook`, в PostScript'овский стек помещаются два значения — номер страницы TEX'а и порядковый номер страницы в файле. Для `start-hook` в стек помещаются горизонтальный и вертикальный размеры, коэффициент масштабирования, разрешение по горизонтали и вертикали (в точках на дюйм) и имя входного `dvi`-файла. Процедуры должны *оставлять*

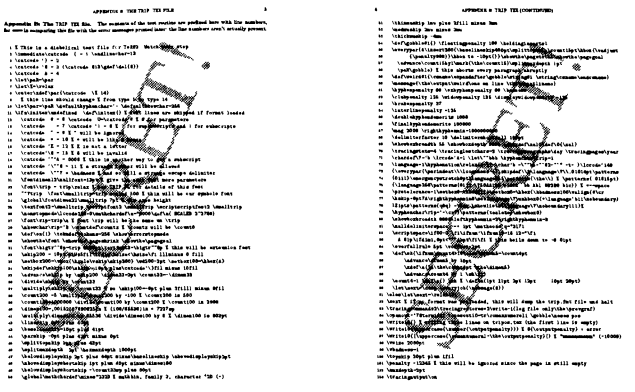


Рис. 11.1. Наложение «штемпеля» поверх текста с помощью dvips.

эти параметры в стеке. Остальные процедуры hook не помещают в стек никаких значений.

Если вы *на самом деле* используете bop-hook или eop-hook, чтобы передавать некую информацию от страницы к странице, то документ не будет более удовлетворять соглашению Adobe Document Structuring Conventions, так что вы должны использовать опцию -N для того, чтобы отключить структурированные комментарии.

Типичными примерами использования процедуры bop-hook является создание надписей, идущих очень большими буквами поперек страницы (поверх текста), или печать границ обрезки на каждой странице. Например, если вы запишете следующие команды в файл с именем secret.pro:

```
userdict begin
/bop-hook{gsave
 200 150 translate
 65 rotate
 /Times-Roman findfont 144 scalefont setfont
 0 0 moveto
 0.7 setgray
 (SECRET!) show
grestore} def end
```

и затем обработаете файл test.dvi с помощью dvips, запустив его с командной строкой

```
dvips -h secret.pro test.dvi -o test.ps
```

то каждая страница будет выглядеть так, как показано на рис. 11.1.

Пакет draftcopy Юргена Фольмера использует этот подход, предоставляя более удобный LaTeX'овский интерфейс для добавления информации поверх страниц. Текст, выводимый по умолчанию, может быть на нескольких языках («Draft» на английском, «Entwurf» на немецком и т. д.), однако пользователь может указать любое другое слово. Этот пакет позволяет работать с несколькими другими dvi-драйверами кроме dvips.

Элегантный прием, предложенный автором пакета dvips, позволяет записывать заголовочный файл динамически из конфигурационного файла, что дает возможность получить доступ, например, к текущей дате. Если поместить следующие команды в файл с именем `config.dte`:

```
E echo /bop-hook userdict begin \{ gsave  > date.txt
E echo  /Times-Roman findfont 10 scalefont setfont  >> date.txt
E echo   10 10 moveto \(printed on `date`\) show grestore  >> date.txt
E echo   \} def end >>  date.txt
h date.txt
```

и затем выполнить

```
dvips -Pdte test.dvi -o test.ps
```

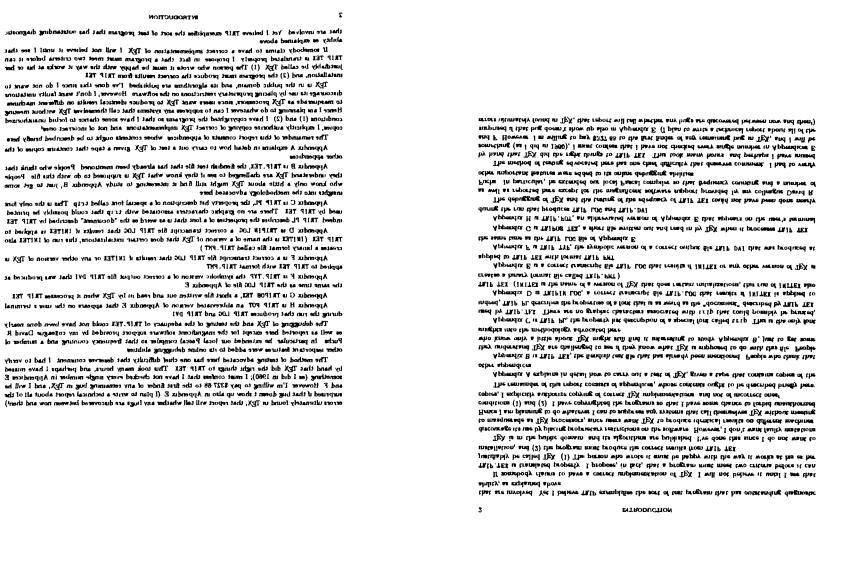
то внизу каждой страницы будет напечатана дата обработки документа программой dvips. Этот прием работает только в системе Unix (программа должна быть способна сама запускать на исполнение команды типа `date`) и не будет работать, если поддержка опции E была запрещена при компиляции dvips (эта опция представляет собой дыру в защите).

Альтернативным способом сделать то же самое может быть запись материала в PostScript'овский заголовок, используя технику `\special{!...}`. При этом предыдущий пример может быть переписан следующим образом:

```
\special{!
 userdict begin
 /bop-hook {gsave
 /Times-Roman findfont 10 scalefont setfont
 10 10 moveto (typeset on \today) show grestore
 } def end
}
```

При этом, разумеется, будет напечатана дата, когда документ был создан, а не дата, когда он был обработан программой dvips.

Другим применением рассматриваемых приемов является подготовка документов к печати. При использовании LaTeX'а и драйвера dvips для подготовки пленок для печати непосредственно на принтере вам может понадобиться напечатать текст «лицом вниз», как бы в зеркальном отражении. Следующие команды в заголовочном файле (частное сообщение Хеннинга Хейнца) позволяют достигнуть этого достаточно простым образом с помощью операторов PostScript'а `translate` и `scale` (заметим, что оператор `translate` в примере



(a) (b)

Рис. 11.2. Зеркальная печать с помощью dvips.

выражен в единицах ширины страницы; для этого dvips предоставляет PostScript'овский макро, называемый `hsize`):

```
userdict begin /mirror {
 hsize 25.4 div 72 mul 0 translate
 -1 1 scale } def
 /bop-hook{ mirror} def
end
```

Полученный результат показан на рис. 11.2(a). Простое изменение позволяет получить отражение вокруг горизонтальной оси (рис. 11.2(b). В PostScript'е второго уровня этот эффект отражения и эффект «белое-на-черном», который также бывает нужен для печати, прямо поддерживаются командой `setpagedevice` (см. Adobe Systems, 1990, с. 237). Это значит, что вам просто нужно включить следующий код в преамбулу PostScript'овского документа:

```
<< /MirrorPrint true >> setpagedevice
```

чтобы получить отражение и

```
<< /NegativePrint true >> setpagedevice
```

чтобы получить белое на черном. К сожалению, не все реализации PostScript'а поддерживают это (Ghostscript, в частности, этого не делает). Однако высококачественные фотонаборные аппараты, для которых эти функции важны для печати на пленку, чаще всего поддерживают их.

В разд. 9.5.2 на с. 404 был описан процесс цветоделения, в котором также использовалась процедура `bop-hook`. Следует, однако, предупредить пользователей описываемых приемов, что dvips не позволяет производить проверку содержимого аккумулирующих команд. Если вы хотите написать SECRET! на каждой странице и *одновременно* напечатать границы обрезки, вы не можете просто загрузить два заголовочных файла друг за другом, но должны написать вместо этого более сложный PostScript-код. Файл `bop-hax.tex` Богуслава Яцковского из коллекции полезных приемов TeX-PS (см. архив CTAN, `macros/generic`) показывает, как это можно сделать.

11.2.6 Отладка

Опция `-d` в dvips служит для отслеживания ошибок и анализа того, что происходит. После ключа вы должны указать целое число, которое указывает на тип информации, который будет выводиться. Чтобы получить информацию нескольких типов, просто сложите вместе числа, отвечающие тем типам данных, которые вас интересуют. Вот эти числа:

1	команды `\special`	4	шрифты	16	заголовки	64	файлы
2	пути	8	страницы	32	сжатие шрифтов	128	память

Например, вызов dvips с опцией `-d 4` даст примерно следующее:

```
This is dvipsk 5.58f Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.03.25:2323' -> mirror2.ps
Defining font () cmr7 at 17.4pt
Defining font () cmr10 at 10.0pt
Defining font () cmsl10 at 10.0pt
Defining font () cmr8 at 8.0pt
Defining font () cmbx10 at 10.0pt
Defining font () cmtt10 at 10.0pt
Font cmr10 <cmr10> is resident.
Font cmr8 <cmr8> is resident.
Font cmtt10 <cmtt10> is resident.
<texc.pro><texps.pro>. <cmtt10.pfb><cmr8.pfb><cmr10.pfb>[2]
```

Это показывает, какие шрифты были вызваны и как они были расположены (в этом примере в выходной файл были включены, а затем загружены в принтер PostScript'овские версии шрифтов CM).

11.3 Средства манипулирования PostScript'овскими страницами

Очень часто оказывается полезным иметь возможность манипулировать целыми PostScript'овскими страницами, такими, какими они получаются с помощью программ, подобных dvips. Одним из оснований для принятия соглашения Adobe Document Structuring Conventions как раз и явилось требование иметь достаточно информации для того, чтобы это было возможно. Например,

вы можете захотеть поменять страницы местами, изменить их масштаб, слить несколько файлов вместе и т. п. В следующем разделе мы рассмотрим набор полезных программ, которые осуществляют бóльшую часть таких преобразований. Перед этим, однако, кратко рассмотрим некоторые другие возможности объединения нескольких логических страниц для печати на одном листе бумаги.

Многим людям хотелось бы просто уметь распечатать две обычные страницы, сжатые так, чтобы уместить их на одном листе бумаги (либо с целью экономии бумаги, либо чтобы собрать буклет). В LaTeX'е это можно сделать целиком в рамках самого TeX'а, по-умному переопределив примитив `\shipout` и масштаб (magnification). Пакет `2up` Тимоти ван Зандта, входящий в пакет `seminar` (который обсуждается в разд. 9.4 на с. 392), использует этот подход для того, чтобы распечатывать два слайда на одной странице. Его большим преимуществом является то, что он не требует никакого специального `dvi`-драйвера и правильно масштабирует шрифты (но требует, чтобы пользователь создал новый набор растровых шрифтов для сжатых страниц или работал с векторными шрифтами). Вы можете указывать ширину и высоту исходной и конечной страниц и изменять другие опции; единственным требованием к драйверу при стандартном использовании является то, чтобы могла быть установлена горизонтальная ориентация для печати двух сжатых страниц, расположенных рядом друг с другом.

Другим простым способом напечатать несколько страниц на одном листе бумаги (если вы пользуетесь PostScript'овским `dvi`-драйвером) является включение заголовочного PostScript-файла, переопределяющего низкоуровневый PostScript'овский оператор `showpage`. (Здесь, безусловно, существенно, чтобы вы пользовались векторными PostScript'овскими шрифтами, а не растровыми шрифтами `pk`, иначе при масштабировании изображения глифов резко ухудшатся.) Одним из таких заголовочных файлов является файл `multi.pro`, написанный Россом Картлиджем; если вы загружаете его как заголовочный файл, вам надо поместить небольшой стартовый PostScript'овский код в преамбулу и некий завершающий код в конце страниц. Для этой цели могут быть использованы процедуры `start-hook` и `end-hook`, описанные в предыдущем разделе. Завершающим PostScript'овским кодом всегда является `endmulti`; стартовым кодом является команда `multi`, которой (в любом месте) предшествует от 3 до 7 параметров. Обязательными параметрами являются:

landscape делит страницу в горизонтальной или вертикальной ориентации. Возможными значениями являются `true` (горизонтальная ориентация) или `false` (вертикальная ориентация);

rows число строк, составленных из страниц;

columns число столбцов, составленных из страниц.

Другими параметрами являются:

left-to-right определяет, заполняется ли физический лист логическими страницами слева направо (`true`) или справа налево (`false`); по умолчанию используется `true`;

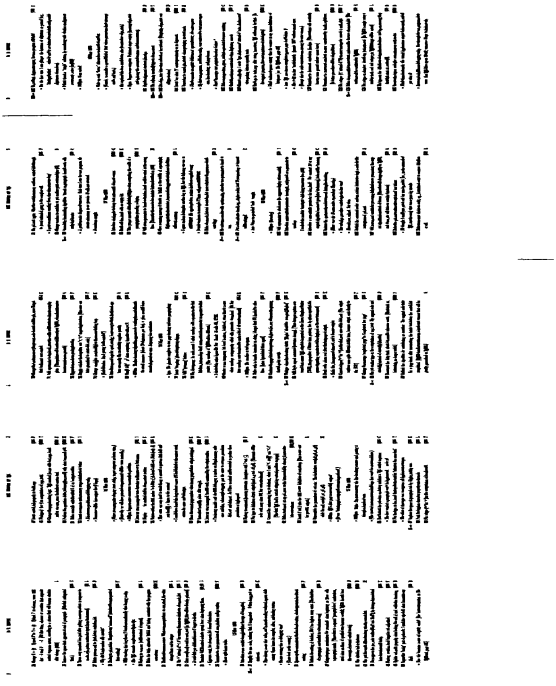


Рис. 11.3. Размещение нескольких логических страниц на одном физическом листе с использованием `multi.pro`.

top-to-bottom определяет, заполняется ли физический лист логическими страницами сверху вниз (`true`) или снизу вверх (`false`); по умолчанию используется `true`;

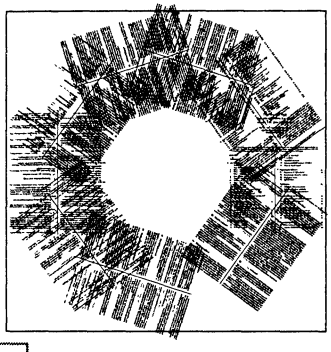
row-first определяет, заполняются ли первыми строки (`true`) или столбцы (`false`);

dividers определяет, следует ли печатать разделительные линейки между логическими страницами; по умолчанию используется `true`.

Таким образом, стартовый код `true 1 5 multi` располагает логические страницы на физическом листе в пять колонок и в одну строку в горизонтальной ориентации (см. рис. 11.3). Размер логических страниц подгоняется под заказанное расположение, т. е. в рассматриваемом примере страницы сжаты с обеих сторон, чтобы поместиться. Более полезное размещение получается при указании `false 4 4 multi`, четырех строк и четырех столбцов в вертикальной ориентации; печать шестнадцати страниц на одном листе оказывается удивительно полезной, когда вы делаете что-нибудь подобное проверке расположения текста в большой книге.

Для совершенно произвольного размещения набора логических страниц на физическом листе вы можете прибегнуть к технике, которую мы использовали в настоящей книге. Чтобы продемонстрировать ее возможности, мы использо-

вали программу dvips для создания eps-файла результата как одной страницы LaTeX'а и включали его обратно в наш текст как рисунок. Это позволяло добиться таких забавных эффектов, как печать крошечных версий первых девяти страниц LaTeX'овского «Frequently Asked Questions», расположенных по окружности (используя PSTricks):



11-3-1

```
\usepackage {pstricks,graphicx}
\newcommand{\Page}[2]{% angle, filename
  \uput[#1]{#1}(0,0){%
   \includegraphics[width=.5in]{#2}}}
\fbox{%
 \begin{pspicture}(-2,-2)(2,2)
  \psset{labelsep=.8cm}
  \Page{360}{p1} \Page{36}{p2}
  \Page{72}{p3}  \Page{108}{p4}
  \Page{144}{p5} \Page{180}{p6}
  \Page{216}{p7} \Page{252}{p8}
  \Page{324}{p9}
 \end{pspicture}%
}
```

11.3.1 Пакет psutils

Достаточно универсальным набором средств для манипулирования PostScript'овскими файлами является свободно распространяемый пакет psutils, созданный Ангусом Даггэном; см. архив CTAN, `support/psutils`. Это — набор скриптов для командного процессора Unix, сценариев на языке Perl и программ на языке C, которые легко компилируются для Unix и других операционных систем[1] Эти программы не являются интерактивными и не имеют графического интерфейса, а управляются с помощью ключей из командной строки. Поскольку программы могут быть достаточно сложными и иметь много опций, мы будем выделять *необязательные* параметры серым фоном: -w ***width***. Размеры, там, где они необходимы, могут задаваться в дюймах (`3in`) или сантиметрах (`10cm`). Многие из этих программ имеют опцию выполняться «молча» (quietly, -q) вместо того, чтобы печатать номера страниц по мере их обработки, а бóльшая часть программ позволяет либо читать со стандартного входа и писать на стандартный выход (выполняя роль фильтра), либо, указав два параметра (имена входного и выходного файлов), работать с ними.

Обсуждаемые программы работают в режиме постраничной обработки. Почти всегда требуется, чтобы исходный PostScript'овский файл содержал достаточно комментариев в соответствии с Adobe Document Structuring Conventions, чтобы позволить программам идентифицировать страницы. Большинство `dvi`-драйверов производят PostScript'овские файлы, удовлетворяю-

[1] Скомпилированные для операционных систем DOS и Windows программы пакета psutils можно найти в дистрибутивах систем emTeX и MikTeX. — *Прим. перев.*

щие этим критериям, однако нельзя считать, что любой PostScript'овский файл подходит для этого. Поэтому в пакет входит набор фильтров для «коррекции» файлов, полученных из различных приложений, которые генерируют нестандартный выходной код PostScript.

В качестве примера использования psutils снова вернемся к задаче размещения нескольких логических страниц на одном физическом листе. Программа psnup является простым и эффективным способом сделать это. На рис. 11.4 показан результат, полученный путем указания в командной строке следующих параметров:

```
psnup -9 -d
```

которые просят разместить девять логических страниц на одном физическом листе и заключить каждую страницу в рамку.

Программа pstops выполняет ту же самую функцию, но позволяет более детально контролировать размещение и масштабирование страниц. Например, чтобы расположить четыре логические страницы формата A4 на одном физическом листе того же формата, следует указать:

```
pstops '4:0@.5(0,14.5cm)+1@.5(10.5cm,14.5cm)' \
'+2@.5(0,0.5)+3@.5(10.5cm,0.5)'
```

В табл. 11.1 на с. 508 дан краткий обзор программ и скриптов, входящих в пакет psutils. Сейчас же мы рассмотрим наиболее важные из них.

pstops: перегруппировка страниц в PostScript'овском файле

```
pstops -q -b -wwidth -hheight -ppaper -dlinewidth
    pagespecs infile outfile
```

Программа pstops перегруппировывает страницы достаточно произвольным образом, позволяя в том числе печатать n страниц на листе, собирать буклеты, печатать в обратном порядке, масштабировать и т. д. Это — наиболее универсальная утилита в пакете, но многие пользователи находят, что проще использовать высокоуровневые программы psnup или psbook. Программа имеет следующие опции:

-b Запрещает работу операторов bind, определенных в прологе PostScript. Это может потребоваться при выполнении сложных многостраничных перегруппировок.

-d*linewidth* Обводит каждую страницу рамкой, нарисованной линией ширины *linewidth*; если численный параметр не задан, то линия имеет ширину в 1 пункт. Важно иметь в виду, что ширина отсчитывается относительно размера *исходной* страницы и также подвергается масштабированию. Поэтому если вы размещаете четыре логические страницы на одном листе и указываете -d 2pt, то ширина линии в действительности будет 1pt.

-h*height* Высота, используемая в качестве спецификатора размера h.

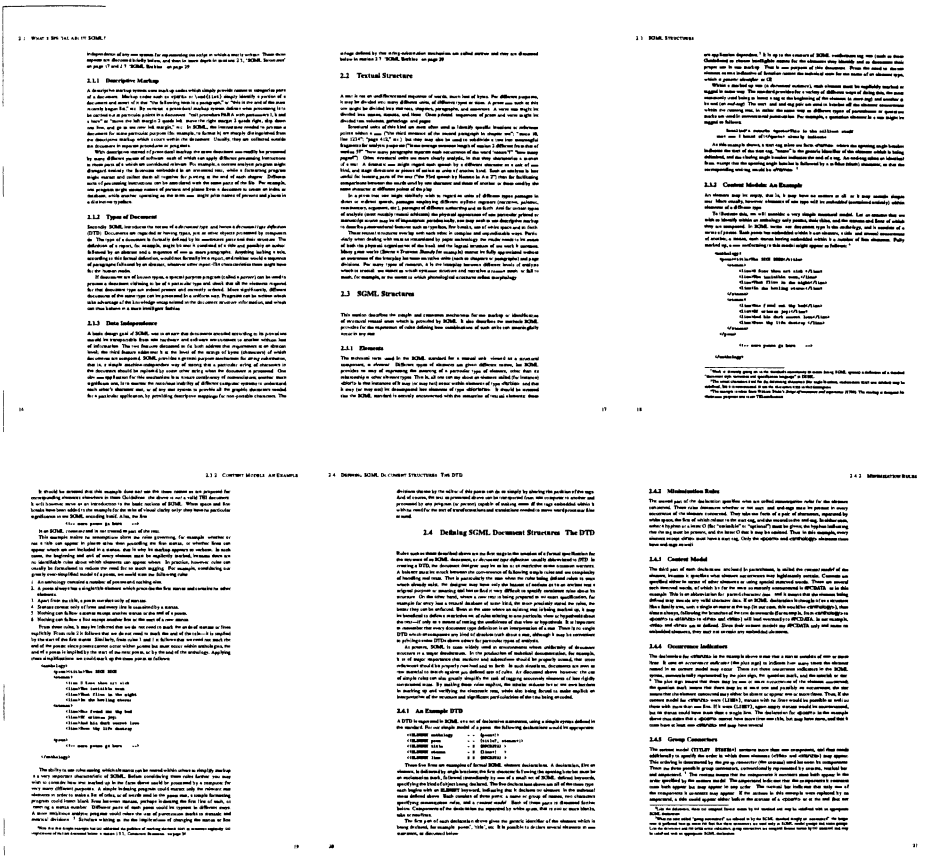


Рис. 11.4. Девять логических страниц, размещенные на одной странице выходного файла.

Таблица 11.1. Обзор утилит, входящих в пакет psutils.

Программы, сценарии для Perl[†] и скрипты для командного процессора[‡]	
psbook	перегруппировывает страницы в тетради (signatures)
psselect	извлекает страницу или диапазон страниц
pstops	общая утилита для перегруппировки и извлечения страниц
psnup	размещает несколько страниц на одном физическом листе
psresize	изменяет размеры листа бумаги в документе
epsffit	подгоняет файл encapsulated PostScript в заданный размер бокса
getafm[‡]	генерирует PostScript'овский файл, который печатает afm-файл для PostScript'овского шрифта
showchar[‡]	генерирует PostScript'овский файл с изображением глифа и его метрикой
fixdlsrps[†]	корректирует выходной файл dvilaser/PS так, чтобы программы psutils работали правильно
fixfmps[†]	корректирует документы FrameMaker
fixmacps[†]	корректирует некоторые ошибки в выходных файлах Macintosh
fixpsditps[†]	корректирует выходной файл Transcript psdit
fixpspps[†]	корректирует выходной файл psprint PostScript
fixscribeps[†]	корректирует выходной файл Scribe PostScript
fixtpps[†]	корректирует выходной файл troff Tpscript
fixwfwps[†]	корректирует выходной файл Microsoft Word for Windows
fixwpps[†]	корректирует выходной файл WordPerfect
fixwwps[†]	корректирует выходной файл Windows Write
extractres[†]	извлекает ресурсы из PostScript'овских файлов
includeres[†]	включает ресурсы в PostScript'овские файлы
psmerge[†]	скрипт для объединения нескольких PostScript-файлов

`-p`*paper* Использовать именованный формат бумаги; возможны следующие имена: `a3`, `a4`, `a5`, `b5`, `letter`, `legal`, `tabloid`, `statement`, `executive`, `folio`, `quarto` или `10x14` (по умолчанию используется бумага формата `a4`).

`-q` Не выводить сообщений о ходе выполнения программы.

`-w`*width* Ширина, используемая в качестве спецификатора размера `w`.

Параметр `pagespecs` позволяет вам указать, как страницы комбинируются и обрабатываются. Он определяется следующим образом:

```
pagespecs = [modulo:]specs
specs     = spec[+specs][,specs]
spec      = [-]pageno[L][R][U][@scale][(xoff,yoff)]
```

`modulo` Число страниц в каждом блоке. Должно быть `>0` (по умолчанию равно 1).

`specs` Спецификации страниц в каждом блоке. Номер страницы `pageno` в каждой `spec` должен быть между 0 (первая страница в блоке) и `modulo-1` (последняя страница в блоке). Необязательные размеры `xoff` и `yoff` сдвигают страницу на указанную величину. По умолчанию эти размеры задаются в PostScript'овских пунктах, однако их можно также указывать в сантиметрах или дюймах (отмечая это окончаниями `cm` или `in`) или (с помощью флагов `w` и `h`) в единицах ширины или высоты, заданной спецификаторами `-h` и `-w`. Необязательные параметры `L`, `R` и `U` позволяют повернуть страницу налево, направо или вверх ногами. Необязательный параметр `scale` масштабирует страницу на указанную величину. Если указан необязательный знак минус, то номер страницы отсчитывается от конца документа вместо его начала.[1]

Страницы в *specs*, помеченные знаком `+`, указывают страницы, которые должны быть объединены на одном листе; страница в *specs*, отделенная пробелом, должна быть размещена на отдельном листе. Если спецификация относится только к одной странице и `pageno=0`, то `pageno` можно опустить.

Сдвиг, поворот и масштабирование производятся в последовательности, которая не зависит от порядка, в котором эти опции указываются в командной строке.

psnup: размещение нескольких страниц на одном листе

```
psnup -wwidth -hheight -ppaper -Wwidth -Hheight -Ppaper
      -l -r -f -c -mmargin -bborder -dlinewidth -sscale
      -nup -q infile outfile
```

Программа `psnup` берет PostScript'овский файл и создает выходной файл, в котором несколько логических страниц размещены на одном физическом листе. Опциями программы являются:

`-b`*border* Вставляет дополнительные поля ширины *border* вокруг каждой логической страницы.

`-c` Упорядочивает страницы на листе бумаги по столбцам, а не по строкам.

`-d`*linewidth* Рисует вокруг каждой страницы рамку линией ширины *linewidth*; если указана только буква `-d`, то ширина устанавливается равной 1 пункту. Важно иметь в виду, что ширина линии отсчитывается относительно размера *исходной* страницы и также масштабируется. Если вы размещаете четыре логические страницы на листе бумаги и указали `-d 2pt`, ширина нарисованной линии в действительности составит 1pt.

[1] Для иллюстрации имеет смысл привести практический пример печати двух соседних страниц формата A5 на листе A4:
`pstops "2:0L(27.2cm,-0.4cm)+1L(27.2cm,13.8cm)" inpfile outfile`
— *Прим. перев*

`-f` Меняет местами высоту и ширину страницы.

`-h`*height* Устанавливает высоту выходных страниц.

`-H`*height* Устанавливает высоту входных страниц.

`-l` Печатает выходной файл в горизонтальной ориентации (повернутым на 90° против часовой стрелки).

`-m`*margin* Оставляет поле шириной *margin* вокруг всей выходной страницы.

`-`*nup* Указывает число логических страниц, которое надо разместить на каждой выходной странице. Это может быть произвольное число, и тогда программа `psnup` сделает все возможное, чтобы найти наиболее подходящее размещение, но может выдать ошибку, если не найдет ничего подходящего.

`-p`*paper* Вместо точного указания размеров выходных страниц эта опция позволяет задавать именованные форматы бумаги. Возможные варианты включают: `a3`, `a4`, `a5`, `b5`, `letter`, `legal`, `tabloid`, `statement`, `executive`, `folio`, `quarto` и `10x14` (по умолчанию принимается формат `a4` для страниц на входе и на выходе).

`-P`*paper* То же самое, что и `-p`, но для страниц на входе.

`-r` Заставляет выходную страницу печататься повернутой на 90° по часовой стрелке.

`-s`*scale* Точно устанавливает параметр *scale* для уменьшения размеров страниц. Это необходимо, когда страницы уже имеют нужный размер и вы хотите просто разместить их по *n* штук на выходных страницах.

`-w`*width* Устанавливает ширину выходных страниц.

`-W`*width* Устанавливает ширину входных страниц.

psbook: сборка страниц в PostScript-файле в тетради (signatures)

```
psbook-q -ssignature infile outfile
```

Программа `psbook` берет страницы из PostScript'овского документа и создает новый файл, страницы в котором перегруппированы как тетради в книге. При печатании книги определенное число страниц размещается на большом листе бумаги, который затем сгибается (образуя тетрадь), а тетради переплетаются, чтобы получился окончательный том. При изготовлении буклета вручную полезно упорядочить страницы так, чтобы, когда лист будет согнут, они следовали в правильном порядке. Программа имеет следующие опции:

`-q` Выполнять «молча», не показывая номеров обработанных страниц.

`-s`*signature* Установить размер сигнатуры (по умолчанию 1), т. е. число страниц (кратное четырем), которые будут сгибаться и переплетаться вместе в конечном издании. При необходимости программа добавляет необходимое число пустых страниц, чтобы получить правильное полное число страниц.

psselect: извлечение страниц из PostScript'овского файла

```
psselect -q -e -o -r -ppages pages infile outfile
```

Программа psselect извлекает подмножество страниц из входного файла и записывает их в выходной файл. Опциями программы являются:

`-e` Извлечь только четные страницы. Эту опцию можно использовать совместно с другими опциями.

`-o` Извлечь только нечетные страницы.

`-p`*pages* Указывает страницы, которые следует извлечь, разделяя диапазоны страниц запятыми. Диапазонами могут быть либо номер отдельной страницы, либо диапазон страниц — два числа, разделенных двоеточием (`:`). В диапазоне можно опустить номера начальной и конечной страниц; в этом случае будут использованы значения по умолчанию — номера первой и последней страниц в файле. Номера страниц, перед которыми стоит знак подчеркивания (_), отсчитываются относительно конца документа, причем счет идет в обратном направлении. Важно иметь в виду, что номера страниц являются абсолютными (физическими), первая страница документа всегда имеет номер 1; номер страницы, который будет на ней *напечатан*, может быть совершенно произвольным.

`-q` Не выводить сообщений о ходе выполнения программы.

`-r` Вывести выбранные страницы в обратном порядке.

psmerge: объединение PostScript'овских файлов

`psmerge -o`*outfile.ps file1.ps file2.ps ...*

Программа psmerge объединяет несколько PostScript'овских файлов в один выходной файл. Сделать это достаточно сложно и удается только в том случае, когда все входные файлы получены одной и той же программой, поскольку спецификации шрифтов и других ресурсов обычно различаются достаточно сильно. Опцией команды является:

`-o`*outfile.ps* Указать имя выходного файла.

psresize: масштабирование и изменение размеров PostScript'овских документов

```
psresize -wwidth -hheight -ppaper -Wwidth -Hheight -Ppaper
        -q infile outfile
```

Программа psresize берет входной PostScript'овский документ, масштабирует и центрирует его, чтобы подогнать под другой размер бумаги. Опциями программы являются:

`-h`*height* Установить высоту выходной страницы.

`-H`*height* Установить высоту входной страницы.

`-p`*paper* Использовать именованный формат бумаги для вывода. Возможные варианты включают: `a3`, `a4`, `a5`, `b5`, `letter`, `legal`, `tabloid`, `statement`, `executive`, `folio`, `quarto` или `10x14` (по умолчанию используется бумага формата `a4`).

`-P`*paper* То же, что и `-p`, но для входной страницы.

`-q` Не выводить сообщений о ходе выполнения программы.

`-w`*width* Установить ширину выходной страницы.

`-W`*width* Установить ширину входной страницы.

Например, если возникла необходимость напечатать на принтере с бумагой формата letter документ, созданный для бумаги формата A4, вы можете использовать следующую команду, чтобы преобразовать файл:

```
psresize -pletter -Pa4 a4file.ps letter.ps
```

Работа с информацией о ресурсах в PostScript'овском файле

Ресурсы являются элементами PostScript'овских файлов, которые присутствуют во многих документах; примерами ресурсов могут служить шрифты и набор PostScript'овских процедур (procsets), а в PostScript'е второго уровня также поддерживаются такие ресурсы, как patterns и forms. Они запрашиваются в прологе PostScript'овского документа как комментарии следующего вида:

```
%%IncludeResource
```

Ресурсы, действительно присутствующие в PostScript'овском файле, должны быть отмечены в комментариях следующего вида:

```
%%DocumentSuppliedResources
```

В настоящее время сценарии на Perl, описанные ниже, не изменяют этих комментариев, так что результат работы этих сценариев не является полностью совершенным.

Программа (сценарий) extractres ищет ресурсы в PostScript'овском документе и записывает их в файлы с тем же именем как имя ресурса и расширением `.ps`. Напротив, программа (сценарий) includeres физически включает указываемые ресурсы в документ PostScript. Ресурсы ищутся в текущей директории и в заданной по умолчанию системной директории по имени ресурса.

Возможным применением этих утилит может быть переписывание документа, содержащего несколько копий ресурсов так, чтобы в нем осталась только одна копия каждого из них. Используя каналы (pipes) в Unix, вы можете сделать это с помощью команды

```
extractres file1.ps | includeres > file2.ps
```

Вы также можете использовать extractres, чтобы уменьшить размер файлов, которые вы посылаете коллегам по e-mail, если вы уверены, что у них имеются те же шрифты и procsets, чтобы включить их, когда они получат ваш файл.

Описываемые программы имеют следующий синтаксис:

```
extractres-m< infile.ps > outfile.ps
includeres   < infile.ps > outfile.ps
```

`-m` Объединить ресурсы с одинаковым именем в одном файле.

epsffit: подгонка eps-файлов под заданный размер

```
epsffit-c -r -a -m -s llx lly urx ury infile outfile
```

Программа epsffit добавляет соответствующий PostScript'овский код в eps-файл, чтобы заставить его уместиться в новом боксе. Размер бокса задается с помощью четырех чисел *llx lly urx ury*, где *llx lly* — x и y-координаты (в пунктах PostScript) левого нижнего угла, а *urx ury* — координаты верхнего правого угла. Опциями команды являются:

`-a` Разрешить изменять пропорции, чтобы вписать объект в бокс (по умолчанию пропорции остаются неизменными).

`-c` Центрировать изображение в боксе.

`-m` Повернуть изображение для того, чтобы увеличить его размер, если при этом оно будет вписываться в бокс лучше.

`-r` Повернуть изображение на 90 градусов против часовой стрелки.

`-s` Добавить команду `showpage` в конце выходного файла, чтобы форсировать вывод изображения.

11.4 Ghostscript — интерпретатор PostScript'a

Программа Ghostscript, выпускаемая компанией Aladdin Enterprises, является общедоступным интерпретатором PostScript'a, написанным Петером Дойчем. Строго говоря, эта программа интерпретирует язык программирования Ghostscript, «подобный» разработанному Adobe Systems языку PostScript, в котором всего лишь несколько особенностей PostScript'a второго уровня реализованы не полностью. Кроме того, он интерпретирует язык Portable Document Format (PDF). Поскольку исходный код Ghostscript'a написан на C и свободно доступен в архивах Internet, эта программа была скомпилирована почти для всех возможных компьютерных платформ. Ghostscript может быть использован для различных целей:

- *Предварительный просмотр PostScript'овского кода.* Ghostscript показывает ваш PostScript'овский файл на экране (поддерживая множество устройств), позволяя быстро проверить ваши рисунки перед тем, как распечатать их на принтере. Для этой цели вы можете предпочесть использовать более удобные интерфейсы, доступные для X Windows System (в Unix) и Microsoft Windows.

- *Создание выходных данных для различных печатных устройств.* Программа Ghostscript имеет большой набор драйверов, поэтому PostScript'овский код может быть преобразован практически в любой формат, понимаемый принтерами.
- *Преобразование в растровые форматы.* Ghostscript может преобразовывать PostScript во множество различных растровых форматов, таких, как TIFF, PCX и PBM.
- *Манипулирование PostScript'ом.* С помощью Ghostscript'а можно делать такие полезные вещи, как извлекать текст из PostScript'овских файлов, вычислять размеры боксов eps-файлов и преобразовывать PostScript в формат Adobe Illustrator.
- *Просмотр PDF-файлов.* Последние версии Ghostscript'а также читают файлы в формате PDF (Portable Document Format), так что вы можете работать с файлами Acrobat на таких компьютерных платформах, для которых бесплатный Acrobat Reader фирмы Adobe пока недоступен. Большинство последних версий Ghostscript'а могут даже «дистиллировать» PostScript'овский код в PDF.

Как запускать и пользоваться Ghostscript'ом, зависит от операционной системы. Вопрос о том, как получить, установить, при необходимости скомпилировать и запустить Ghostscript, лежит вне области интересов нашей книги. В прекрасной книге Томаса Мерца о PostScript'е и PDF (Merz, 1996) имеется подробное приложение, которое должно дать ответ на большинство вопросов; эта книга лежит в свободном доступе в Интернете по адресу `http://www.muc.de/~tm`. В этом разделе мы будем предполагать, что вы работаете в Unix, DOS или VMS с управлением из командной строки; графический интерфейс для пользователей Windows и Macintosh устроен немного по-другому.

Команда

```
gs options files ...
```

читает каждый из указанных файлов и выполняет содержащиеся в них PostScript'овские команды. В зависимости от указанных опций, результат может быть отображен на экране, передан другому процессу или записан в файл. Когда все файлы считаны, команда обычно ожидает дальнейшего ввода с клавиатуры. Чтобы выйти из интерпретатора, надо набрать команду `quit`.

11.4.1 Опции команды Ghostscript и инициализация

Опции программы Ghostscript, вводимые в командной строке, осуществляют три функции: они позволяют вам указать файлы, с которыми надо работать, изменить параметры настройки и задать значения различным PostScript'овским объектам, к которым можно получить доступ с помощью программы или ваших файлов.

Ниже приведен список опций программы:

@filename Считать файл *filename* и интерпретировать его содержимое, как если бы он был написан в командной строке. Опция предназначена для использования в системах типа DOS, в которой невозможны длинные командные строки. Опции или имена файлов в этом файле можно размещать на нескольких строках любым удобным способом.

`--`*filename ...*

`-+` *filename ...* Интерпретировать *filename* как обычно, но взять все последующие аргументы и записать их в PostScript'овский словарь `userdict` как массив строк под именем `ARGUMENTS`.

`-c` *tokens ...* Интерпретировать как код PostScript'a идущие следом аргументы вплоть до следующей опции, которая начинается со знака `-`, за которым идет символ `@` или символ, отличный от цифры. В следующем примере опция `-c` используется, чтобы форсировать исполнение команды PostScript'a `showpage` в конце файла `picture.ps`:

```
gs picture.ps -c showpage
```

Часто командные строки заканчиваются конструкцией `-c quit`, которая заставляет Ghostscript завершить работу после того, как все файлы обработаны.

`-D`*name*`=`*value*

`-d`*name*`=`*value* Присвоить значение *value* имени *name* в словаре `systemdict`. Имя должно быть правильным PostScript'овским токеном. Если никакое значение не указано, оно устанавливается в `true`.

`-f`*filename* Обработать файл, даже если его имя начинается с `-` или `@`. Сама по себе опция `-f` ничего не делает, но может использоваться для завершения набора опций `-c`.

`-g`*n1*`x`*n2* Эквивалентно опциям `-dDEVICEWIDTH=`*n1* и `-dDEVICEHEIGHT=`*n2*. Она позволяет установить ширину и высоту, отвечающие данному устройству вывода.

`-I`*directories* Добавить список директорий в начало списка путей поиска для системных файлов. Имена директорий разделяются знаком `:` в Unix и знаком `;` в DOS и VMS.

`-P` Искать системные файлы сначала в текущей директории (установлено по умолчанию).

`-P-` Искать системные файлы сначала *вне* текущей директории.

`-q` Исполнять «молча», не выдавая стартовых сообщений.

`-r`*number*

`-r`*n1*`x`*n2* Установить разрешение таких выходных устройств, как принтер с изменяемым разрешением или растровый файл. Опция производит такое же действие, как две опции `-dDEVICEXRESOLUTION=`*n1* и `-dDEVICEYRESOLUTION=`*n2*. Вариант опции `-r`*number* используется для

устройств с равным разрешением по обеим осям. Поскольку некоторые печатные устройства имеют разное разрешение по x и y, их можно указать во втором варианте опции.

`-S`*name*`=`*string*

`-s`*name*`=`*string* Присвоить значение *string* имени *name* в словаре `systemdict`. Различие между этой опцией и опцией `-d` заключается в том, что здесь значением является PostScript'овская *строка*, а не токен.

`-u`*name* Сделать имя *name* неопределенным, отменить `-d` или `-s`.

`-` Указать Ghostscript'у, что стандартный ввод программы происходит из файла или канала (pipe). Ghostscript читает из `stdin` (стандартного ввода), пока не дойдет до конца файла, исполняя его как любой другой файл, а затем продолжает обрабатывать командную строку. Дойдя до конца командной строки, Ghostscript завершает работу, а не возвращается к интерактивному вводу. Из-за способа, с помощью которого структурированы файлы PDF, этот ключ может использоваться только с входными файлами PostScript'а.

На практике опции `-d` и `-s` обычно используются для присвоения значений некоторым специальным именам, определенным в Ghostscript'е. Ниже перечислены те наиболее часто используемые из них, которые вам может понадобиться установить. Список имен, однако, не исчерпывается приведенными, поскольку для многих устройств вводятся дополнительные функции.

-dCOLORSCREEN

-dCOLORSCREEN=0

-dCOLORSCREEN=false Использовать отдельные полутоновые экраны на устройствах с разрешением выше 150 точек на дюйм; `-dCOLORSCREEN=0` использует отдельные экраны с одной и той же частотой и углом;[1] `-dCOLORSCREEN=false` заставляет использовать один двоичный экран. Действие по умолчанию — использование раздельных экранов с различными углами, если устройство имеет менее 5 битов на цвет.

-sDEVICE=*device* Установить тип выходного устройства.

-dDISKFONTS Загружать только необходимые глифы из шрифта. Обычно Ghostscript читает шрифт целиком в память, однако когда объем памяти ограничен, может понадобиться эта опция (хотя она и замедляет работу программы).

-dFirstPage=*page* Начать интерпретацию PDF-документа со страницы *page*.

[1] По-видимому, здесь имеет смысл пояснить способ формирования цветного изображения в PostScript'е. Для этого он использует четыре экрана: красный, зеленый, синий и серый для цветовой модели RGB («серый» экран используется для формирования серых полутоновых изображений) и голубой, пурпурный, желтый и черный цвета для цветовой модели CMYK. Цветные пикселы далее объединяются в ячейки, расположенные в пространстве с определенной частотой (в ячейках на дюйм) и ориентированные по отношению к координатной сетке экрана под определенным углом. Выбор разных углов для разных цветов позволяет получить изображение наиболее высокого качества. — *Прим. перев.*

-dFIXEDMEDIA Поправить размеры устройства после инициализации так, чтобы страницы другого размера или ориентации, используемые во входных файлах, автоматически поворачивались или масштабировались.

-dFIXEDRESOLUTION Использовать фиксированное разрешение устройства после инициализации независимо от того, что указано во входном файле. Опция `-r` также устанавливает это.

-sFONTMAP=*filename* Указать имя файла `Fontmap`, который следует загрузить.

-sFONTPATH=directories Указать список директорий, которые будут автоматически сканироваться при поиске шрифтов.

-dLastPage=*page* Остановить интерпретацию PDF-документа после страницы *page*.

-dNOBIND Запретить работу PostScript'овского оператора `bind` (полезно при отладке).

-dNOCACHE Запретить кеширование символов (полезно при отладке).

-dNOCIE Заменить цветовые схемы `DeviceGray` и `DeviceRGB` на `CIEBasedA` и `CIEBasedABC` соответственно. Это ускоряет печать, если не нужна высокая точность передачи цвета.

-dNODISPLAY Не начинать вывод на дисплей. Эта опция используется, когда вы просто хотите увидеть, что программа скажет о вашем входном файле.

-dNOFONTMAP Не загружать файл `Fontmap`.

-dNOGC Запретить использовать «собиратель мусора» в PostScript'е второго уровня.

-dNOPAUSE Запретить выдачу приглашения к вводу и остановку в конце каждой страницы. Эта опция часто указывается при выводе на принтер.

-dNOPLATFONTS Запретить использование шрифтов, поставляемых вместе с используемой операционной системой (например, X Windows System или MS Windows).

-dNOPROMPT Не отображать приглашение к вводу в конце каждой страницы. Эта опция отличается от `NOPAUSE`, поскольку в данном случае программа все равно останавливается, но изображение знака приглашения не накладывается на изображение на некоторых дисплеях.

-sOutputFile=*filename* Задать имя файла для вывода.

-sPAPERSIZE=*papersize* Выбрать один из форматов бумаги, известных программе Ghostscript, например, `-sPAPERSIZE=a4` или `-sPAPERSIZE=legal`. Используемый по умолчанию формат может быть указан в файле инициализации `gs_init.ps`, а допустимые значения имен приведены в файле `gs_statd.ps`.

-dQUIET Работать «молча», не выдавая сообщений.

-dSAFER Запретить работу PostScript'овских операторов `deletefile` и `renamefile` и не разрешать открывать файлы для записи. Эту опцию имеет

смысл включить, если вы используете Ghostscript для просмотра чужих документов, поскольку язык PostScript имеет достаточно средств, чтобы стереть все ваши файлы.

-sSUBSTFONT=*fontname* Использовать шрифт *fontname* для замещения всех неизвестных шрифтов. Ghostscript обычно пытается найти подходящий шрифт и использует Courier только в самом крайнем случае.

Когда Ghostscript стартует, ему нужно найти несколько инициализационных файлов (большинство функций интерпретатора написаны на самом PostScript'е и загружаются при запуске программы). Обычно программа настроена так, что ищет ваши файлы в текущей директории, а программные файлы — в системной директории, однако вы можете изменить эти установки. Вы можете указать несколько дополнительных директорий для поиска с помощью опции `-I` в командной строке либо определить переменную окружения `GS_LIB`, которая указывала бы на эти директории. Имена директорий указываются в виде списка, разделенного знаками : в Unix или знаками ; в DOS и VMS.

11.4.2 Ghostscript и шрифты

Кроме своих системных файлов, Ghostscript'у необходимо найти шрифты. В отличие от большинства PostScript'овских принтеров, в нем обычно нет ни одного встроенного шрифта, и он загружает все необходимое типа PostScript'овских шрифтов Type 1 или шрифтов TrueType с диска. В стандартной поставке Ghostscript имеется базовый набор из 35 шрифтов, которые соответствуют шрифтам, установленным в большей части PostScript'овских принтеров. Это — высококачественные PostScript'овские шрифты Type 1, бесплатно распространяемые немецкой шрифтовой компанией URW. Ghostscript начинает поиск шрифтов с поиска каталога шрифтов, называемого `Fontmap` среди системных файлов (имя каталога можно изменить с помощью опции `-sFONTMAP`). В этом каталоге указывается, как связаны PostScript'овские имена шрифтов с настоящими именами файлов в системе. После этого во всех директориях, указанных с помощью переменной окружения `GS_FONTPATH` или опцией `-sFONTPATH`, он ищет файлы, похожие на файлы PostScript'овских шрифтов. Если Ghostscript решает, что это правильные шрифты, то он добавляет эти файлы и шрифты в свою внутреннюю копию каталога `Fontmap`.

Когда ваши файлы считаны, запросы к шрифтам удовлетворяются путем просмотра каталога `Fontmap`; если в нем не найдено ничего подходящего, то используется шрифт по умолчанию. Некоторые системы Unix (например, OpenWindows фирмы Sun) имеют систему Display PostScript с набором PostScript'овских шрифтов Type 1. В этом случае с помощью опции `GS_FONTPATH` имеет смысл указать на эти шрифты с тем, чтобы все шрифты в системе были автоматически доступны.

Файл каталога `Fontmap` состоит из набора строк, описывающих каждый шрифт. В каждой строке указывается имя PostScript'овского шрифта (начинающееся со знака /), за которым через пробел или табуляцию в скобках

указывается имя файла на диске. Вторая часть строки может быть именем уже определенного шрифта; в этом случае это определение используется как синоним. Строка должна завершаться точкой с запятой. Комментарии добавляются в текст строками, начинающимися со знака %.

В качестве примера рассмотрим известное семейство Palatino, которое представлено шрифтами Palladio компании URW. Ниже приведен фрагмент из `Fontmap`, иллюстрирующий, как имена шрифтов отображаются на реальные имена файлов на диске:

```
/URWPalladioL-Roma      (p0520031.pfb) ;
/URWPalladioL-Ital      (p0520231.pfb) ;
/URWPalladioL-Bold      (p0520041.pfb) ;
/URWPalladioL-BoldItal (p0520241.pfb) ;
```

Полный путь к файлу обычно не указывается, поскольку считается, что шрифты находятся в пути к директории шрифтов. Чтобы разрешить в файлах вызывать шрифты прямо по их настоящим именам Palatino, в каталог добавлен список синонимов:

```
/Palatino-Roman         /URWPalladioL-Roma ;
/Palatino-Italic        /URWPalladioL-Ital ;
/Palatino-Bold          /URWPalladioL-Bold ;
/Palatino-BoldItalic    /URWPalladioL-BoldItal ;
```

Поскольку Ghostscript обычно сам в любом случае просматривает директории в поисках подходящего файла, первые четыре строки на самом деле избыточны, однако список синонимов в `Fontmap` имеет жизненно важное значение до тех пор, пока у вас на диске нет настоящих шрифтов Palatino. Пользователи Adobe Type Manager *действительно* имеют настоящие шрифты Adobe, которые Ghostscript может использовать; для этой и (похожих) установок существуют специальные версии файла `Fontmap`.

11.4.3 Использование различных устройств с Ghostscript'ом

Ghostscript поддерживает множество устройств вывода. Набор устройств, которые вам доступны, зависит от того, как была скомпилирована ваша версия Ghostscript'a, но во всех них обязательно есть вывод на дисплей и поддержка ряда принтеров и растровых форматов файлов. Вы можете посмотреть список доступных устройств, набрав в командной строке команду `gs -h` или запустив программу с опцией `-sNODISPLAY` (чтобы запретить Ghostscript'у открывать окно для вывода, поскольку мы просто хотим обратиться к интерпретатору) и после приглашения набрав:

```
devicenames ==
```

В результате для типичной установки программы в Unix получим примерно следующее:

```
> gs -sNODISPLAY
Aladdin Ghostscript 4.03 (1996-9-23)
```

```
Copyright (C) 1996 Aladdin Enterprises, Menlo Park, CA.
All rights reserved. This software comes with NO WARRANTY:
see the file PUBLIC for details.
GS>devicenames ==
[/ljet3 /pjxl300 /ljet2p /nullpage /tiffpack /pgm /ljetplus
/tifflzw /pbmraw /pjxl /laserjet /tiffg4 /pbm /pj /png256
/tiffg32d /pcx24b /djet500 /cdj550 /png16 /tiffg3 /pcx256
/deskjet /pnggray /tiffcrle /pdfwrite /pcx16 /x11mono
/pngmono /ppmraw /pcxgray /x11cmyk /bitcmyk /ppm /pcxmono
/x11alpha /cdjmono /bitrgb /pnmraw /faxg4 /bjc600 /cdjcolor
/bit /pnm /faxg32d /x11 /cdeskjet /psmono /pgnmraw /faxg3
/bj200 /png16m /tiff24nc /pgnm /bjc800 /bj10e /ljet4
/tiff12nc /pgmraw]
GS>quit
>
```

Некоторые из устройств, доступных в версии 4.03 Ghostscript (последней в момент написания книги), перечислены в приложении А.8 на с. 571.

Вы можете выбрать устройство с помощью опции `-sDEVICE=`*device* в командной строке. Например, чтобы создать выходной файл для цветного принтера Deskjet 550C (если в скомпилированную программу включен драйвер этого устройства, как в примере выше), вы можете набрать команду:

```
gs -sDEVICE=cdj550 file.ps
```

Заметим, что вам надо еще позаботиться о том, чтобы направить вывод действительно на принтер; чуть ниже мы обсудим, как это сделать.

Иначе, прямо из программы, то же самое можно сделать так:

```
(cdj550) selectdevice
(file.ps) run
```

Последний вариант позволяет легко переключаться между устройствами, набирая другие команды `selectdevice` в любое время, например:

```
(cdj500) selectdevice
(cat.eps) run
(tiffg4) selectdevice
(cat.eps) run
```

В этом примере из `cat.eps` сначала будет создан рисунок для печати на цветном Deskjet, а после этого — монохромный сжатый рисунок в формате TIFF.

Если вы хотите всегда использовать одно и то же устройство, вы можете записать его имя в переменную окружения `GS_DEVICE`. Разумеется, эта установка может быть изменена в командной строке.

Когда в качестве устройства указан принтер или растровый графический формат, вывод из программы направляется в файл (по умолчанию Ghostscript открывает для него пустой файл в системной рабочей директории, указываемой переменной окружения `TEMP`). Однако вы можете указать, где создавать выходной файл, с помощью опции в командной строке:

`-sOutputFile=`*outfile*

Для файла, содержащего более одной страницы, вы можете создать *набор файлов*, используя следующий синтаксис:

```
-sOutputFile=name%d.ext
```

Здесь `%d` — спецификация формата, принятая в C для функции, которая заменяется номером каждой страницы (при указании %02d будут выводиться две десятичные цифры). В каждый файл при этом направляется одна страница выходного файла; выбор расширения файла (*ext*) предоставляется вам самим. Например, с помощью команды

```
gs -sOutputDevice=tiffg4 -r600 -dNOPAUSE -sOutputFile=doc%02d.tif doc.ps
```

вы можете создать набор файлов в формате TIFF с именами `doc01.tif`, `doc02.tif` и т. д. для многостраничного PostScript'овского документа с именем `doc.ps`. Мы использовали здесь опцию `-dNOPAUSE`, чтобы запретить программе останавливаться после вывода каждой страницы, и установили разрешение, равное 600 точек на дюйм (см. ниже). Подобную процедуру можно использовать, чтобы посылать сгенерированные страницы факса из вашего компьютера. При создании растровых изображений вы можете попросить Ghostscript включить анти-алиасинг, установив `TextAlphaBits` и/или `GraphicsAlphaBits`. По умолчанию эти параметры установлены в 1 (анти-алиасинг выключен), но значения 2 или 4 включают эту функцию (например, `-dTextAlphaBits=4`). Анти-алиасинг (anti-aliasing) является широко используемым приемом, позволяющим сделать текст на экране более удобочитаемым. Это осуществляется путем установки некоторых пикселов в буквах и на линиях в градации серого с тем, чтобы сделать переход от черного к белому менее резким (см., например Foley et al., 1990, сс. 132ff для более подробного объяснения). Мы продемонстрируем эту функцию вместе с другими возможностями генерации растровых файлов с помощью Ghostscript'a в разд. 11.6 на с. 529.

Если ваша система поддерживает каналы (pipes) (например, Unix), вы можете направить вывод Ghostscript'a прямо на принтер, указав опцию:

```
-sOutputFile=\|lpr
```

Эта опция направляет поток вывода Ghostscript'a на *стандартный ввод* программы lpr (которая используется в Unix для создания очереди на принтер). Под Windows 95 или NT вы можете указать путь к принтеру в формате UNC, например:

```
-sOutputFile="\\spool\printername"
```

В более общем случае выходной поток программы может быть направлен на стандартный выход (`stdout`) для передачи в канал с помощью опции:

```
-sOutputFile=-
```

Если вы это делаете, вам следует указать также ключ `-q`, чтобы быть уверенным, что Ghostscript не будет добавлять в выходной поток свои системные сообщения.

Для некоторых принтеров и большинства растровых форматов файлов вам необходимо указать разрешение на выходе. Это можно сделать с помощью опции `-r`:

```
gs -sDEVICE=device -rresolution
```

Некоторые устройства имеют различное разрешение по осям x и y, и если это требуется, вы можете установить его раздельно. Например, для 9-игольчатого принтера, совместимого с Epson, вы можете указать режим с наименьшей плотностью (самый быстрый) так:

```
gs -sDEVICE=epson -r60x72
```

а режим вывода с наибольшей плотностью — так:

```
gs -sDEVICE=epson -r240x72
```

На практике большинство пользователей, кто хочет управлять своими принтерами с помощью Ghostscript'а, либо пользуются программами-оболочками GSview или ghostview, которые скрывают детали взаимодействия с основной программой удобным интерфейсом, либо интегрируют Ghostscript в Unix'овскую систему печати со спулингом. Как сделать последнее, кратко объяснено в файле документации к Ghostscript'у (`unix-lpr.txt`), но для этого нужен опыт программирования в системе Unix.

При использовании цветных струйных принтеров, таких, как HP DeskJet, HP PaintJet, Epson Stylus Color и Canon Bubblejet Color, обычно требуется тщательная настройка Ghostscript'а. Драйверы для этих принтеров имеют много дополнительных параметров настройки, которые подробно описаны в файле `devices.txt` в дистрибутиве Ghostscript'а. Например, для принтеров DeskJet 850C и 660C рекомендуется добавить следующие опции, чтобы обеспечить разумно быструю и чистую печать (используя всю мощь принтера), но при этом не тратя много цветных чернил (при простой установке драйвера по умолчанию чернила могут замазать страницу):

```
-r300 -dShingling=2 -dDepletion=1 -dBitsPerPixel=32 -sDEVICE=cdj550
```

Установка `BitsPerPixel` дает указание принтеру создавать полноцветное изображение, используя черный картридж одновременно с цветным; опция `Depletion` устанавливает режим адаптивного удаления точек (intelligent dot removal), а `Shingling` включает многопроходную печать для получения более качественных результатов. Разумеется, выбор установки также определяется качеством используемой бумаги и тем, что вам важнее — скорость печати или воспроизведение цвета.

11.4.4 Интерактивные версии Ghostscript'а

Сама программа Ghostscript не предоставляет удобных средств для перемещения по документу взад-вперед, увеличения отдельных частей страницы, поворота и печати выбранных страниц и т. п. Это может быть сделано программными оболочками, которые запускают Ghostscript для вас с более удоб-

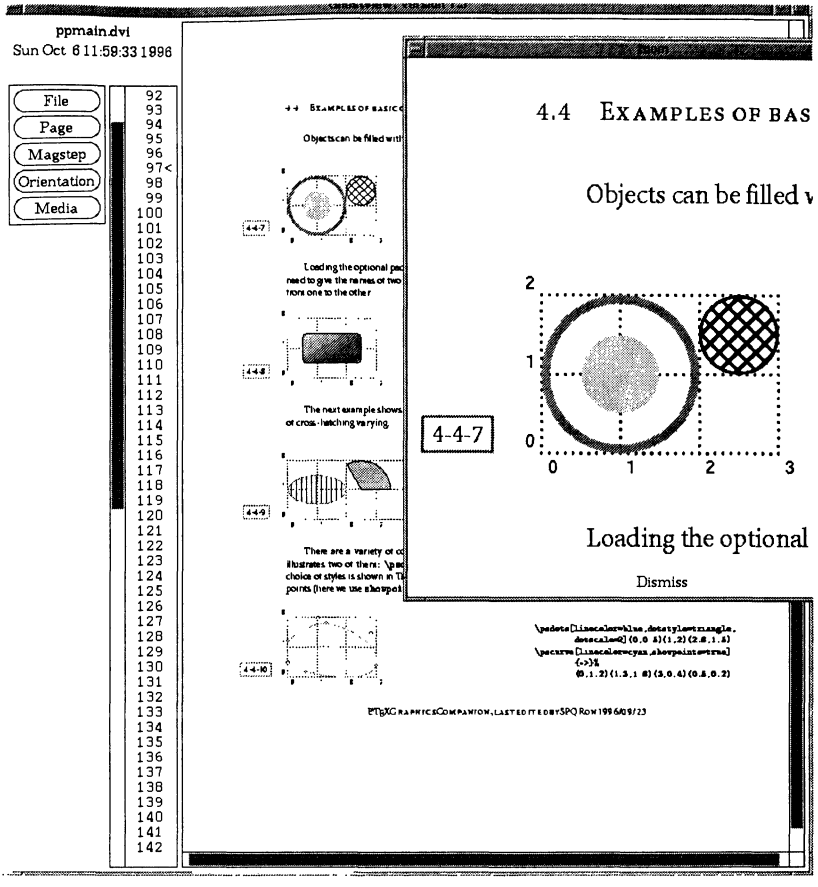


Рис. 11.5. Пример использования программы ghostview.

ным интерфейсом. Такими доступными бесплатными оболочками являются GSview для MS Windows (автор — Рассел Ланг), Mac GS Viewer для Macintosh и ghostview для X Windows System (Unix). На рис. 11.5 приведена страница этой книги, показываемая программой ghostview (автор — Тимоти Тейсен). Одна часть страницы увеличена. [При установке мыши на рабочую область экрана в верхнем левом углу индицируется положение указателя мыши в координатах PostScript'a.] Меню слева показывает доступные функции, которые позволяют открывать новые файлы, печатать выбранные страницы, переходить на страницу с другим номером, изменять увеличение, а также размеры и ориентацию бумаги. Начальные установки для большинства опций (включая выбор цвета и запуск Ghostscript в «безопасном» режиме) могут устанавливаться с помощью опций в командной строке или из файла ресурсов X Windows System. Поскольку для просмотра файлов PostScript ghostview использует механизм каналов

(piping), вы не можете в настоящее время использовать эту программу для просмотра файлов PDF![1]

Программа GSview предоставляет точно те же возможности для пользователей Windows, добавляя при этом такие полезные функции, как экспорт файла в различных графических форматах, включение заголовков для просмотра в eps-файлы, более тонкое управление принтером и сохранение выделенных страниц в новом файле.

Если вы используете Ghostscript под DOS или в Unix без X Windows, то вас может заинтересовать скрипт ps_view, написанный на самом PostScript'е (авторы — Петр Пяновский и Богуслав Яцковский). Он позволяет вам перемещаться по вашему документу взад и вперед, менять увеличение и т. д.

11.4.5 Приложения Ghostscript'а

Работа многих приложений с PostScript'овскими файлами опирается на способность Ghostscript'а обрабатывать (конвертировать, преобразовывать, растеризовать) данные, и мы рассмотрим здесь некоторые из этих приложений. Программа gsftopk, которая будет описана в разд. 11.5.2, также использует Ghostscript.

Эти небольшие приложения часто представляют собой скрипты, вызывающие PostScript'овские программы; в этом случае они представлены как в виде скриптов для командного процессора Unix, так и в виде пакетных файлов для DOS. Поскольку в этих скриптах обычно указываются опции для передачи Ghostscript'у в командной строке, пользователи других операционных систем могут также ими воспользоваться.

Извлечение текста из PostScript'овских файлов

В дистрибутиве Ghostscript'а есть PostScript'овский файл `ps2ascii.ps`, который извлекает ASCII-текст из PostScript'овского файла.[2] При этом ничего не отображается, а текстовый файл направляется на стандартный вывод. Этот файл вызывается из скрипта-«оболочки» ps2ascii следующим образом:

```
ps2ascii myfile.ps myfile.txt
```

Необязательный аргумент указывает имя создаваемого текстового файла; если он не задан, то текст выводится на стандартное устройство вывода.

Другая утилита, pstotext, созданная Эндрю Бирреллом и Полом МакДжонсом, несколько более устойчива. Как и `ps2ascii.ps`, она читает один или несколько PostScript'овских файлов и записывает извлеченный

[1] Хотелось бы отметить еще одну программу, которая позволяет делать практически то же самое, что и ghostview, но еще умеет и просматривать файлы PDF. Это — программа gv, созданная Йоханессом Плассом на основе ghostview версии 1.5; см. `http://wwwthep.physik.uni-mainz.de/~plass/gv/`. — *Прим. перев.*

[2] При работе с Ghostscript'ом версии 6.50 эта утилита позволяет извлекать текст и из pdf-файлов. К сожалению, ps2ascii не умеет работать с вектором кодировки и поэтому непригодна для работы с текстами на русском языке. — *Прим. перев.*

текст в таком виде, как если бы PostScript'овский файл был напечатан. Для своей работы pstotext использует Ghostscript и загружает библиотеку PostScript'a, которая локализует каждую строку в файле и извлекает из файла дополнительную информацию, необходимую для вычисления размера прямоугольника, занимаемого этой строкой. После этого pstotext обрабатывает эту информацию и выводит последовательность слов с восстановленными пробелами, началами абзацев и переводами на следующую страницу.

Программа pstotext переводит все восьмибитовые входные коды в символы ISO 8859-1. Если символ в ISO 8859-1 отсутствует, программа вставляет последовательность символов, например «---» для emdash.

Создание eps-файлов

Скрипт-утилита ps2epsi берет входной PostScript'овский файл и создает выходной файл, удовлетворяющий формату Encapsulated PostScript Interchange (EPSI) фирмы Adobe. В этой специальной разновидности Encapsulated PostScript в начало файла (в виде комментариев PostScript) добавлена растровая версия отображаемой страницы. Этот формат полезен для текстовых процессоров или настольных издательских программ типа FrameMaker, в которых он используется, чтобы увидеть примерный вид рисунка на обычном дисплее. Этот скрипт вызывается следующим образом:

```
ps2epsi myfile outfile
```

Если имя выходного файла не указано, то используется то же имя файла, что и у входного файла, но с расширением `epsi`.

Подготовка файлов для Adobe Illustrator

Программа Adobe Illustrator широко используется для подготовки и редактирования достаточно сложных иллюстраций. В качестве естественной формы хранения рисунков она использует сильно структурированный и упрощенный язык PostScript, и поэтому в Ghostscript включена PostScript'овская программа `ps2ai.ps`, которая пытается преобразовать обычный PostScript в форму, которую может использовать Adobe Illustrator. Существуют некоторые ограничения на код PostScript'a, который она понимает, но в целом полученный ею код может использоваться другими программами, которые в качестве входного формата принимают PostScript'овский код, совместимый с Adobe Illustrator. Эта программа может быть вызвана следующим образом:

```
gs -q -dNODISPLAY ps2ai.ps infile.ps > outfile.ai
```

Программа имеет несколько настраиваемых параметров для управления процессом (`jout` для записи в файл, `jtxt3` для вывода в формате третьей версии Adobe Illustrator, `joutln` для замены шрифтов их векторными изображениями и т. п.); более подробно о них написано в комментариях в файле `ps2ai.ps`.

Работа с PDF (Portable Document Format)

Программа ghostscript выпускается с двумя скриптами, ps2pdf и pdf2ps, которые конвертируют, соответственно, обычный PostScript в PDF и PDF в PostScript. PDF — формат, используемый в продуктах Adobe Acrobat, — по сути является менее сложным и структурированным вариантом PostScript'а; из него удалены функции языка программирования, добавлены возможности гипертекста и адаптивного замещения шрифтов, а структура PDF-файлов оптимизирована для быстрого отображения на дисплее. Обе программы являются простыми скриптами, которые, как обычно, вызывают Ghostscript с соответствующими параметрами. Их использование иллюстрируется следующими командными строками:

```
ps2pdf myfile.ps outfile
pdf2ps myfile.pdf outfile
```

11.5 Преобразование шрифта формата PostScript в шрифт формата PK

Большинство визуализаторов TEX'а не могут показывать PostScript'овские шрифты, пока те не переведены в формат pk (или более старый, вышедший из употребления формат pxl), как это делает METAFONT. Исключениями являются визуализаторы, которые глубоко интегрированы с развитыми системами отображения шрифтов типа Adobe Type Manager для Windows или операционной системы Macintosh. То есть, TEX-визуализатор для NeXTStep, Textures фирмы Blue Sky для Macintosh, dviwindo фирмы Y&Y и TrueTeX-визуализатор фирмы Kinch для Windows — все они используют *системное* программное обеспечение для рисования глифов, работая при этом с PostScript'овскими шрифтами Type 1 или шрифтами True Type. Несомненно, это наилучший способ организации работы, поскольку визуализатор может сконцентрироваться на dvi-файле и вычислении положений слов, оставляя задачу построения прецизионных изображений глифов специализированным программам. Другим способом визуализации текста может быть преобразование dvi-файла в PostScript и просмотр его с помощью PostScript'овского визуализатора, подобного Ghostscript'у, но многие пользователи находят этот двухэтапный процесс слишком медленным.

Существует, однако, много пользователей TEX'а, кто не работает в окружении Windows или Macintosh и не хочет вкладывать средства в приобретение коммерческих продуктов, а хотел бы использовать замечательные TEX'овские визуализаторы, которые уже имеются в их системах. Эти пользователи могут создавать файлы шрифтов в формате pk из PostScript'овских шрифтов Type 1, используя одну из двух свободно доступных программ — ps2pk или gsftopk. Получаемое качество зависит от используемых ими растеризаторов, и вам следует очень внимательно рассмотреть получаемые на выходе документы, если вы планируете делать серьезную работу с использованием этих растровых шрифтов. Однако для целей предварительного просмотра они, безусловно, прекрасно подойдут.

11.5.1 Программа ps2pk

Программа ps2pk была написана Питом Тютеларсом на основе кода, который фирма IBM подарила X Consortium'у для растеризации PostScript'овских шрифтов Type 1. Это отдельная бесплатная программа, которая читает шрифт Type 1 и его метрический файл в формате Adobe (afm) и записывает файл pk для заданного разрешения.

```
ps2pk-v -Xxresolution -Yyresolution -Ppointsize
     -Sslant -eencoding -Eexpansion -aAFMfile
     -mmodename -raspectratio -Rbaseresolution
     type1fontname pkfontname
```

Ниже перечислены значения различных опций:

-a*AFMfile* Указывает имя afm-файла. По умолчанию оно считается таким же, как и имя самого шрифта Type 1.

-e*enc* Указывает файл кодировки (по умолчанию используется кодировка, указанная в afm-файле); формат вектора кодировки здесь такой же, как и описанный в разд. 10.3.3.

-E*extension* Растянуть или сжать шрифт (по умолчанию *extension* = 1.0, т. е. шрифт не изменяется).

-P*pointsize* Установить размер результирующего шрифта в пунктах (по умолчанию 10pt).

-S*slant* Наклонить шрифт (по умолчанию шрифт не наклоняется).

-X*xres* Установить разрешение в направлении x в точках на дюйм (по умолчанию 300).

-Y*yres* Установить разрешение в направлении y в точках на дюйм (по умолчанию 300). Обычно это разрешение совпадает с разрешением в направлении x и не указывается.

-v Разрешить выдачу сообщений о ходе выполнения программы.

Остальные опции служат главным образом для обеспечения совместимости с другими программами, генерирующими pk-шрифты (подобными METAFONT'у), и записывают дополнительную информацию в выходной файл:

-m*modename* Установить имя режима, как это делает METAFONT (используется для записи идентифицирующей информации в выходной файл).

-r*aspectratio* Установить пропорции (по умолчанию 300/300).

-R*baseres* Установить основное разрешение, с котором вы работаете (по умолчанию 300dpi). Различие между этим разрешением и разрешением по оси x (если оно есть) записывается в выходной файл как TEX'овский параметр magnification.

Использование опций -X и -P может привести к некоторым недоразумениям. Важно понимать теоретическое различие между шрифтом, растеризованным с удвоенным размером для устройства с разрешением 300dpi, и шрифтом нормального размера для устройства 600dpi. Оба они имеют одинаковое разрешение (600 точек на дюйм), но используются для разных целей. Полученные файлы называются по-разному и внутренние идентификаторы в них делают это различие яснее. Таким образом, две команды

```
ps2pk -P20 Times-Roman.pfb
ps2pk -X600 Times-Roman.pfb
```

порождают выходные файлы Times-Roman20.300pk и Times-Roman10.600pk, соответственно. Практически, при использовании схемы именования шрифтов и кодировке 8r (о которых мы говорили в гл. 10) эти команды скорее всего будут вызываться следующим образом:

```
ps2pk -e8r.enc -mcx -P20 ptmr8a.pfb  cx/dpi600/ptmr8r.pk
ps2pk -e8r.enc -mljfour -X600 ptmr8a.pfb ljfour/dpi600/ptmr8r.pk
```

Здесь мы назвали выходные pk-файлы явным образом и дополнили их именами директорий с тем, чтобы было совершенно ясно, какой файл предназначен для какого устройства.

Спецификации кодировки, наклона и сжатия–растяжения шрифта прямо соответствуют спецификациям в map-файле программы dvips (см. разд. 11.2.4 на с. 497).

Программа ps2pk находит шрифты Type 1 и их afm-файлы, используя переменную окружения T1INPUTS. Эта программа включена в teTeX (дистрибутив TEX для Unix) в полностью установленном виде, но также может быть найдена в компилированном виде для DOS и OS/2 в архивах CTAN.

11.5.2 Программа gsftopk

Программа gsftopk (автор — Пол Войта) выполняет те же функции, что и ps2pk, но использует Ghostscript для растеризации глифов из PostScript'овских шрифтов Type 1 и записи их в TEX'овский формат pk. Эта программа применяется в основном в системах Unix. Программа вызывается следующим образом:

```
gsftopk -q fontname dpi
```

где опция -q указывает программе работать, не выдавая сообщений («молча»)[1]. Обязательными параметрами являются имя шрифта (например, ptmr8r) и разрешение (например, 600). Результатом работы программы будет pk-файл с именем ptmr8r.600pk.

Программе gsftopk необходимо найти TEX'овские метрические (tfm) файлы для шрифтов; кроме того, она может использовать тот же map-файл для имен

[1] На самом деле программа имеет еще массу других необязательных параметров. — *Прим. перев.*

PostScript'овских шрифтов и информацию об их перекодировке, как программа dvips (см. разд. 11.2.4 на с. 497). Программа gsftopk обычно вызывается из скрипта MakeTeXPK, который сам (вместо пользователя) устанавливает правильные пути и т. п. Подобно ps2pk, он входит в полностью установленном виде в teTeX — дистрибутив TEX'a для Unix.

11.6 Генерация рисунков для Web-страничек с помощью dvips и Ghostscript'a

В настоящее время многие пользователи хотели бы выставить свои LATEX'овские документы в Сети. Однако до тех пор пока не появится полная поддержка математики в HTML или пока не появятся модули (plug-in) для просмотра уравнений в навигаторах (Web browsers), вы не можете осуществить полноценное преобразование. Математические формулы и рисунки из LATEX'a обычно преобразуют в рисунки в формате GIF. В этом разделе мы затронем два аспекта этого подхода: создание качественных PostScript'овских изображений фрагментов выходного файла LATEX'a и преобразование этого PostScript'a в формат GIF.

Чтобы создать качественный PostScript'овский выходной файл с помощью dvips (это относится и к файлам, предназначенным для Acrobat Distiller), абсолютно необходимо использовать векторные шрифты. Если вы не будете использовать векторные шрифты и сделаете масштабирование вашего выходного файла, то результат может оказаться чудовищным (сравните рис. 11.6 и рис. 11.7). Если вы захотите превратить ваш документ в PDF, то Distiller сделает очень плохие изображения из pk-шрифтов.

$$-\Phi_0\frac{\partial}{\partial\varphi}\left(\Phi_1 a\sin\varphi\right)-\Phi_1\frac{\partial}{\partial\varphi}\left(\Phi_0 a\sin\varphi\right)-A_1\left[\Phi_0+\frac{\partial}{\partial a}\left(a\Phi_0\right)\right]\sin\varphi=-a\Phi_0 f\sin\varphi \qquad (1)$$

Рис. 11.6. Растровый eps-файл, увеличенный и искаженный.

$$-\Phi_0\frac{\partial}{\partial\varphi}\left(\Phi_1 a\sin\varphi\right)-\Phi_1\frac{\partial}{\partial\varphi}\left(\Phi_0 a\sin\varphi\right)-A_1\left[\Phi_0+\frac{\partial}{\partial a}\left(a\Phi_0\right)\right]\sin\varphi=-a\Phi_0 f\sin\varphi \qquad (1)$$

Рис. 11.7. eps-файл с векторным шрифтом, увеличенный и искаженный.

Вторым по порядку важности является использование правильных размеров боксов. Как это ни удивительно, но многие приложения неправильно указывают размер рисунка, считая его просто равным размеру страницы (независимо от того, заполнена вся страница или нет). Программа dvips по умолчанию делает то же самое, но имеет опцию `-E` в командной строке, которая заставляет dvips рассчитать размер действительно занятого пространства. Заметим, что по определению eps-файлы могут занимать только одну страницу, поэтому вы также должны задать dvips опции, чтобы работать только с одной страницей. Необходимо сделать два предостережения относительно подготовки входного файла. Во-первых, убедитесь, что у вас выключена нумерация страниц (используйте `\pagestyle{empty}` в LaTeX'е), так как иначе размер бокса будет вычислен с учетом номера страницы. Во-вторых, dvips не всегда обрабатывает текст корректно. Например, если вы пишете:

```
Hello\raisebox{10pt}[0pt][0pt]{Up there}!
```

то вы просите LaTeX поднять слова *Up there* над базовой линией, но при этом притвориться, что высота строки не изменилась. Программа dvips верит этому и вычисляет размер бокса на основании *декларированной* высоты. Если вы используете сложные пакеты, подобные PSTricks, которые добавляют произвольный PostScript'овский код, то вы также можете попасть впросак. В этих случаях вам надо либо скорректировать строку `%%BoundingBox` в PostScript'овском файле вручную, либо вставить невидимые метки в LaTeX'е, чтобы быть уверенным, что dvips вычислит размер правильно.

Если вы уверены, что LaTeX понимает, что вы хотите, а dvips — нет, то полезно вспомнить следующий прием: намеренно использовать цвет. Допустим, что вы хотите использовать PSTricks, чтобы обвести кружком математический символ. Вы можете написать:

absurd (√)

```
\usepackage {pstricks,graphicx}
absurd \pscirclebox{$\surd$}
```

11-6-1

При этом LaTeX выделит нужное место, поскольку макро PSTricks понимают, что нужно будет сделать. Однако поскольку dvips дается указание нарисовать кружок с помощью добавления чистого кода PostScript'а, а при вычислении размеров обрамляющего бокса этот код не учитывается, в результате границы бокса будут установлены около границ букв. Если же вы напишете:

```
\framebox{absurd \pscirclebox{$\surd$}}
```

то это сработает правильно, поскольку dvips видит включенную рамку, а не просто слова. Однако при этом вы получите нежелательную рамку вокруг слов. Поэтому вам следует сделать бокс невидимым, написав:

```
{\color{white}\fboxsep{0pt}%
\framebox{%
  {\color{black}absurd \pscirclebox{$\surd$}}%
  }%
}
```

Это создаст белую рамку вокруг черного текста. LaTeX и dvips благополучно обработают это, рассчитав правильный размер, но ничего не показав на бумаге. Очевидно, это будет работать только в монохромном окружении.

Другим способом найти правильные размеры обрамляющего бокса в вашем выходном файле является использование программы Рассела Ланга epstool, которая распространяется вместе с GSview для Windows; мы активно использовали эту программу при подготовке настоящей книги. Программа добавляет заголовок для просмотра в PostScript'овский файл, но в виде побочного эффекта она позволяет рассчитать правильные размеры бокса, поскольку обращается к Ghostscript'у для создания растрового изображения; размер мы затем можем использовать. Вы можете далее использовать ту же программу, чтобы (если хотите) удалить заголовок для просмотра. Эта программа вместе с другими методиками обсуждается в (Merz, 1996, сс. 97–99).

Зачем же вам все это может понадобиться практически? Как мы уже говорили, часто это нужно, чтобы иметь возможность показывать математические формулы из LaTeX'а в Сети, и единственным способом это сделать является включение рисунков GIF в ваш HTML. Достаточно развитая программа latex2html сделает это для вас. В технике, используемой в этой программе, стоит разобраться подробнее, поскольку это достаточно общая утилита. Ниже представлена последовательность того, что делает эта программа (большой сценарий, написанный на Perl):

1. помещает фрагменты LaTeX'а в специальный файл, по фрагменту на странице и без нумерации страниц;
2. запускает LaTeX, чтобы получить многостраничный dvi-файл;
3. использует опции dvips `-i` и `-S` для генерации отдельных файлов для каждой страницы;
4. запускает Ghostscript, чтобы преобразовать каждую страницу в формат PBM (Portable Bitmap);
5. использует утилиту pnmcrop из пакета netpbm, чтобы отсечь белое пространство;
6. использует утилиту ppmtogif из того же пакета, чтобы преобразовать результат в картинку GIF.

Заметим, что при запуске dvips *не используется* опция `-E`, а просто из полученного рисунка удаляются все белые пикселы, пока не останется только текст. Преимуществом этого подхода является то, что при этом мы обходим проблемы с вычислением правильных размеров бокса, однако этот подход имеет три недостатка:

1. утилиты PBM прежде всего известны в Unix, и многим пользователям они недоступны;[1]

[1] В настоящее время пользователи TeX'а под Windows могут получить доступ ко всем утилитам, входящим в пакет netpbm, установив архив `netpbm-1,2-win32.zip`, имеющийся на CD-ROM'е TeX Live 6. — *Прим. перев.*

2. процедура отсечения белого пространства требует много памяти, медленна и использует большие массивы для временных файлов на диске;
3. процесс отсечения белого пространства эффективно выводит все на базовую линию. Поэтому символы типа emdash (—), которые обычно располагаются выше базовой линии, отсекают сверху и снизу и полученный рисунок GIF неправильно располагается по вертикали.

Корень проблем состоит в использовании Ghostscript'а, который всегда создает растровый файл размером в одну страницу, даже если на ней расположено всего одно слово. Допустим, вам хочется, чтобы Ghostscript перевел только часть рисунка внутри бокса (вы уже указали флаг `-E` для dvips). Вы можете добиться этого, указав Ghostscript'у искусственный размер страницы, который равен размеру бокса. После этого вы можете добавить небольшой фрагмент кода PostScript'а и сдвинуть рисунок так, чтобы он начинался в точке с координатами (0,0) (и скорректировать информацию о боксе соответствующим образом). После этого Ghostscript показывает или конвертирует рисунок только в пределах указанной области и никакого отсечения белого пространства не требуется.

Преобразования размеров бокса можно осуществить программой epsffit из коллекции psutils, о которой мы говорили в разд. 11.3.1 на с. 505. Изменение размера страницы наиболее просто сделать оператором `setpagedevice` из PostScript'а второго уровня. В этом случае PostScript'овский файл, который начинается так:

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 135 528 284 668
...
```

должен быть преобразован во что-нибудь подобное

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 0 0 149 140
<< /PageSize [149 140] >> setpagedevice
gsave -135 -528 translate
...
grestore
```

В этом примере мы рассчитали ширину и высоту прямоугольника (149 × 140 единиц), сдвинули начало отчета в (0,0) и установили размер страницы. Команда `setpagedevice` не совсем правильна в Encapsulated PostScript, но раз вы используете этот файл локально и только для последующей обработки с помощью Ghostscript'а, все достаточно «безопасно». На рис. 11.8 приведен текст простого сценария на Perl'е, который производит необходимые изменения в файле PostScript'а для дальнейшей передачи программе Ghostscript без использования epsffit.

Теперь, когда Ghostscript работает только с выделенной областью, вы можете использовать его встроенную функцию для создания растров. Командная строка для Unix или DOS

```
gs -dNOPAUSE -q -r100 -sDEVICE=tiffg4 -sOutputFile=foo.tif foo.ps -c quit
```

```
$bbneeded=1;
$bbpatt="[0-9\.\-]";
while (<>) {
 if ( /%%BoundingBox:(\s$bbpatt+)\s($bbpatt+)\s($bbpatt+)\s($bbpatt+)/ )
 {
     if ($bbneeded) {
         $width = $3 - $1;
         $height = $4 - $2;
         $xoffset = 0 - $1;
         $yoffset = 0 - $2;
         print "%%BoundingBox: 0 0 $width $height\n";
         print "<< /PageSize [$width $height] >> setpagedevice\n";
         print "gsave $xoffset $yoffset translate\n";
         $bbneeded=0;
    }
}
else {  print; }
}
print "grestore\n";
};
```

Рис. 11.8. Сценарий на Perl'e, производящий необходимые преобразования в eps-файле для Ghostscript.

$$-\Phi_0\frac{\partial}{\partial\varphi}(\Phi_1 a\sin\varphi)-\Phi_1\frac{\partial}{\partial\varphi}(\Phi_0 a\sin\varphi)-A_1\left[\Phi_0+\frac{\partial}{\partial a}(a\Phi_0)\right]\sin\varphi=-a\Phi_0 f\sin\varphi$$

Рис. 11.9. Рисунок, полученный с помощью преобразования по цепочке LaTeX → dvi → EPS → GIF.

создает рисунок в формате TIFF fax group 4 с разрешением 100 точек на дюйм для только что выделенной области PostScript'овского файла `foo.ps`, который не требует дальнейшей доработки. В версии 4 программы Ghostscript добавлена поддержка анти-алиасинга; вы можете создать вариант рисунка GIF, используя утилиты netpbm под Unix и командную строку:

```
gs -r100 -dNOPAUSE -q -sOutputFile=-  \
  -sDEVICE=pnm -dTextAlphaBits=4 \
  -dGraphicsAlphaBits=4 foo.ps -c quit | \
  ppmtogif -interlace \
  -transparent \#ffffff > equation.gif
```

На рис. 11.9 показан результат преобразования без анти-алиасинга (рисунок действительно нехорошо выглядит на печатной странице).

У нас осталась одна проблема: программы просмотра World Wide Web обычно умеют выравнивать рисунки по верху, низу или по середине, но что

делать, если некоторые символы имеют элементы, выступающие ниже базовой линии? Выравнивание этих рисунков по низу помещает эти элементы на базовую линию, выравнивание по верху смешно, а выравнивание по центру также не совсем правильно. Решение таково: использовать выравнивание по центру и заставить TEX солгать драйверу dvips (и далее по цепочке) относительно истинных размеров символа. Установка глубины символа равной его высоте и последующее выравнивание по центру в программе просмотра дают желаемый эффект.

Конечно, читателям, создающим документы для Сети, не стоит воспринимать приведенные примеры как «рецепты» без тщательного определения того, как должна выглядеть Web-страница, и понимания некоторых основных методов обработки изображений. Нашей целью было просто показать, как сравнительно просто и эффективно можно создать растровый файл, скомбинировав LATEX и dvips и использовав при этом свободно распространяемые утилиты Ghostscript'a.

11.7 PSfrag — замещение текста в помеченных местах на рисунках

Пакет PSfrag (авторы — Майкл Грант, Крейг Баррет и Дэвид Карлайл) решает общую проблему, возникающую при включении графических файлов. Он предоставляет механизм, посредством которого TEX можно заставить включать надписи, уравнения и другие сложные объекты в PostScript'овский графический файл, которые замещают имеющиеся там текстовые элементы. Работа этого пакета основана на том, что интерпретатор PostScript'а ищет определенные строки в вашем PostScript'овском файле и делает в них замену.

Команда \psfrag, определенная в пакете PSfrag, позволяет вам «пометить» слово в PostScript'овском файле в том месте, где вы хотите, чтобы LATEX сделал замену. Текст-замена для этой «метки» в LATEX'овском документе определяется аргументами этой команды:

\psfrag{*tag*}[*posn*][*psposn*][*scale*][*rot*]{*LATEX material*}

и тогда метка *tag* замещается на *LATEX material*. Первые два необязательных аргумента в команде используются, чтобы указать, как следует расположить боксы LATEX'овского текста и PostScript'овского текста относительно друг друга; их синтаксис аналогичен синтаксису стандартной команды \makebox. Третий и четвертый необязательные аргументы *scale* и *rot* позволяют изменить масштаб и ориентацию набираемого текста.

В качестве примера рассмотрим следующий PostScript'овский файл:

```
%!
%%BoundingBox:100 100 172 136
1 setlinewidth
100 100 moveto
```

```
/Times-Roman findfont
12 scalefont setfont
(nabla) show
100 120 moveto
(ALPHA) show
showpage
```

При обычной обработке он производит:

ALPHA

nabla

Если же вы укажете команду

```
\psfrag{ALPHA}{$\alpha \mathcal{ALPHA}$}
\psfrag{nabla}{$\nabla$}
```

в TEX'овском файле, включающем этот рисунок, и запустите LATEX снова, вы получите:

$\alpha\mathcal{ALPHA}$

∇

Все вызовы \psfrag, которые предшествуют команде \includegraphics (или эквивалентной ей), оказывают влияние и на все последующие включенные PostScript'овские файлы. Это позволяет с помощью \psfrag определять как глобальные, так и локальные замещения.

Описываемый метод не требует редактирования PostScript'овского файла, поскольку все делается интерпретатором PostScript'а;[1] однако при этом существуют некоторые ограничения касающиеся того, как замещаемый текст должен быть представлен в выходном PostScript'овском файле. При выводе в виде PostScript'а, некоторые программы трактуют каждую букву слова как отдельную строку и даже в простых выходных файлах из программ, подобных dvips, не ясно, какие слова пройдут сквозь нее как целые строки. К примеру, в то время как большинство слов нижеследующего PostScript'овского фрагмента поддаются обработке с помощью PSfrag, несколько слов оказались разорванными в непредсказуемых местах:

```
118 w(Y)l(ellowGreen)p SpringGreen 1734 1156 V Black
104 w(SpringGreen)p OliveGreen 46 1255 V Black 85 1305
a(OliveGreen)p RawSienna 384 1255 V Black 127 w(RawSienna)p
Sepia 721 1255 V Black 179 w(Sepia)p Brown 1059 1255
V Black 226 w(Brown)p Tan 1396 1255 V Black 245 w(T)g(an)p
Gray 1734 1255 V Black 260 w(Gray)p Black Black eop
```

Кроме того, при работе с этим пакетом есть вероятность получить нежелательные эффекты. Так, если вы указали в качестве метки строку «A», которая

[1] В более ранних версиях требовалась стадия предварительной обработки.

должна быть заменена на α, то и во всех отдельных строках, в которых генерирующая программа оставит одну букву «A», также будет сделано замещение. С другой стороны, если вы пометили «A» как α, а «B» как β, то «AB» *не будет* замещено.

Если размеры графического файла при вставке должны быть дополнительно изменены, важно понимать различие между действием, производимым командой

```
\resizebox{2in}{!}{\includegraphics{file.ps}}
```

и командой, использующей опцию пакета graphicx,

```
\includegraphics[width=2in]{file.ps}
```

При использовании драйверов современных пакетов графики в первом случае элементы PSfrag будут масштабироваться в соответствии с рисунком, а во втором — нет. Поскольку оба варианта поведения могут понадобиться в различное время, на это следует смотреть как на специфическую особенность, а не как на проблему.

Пакет PSfrag принципиально полезен как дополнение к сравнительно простым программам рисования, таким, как xfig и idraw, и к пакетам построения графиков, таким, как MATLAB и gnuplot. В последних программах оси на графиках могут быть помечены уникальными идентификаторами-строками, которые будут замещены в LaTeX'е. Пакет PSfrag позволяет вам автоматизировать добавление текста при обработке большого набора графиков, создаваемых с помощью какой-нибудь одной программы. Таким образом, хотя PSfrag является «умной» программой, хорошо исполняющей свою функцию, он не идеален для обработки крупных PostScript'овских документов произвольной сложности. В идеале PostScript'овский файл должен быть исходно подготовлен для работы *совместно с* PSfrag. Заметим также, что систематическое использование PSfrag требует ясного понимания как языка PostScript, так и работы приложений, с помощью которых создаются рисунки.

Технические приложения

А.1 Команды \special, используемые в emTEX'е

Драйверы emTEX'а поддерживают четыре команды \special, которые также поддерживаются драйверами, подобными dvips. Первая команда позволяет включать растровые изображения PCX или MSP, используя синтаксис

```
\special{em:graph filename}
```

к которому в реализации dvips добавлена опция масштабирования. В dvi-файле для рисунка не резервируется никакого места, так что автор должен сам зарезервировать необходимое пустое пространство.

Другие команды позволяют пользователю задавать произвольные точки на странице и соединять их прямыми линиями:

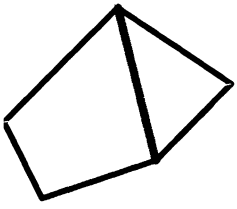
```
\special{em:point n}
\special{em:line n1modifier, n2modifier, width}
\special{em:moveto}
\special{em:lineto}
\special{em:linewidth width}
```

Точка с номером n, определенная с помощью `point`, получает координаты текущей точки (которая может быть установлена, например, в окружении `picture`). Команда `line` соединяет любые две точки линией ширины *width* (этот параметр необязателен). Ее значение по умолчанию может быть установлено с помощью `linewidth`. Ширина линии выражается в обычных единицах, используемых в TEX'е. Модификатором *modifier* в команде `line` может быть одно из значений `h`, `v` или `p`; оно определяет, как будут обрезаны края линий: по горизонтали, вертикали или перпендикулярно линии.

Вместо того чтобы нумеровать точки, вы можете перемещаться к новой точке с помощью команды `moveto` и рисовать линию между текущей и предыдущей точками с помощью команды `lineto`.

Эти простые команды \special имеют большое преимущество потому, что они могут использоваться в стандартном окружении `picture` и не надо беспо-

коиться о ширине линии или угле. Как показано в примере ниже, стандартные команды пакета color также оказывают (ожидаемое от них) влияние на линии:



```
\setlength{\unitlength}{5mm}
\begin{picture}(10,10)(1,0)
\put(2,2){\special{em:point 1}}
\put(5,3){\special{em:point 2}}
\put(7,5){\special{em:point 3}}
\put(4,7){\special{em:point 4}}
\put(1,4){\special{em:point 5}}
\special{em:linewidth 2pt}
{\color{blue}\special{em:line 1,2}\special{em:line 2,3}}
\special{em:line 3,4}  \special{em:line 4,5}
\special{em:line 5,1}
\color{yellow}\special{em:line 3,5,0.5pt}
\color{red}\special{em:line 4,2,3pt}
\end{picture}
```

A-1-1

A.2 Соглашения для \special, используемые в tpic

Как показывают различные примеры, приведенные в гл. 1, соглашения tpic определяют мощный набор команд рисования. Все команды \special в tpic обозначаются двухбуквенными ключевыми словами, за которыми могут следовать несколько аргументов. Все аргументы, определяющие расстояния, представлены целыми числами за несколькими исключениями, отмеченными ниже. Углы выражаются в радианах. К сожалению, в координатной системе dvi ось y начинается наверху страницы и направлена вниз, и вы должны иметь это в виду, когда пишете макропакеты с использованием команд \special в tpic.

В табл. А.1 перечислены графические примитивы, которые должен поддерживать dvi-процессор для работы с tpic; теперь мы опишем подробнее параметры этих примитивов.

Таблица А.1. Сводная таблица команд \special в tpic.

Имя \special	*Действие*
pn	устанавливает ширину линии
pa	добавляет позицию (x, y) к контуру
fp (ip)	рисует текущий контур в виде набора прямых линий
da	то же, но штриховой линией
dt	то же, но точечной линией
sp	рисует текущий контур в виде сплайна
ar (ia)	для заданных начального и конечного углов рисует дугу или окружность
sh	заштриховывает ранее определенный замкнутый объект

`pn` s Установить размер пера в s (в тысячных долях дюйма).

`pa` x y Добавить точку (x, y) к «контуру», который «собирается» в интерпретаторе `dvi`. Значения x и y выражаются в тысячных долях дюйма, а положение отсчитывается от положения *следующего* `\special`, который осуществляет вывод (такого, как `fp`). Набор координат запоминается в памяти, а вывод на страницу происходит только тогда, когда подается команда вывода и только в том случае, если вычислены все абсолютные координаты.

`fp` Нарисовать ранее определенный контур пером текущего размера. После этого число элементов в контуре обнуляется. Если была указана штриховка и контур был замкнут, то его внутренняя область заштриховывается.

`ip` То же, что и команда `fp`, за исключением того, что на самом деле контур не рисуется. Однако если была указана штриховка, то она выводится.

`da` f То же, что и команда `fp`, за исключением того, что штриховая линия рисуется штрихами длины f дюймов (здесь f — действительное число).

`dt` f То же, что и команда `fp`, за исключением того, что рисуется точечная линия с расстоянием между точками f дюймов (здесь f — действительное число).

`sp` d То же, что и команда `fp`, за исключением того, что через точки проводится сплайн. При этом сплайн пройдет точно только через первую и последнюю точки, а его остальной путь будет определяться относительным расположением промежуточных точек. Необязательный аргумент d — вещественное число. Если $d = 0$ или аргумент опущен, то сплайн рисуется в виде непрерывной линии. Если $d > 0$, то линия становится штриховой с длиной штриха d; если $d < 0$, то линия становится пунктирной с шириной точки, равной $-d$.

`ar` x y r_x r_y s e Нарисовать дугу с центром в (x, y) от начального угла s до конечного угла e. Если дуга является полной окружностью или эллипсом, то r_x и r_y определяют, соответственно, радиусы в направлении x и y. В противном случае рисуется дуга от s до e, а x и y являются идентичными (каким бы ни было значение r_x).

`ia` x y r_x r_y s e То же самое, что и `ar`, но кривая не рисуется. Если указана штриховка, то область, ограниченная контуром, заштриховывается.

`sh` s Заштриховать внутреннюю область следующего замкнутого объекта. Для этого необходимо, чтобы ранее были выполнены по крайней мере три команды `pa`, за которыми бы следовали команды `fp` или `ip`, или команды `ar` или `ia`. Величина s — действительное число от 0 до 1. Нуль означает, что «заштрихованная» область становится совершенно белой (включая стирание всего, что лежит под ней), единица обозначает все совершенно черное. Значение 0.5 является «нормальной» серой штриховкой, выбираемой по умолчанию (если s не указано, то используется это значение). Штриховка применяется только к внутренней части объекта; границы объекта рисуются текущим пером (это в особенности важно для `dvi`-драйверов, которые не поддерживают команду `sh`). Пикселы, генерируемые при штриховке, объ-

Таблица А.2. Опции, передаваемые в командной строке программе bm2font .

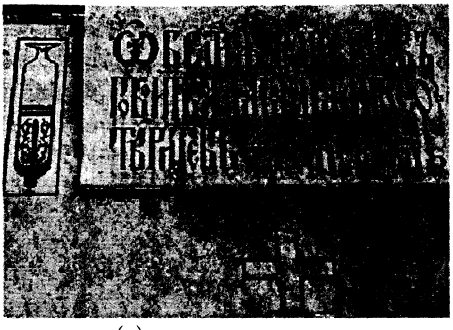
Опция	*Значение*	*По умолчанию*
-f*name*	имя рисунка в TEX'e	
-h*number*	разрешение по горизонтали (в пикселах на дюйм)	300
-v*number*	разрешение по вертикали	300
-l*number*	длина строки изображения в байтах (только для чисто растровых изображений)	
-p*y-or-n*	высота генерируемых символов равна 1 дюйму?	y
-e*y-or-n*	растянуть рисунки для EGA?	y
-i*y-or-n*	инвертировать пикселы?	n
-g*y-or-n*	использовать серые пикселы в растровом рисунке?	n
-w*y-or-n*	разрешить белому стать светло-серым?	y
-d*y-or-n*	выводить ошибки?	y
-s*y-or-n*	разделять серые точки?	n
-r*y-or-n*	повторять каждый серый пиксел?	n
-u*number*	(число) пикселов для серого прямоугольника	8
-c*number*	(число) вертикальных пикселов для прямоугольника	8
-x*number*	(число) битов на пиксел	$0 < x < 9$
-b*number*	уменьшить насыщенность полутоновых цветов	0
-t*number*	величина градации в %	70
-z*number*	площадь градации в %	70
-m*number*	ширина рисунка на бумаге	в mm
-n*number*	высота рисунка на бумаге	в mm
-j*y-or-n*	отсечь белое пространство?	y
-k*character*	цвет: c, m, y или k	k

единяются по «И» с пикселами, которые уже имеются в этой области, а впоследствии накладываемые текстовые пикселы объединяются по «ИЛИ». Таким образом, текст в выделенной области, *после которого* следует команда «`sh 0`», становится невидимым. Любая штриховка со значением s больше нуля добавляет серый фон к существующему тексту, оставляя его видимым (конечно, при $s = 1$ фон станет совсем черным и текст перестанет быть различимым).

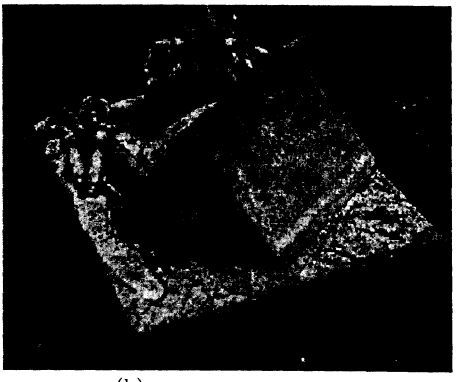
Замечание: команда `sh` не обязана поддерживаться всеми `dvi`-драйверами; драйверы, которые не поддерживают ее, должны просто игнорировать эту команду.

Следующие три команды вышли из употребления, но, возможно, должны поддерживаться с целью совместимости:

`wh` То же, что и «`sh 0`».
`bk` То же, что и «`sh 1`».



(a) опции не указаны



(b) опции не указаны

(c) опции `-b4 -gy`

(d) опция `-iy`

Рис. А.1. Пример использования программы bm2font.

`tx` Эта команда указывает текстуру, которая будет использоваться при штриховке вместо той, которая применяется по умолчанию. Известен только один драйвер, который использует ее. Новые разработки драйверов должны игнорировать эту команду.

А.3 Программа bm2font

Опции программы bm2font, указываемые в командной строке, перечислены в табл. А.2. Влияние трех из них («уменьшить насыщенность», «разрешить использовать серые пикселы» и «инверсия пикселов») показано на рис. А.1. Существенным недостатком программы является то, что выходные файлы становятся чрезвычайно большими и используют слишком много TEX'овских шрифтов, когда вы пытаетесь работать с разрешением фотонаборных машин (выше 600 точек на дюйм). Приведенные здесь примеры были сгенерированы для разрешения 600 dpi.

А.4 Заголовочный файл для цветоделения в dvips

Ниже приведено содержимое файла `colorsep.pro`, распространяемого совместно с dvips, в котором переопределяются операторы цвета PostScript с тем, чтобы осуществить цветоделение в соответствии с системой CMYK. Файл начинается с определения некоторых полезных команд:

```
%!
/seppages  0  def
userdict begin
/Min {% 3 items on stack, find the smallest
2 copy lt { pop }{ exch pop } ifelse
2 copy lt { pop }{ exch pop } ifelse
} def
```

Нам нужно преобразовать цвет, заданный в виде градаций серого или в цветовых моделях HSB или RGB, в модель CMYK, в которой используется четыре цвета. Решение для градаций серого не составляет труда, а HSB является просто одним из способов выразить RGB:

```
/SetGray {
 1 exch sub systemdict begin adjustdot setgray end
} def
/sethsbcolor {systemdict begin
  sethsbcolor currentrgbcolor end
  userdict begin setrgbcolor end}def
```

Обработка самого RGB несколько сложнее. В описании Adobe Systems (1990) на с. 305 говорится: для преобразования RGB в CMYK следует вычесть интенсивность каждой составляющей цвета из 1 и рассчитать наименьший уровень черного:

```
/ToCMYK
  {
  3 { 1 exch sub 3 1 roll } repeat
  3 copy  Min
  blackUCR sub
  dup 0 lt {pop 0} if
  /percent_UCR exch def
```

Затем следует вычесть этот черный «фон» из каждого из цветов и обработать сам черный цвет:

```
  3 { percent_UCR sub 3 1 roll } repeat
  percent_UCR 1.25 mul % 1 exch sub
} def
```

В результате в PostScript'овском стеке теперь находятся значения C, M, Y и K.

Команда `CM` печатает границы обрезки листа:

```
/cX 18 def
/CM{gsave TR 0 cX neg moveto 0 cX lineto stroke
cX neg 0 moveto cX 0 lineto stroke grestore}def
```

Критической частью файла является определение оператора **bop-hook**, который исполняется в начале каждой из четырех страниц, на которые выводится соответствующий цвет. Ниже мы предполагаем, что четырехкратная печать каждой страницы задана с помощью опции **-b 4** программы dvips:

```
/bop-hook{cX dup TR
   seppages 1 add
    /seppages exch def
     seppages 5 eq { /seppages  1  def } if
     seppages 1 eq {
      /ColorName (CYAN) def
      CYAN setupcolor
      /WhichColor 3 def } if
   seppages 2 eq {
      /ColorName (MAGENTA) def
      MAGENTA setupcolor
     /WhichColor 2 def } if
   seppages 3 eq {
      /ColorName (YELLOW) def
      YELLOW setupcolor
      /WhichColor 1 def } if
   seppages 4 eq {
      /ColorName (black) def
      black setupcolor
      /WhichColor 0 def } if
```

Устанавливаем границы обрезки на каждой странице с указанием названия компоненты цвета:

```
gsave .3 setlinewidth
3 -7 moveto
/Helvetica findfont 6 scalefont setfont
ColorName show
0 0 CM
vsize cX 2 mul sub dup hsize cX 2 mul sub dup isls{4 2 roll}if 0 CM
exch CM 0
exch CM
grestore 0 cX -2 mul TR isls
{cX -2 mul 0 TR}if
          } def end
/separations 48 dict def
separations begin
   /cmykprocs [
    { pop pop  pop SetGray  }         % cyan
    { pop pop exch pop SetGray  }     % magenta
    { pop 3 1 roll pop pop SetGray  } % yellow
    { 4 1 roll pop pop pop SetGray  } % black
   ] def
   /rgbprocs [ %def
    { ToCMYK pop pop pop SetGray }
    { ToCMYK pop pop exch pop SetGray }
    { ToCMYK pop 3 1 roll pop pop SetGray }
    { ToCMYK 4 1 roll pop pop pop SetGray  }
```

```
    ] def
    /screenangles [ %def
        105  % cyan
        75   % magenta
        0    % yellow
        45   % black
    ] def
end
```

Команда `setupcolor` принимает значение аргумента, равное 0, 1, 2 или 3 для голубого, пурпурного, желтого и черного цветов соответственно:

```
/CYAN 0 def             /MAGENTA 1 def
/YELLOW 2 def           /black 3 def
/setupcolor{ %def
   userdict begin
       dup separations /cmykprocs get exch get
       /setcmykcolor exch def
       dup separations /rgbprocs get exch get
       /setrgbcolor exch def
       separations /screenangles get exch get
       currentscreen
           exch pop 3 -1 roll exch
       setscreen
       /setscreen { pop pop pop } def
```

Оператор `setgray` переопределен так, что показывает только долю черного цвета:

```
       /setgray {
        WhichColor 0 eq
        {systemdict begin adjustdot setgray end}
        {pop systemdict begin 1 setgray end}
        ifelse}def
    end
} bind def
/adjustdot { dup 0 eq { } { dup 1 exch sub .1 mul add} ifelse } def
```

Наконец, устанавливаем доли удаляемых основных цветов, что необходимо сделать для компенсации добавления черного:

```
/magentaUCR .3 def
/yellowUCR .07 def
/blackUCR .4 def
```

А.5 Каталог имен шрифтов и их сокращений в схеме *Fontname*

В этом каталоге приведены используемые в схеме *Fontname* аббревиатуры для большого числа семейств шрифтов; в скобках указаны рекомендуемые восьмибуквенные имена директорий. Когда один и тот же шрифт выпускается

разными производителями под различными именами, в каталоге перед именем шрифта указано сокращенное имя производителя (согласно табл. 10.4 на с. 445), отделенное от имени шрифта двоеточием.

a1 (arial) Arial
a2 (amasis) Amasis
a3 (adlib) AdLib
a4 (altgoth2) AlternateGothicNo2
a5 (allegro) Allegro
a6 (amelia) Amelia
a7 (amertext) AmericanText
a8 (aurora) Aurora
a9 (adsans) Adsans
b:Humanist970BT
aa (aachen) Aachen
ab (aboeckli) ArnoldBoecklin
ac (acaslon) ACaslon
ad (agaramon) AGaramond
ag (avantgar) AvantGarde
m:MAvantGardeGothicPS
u:URWGothicL
ah (ashleysc) AshleyScript
ai (abadi) Abadi
al (albertus) Albertus
b:Flareserif821BT
am (amercana) Americana
b:Americana
an (anna) Anna
ao (atqolive) AntiqueOlive
b:Incised901BT
ap (apollo) Apollo
aq (antiqua) Antiqua
u:URWAntiqua
m:BookAntiqua
ar (arcadia) Arcadia
as (aldus) Aldus
at (atypewri) AmericanTypewriter
au (auriol) Auriol
b:Freeform721
av (avenir) Avenir
az (azdzgrot) AkzidenzGrotesk
b:Gothic725BT

b0 (blackoak) Blackoak
b1 (blado) Blado
b2 (binnyos) BinnyOldStyle
b3 (braggado) Braggadocio
b4 (bison) Bison
b:Brush738BT
b5 (binghams) BinghamScript
b:Freehand591BT
b6 (block) Block
b:Gothic821
b7 (bernhard) Bernhard BernhardModern
b8 (blippo) Blippo
b9 (bremen) Bremen
bb (bembo) Bembo
b:Aldine401BT
bc (banco) Banco
bd (bodoni) Bodoni
be (bell) Bell
bf (biffo) Biffo
bg (benguiat) Benguiat
bh (bauhaus) Bauhaus
bi (birch) Birch
bk (bookman) Bookman
m:MBookmanPS
u:URWBookmanL
bl (belwe) Belwe
bm (bulmer) Bulmer
bn (bernard) Bernard
bo (balloon) Balloon
bp (bodocamp) BodoniCampanile
b:Modern735
br (berling) Berling
b:Revival565BT
bs (bakersig) BakerSignet
bt (boton) Boton
bu (brush) Brush
bv (baskervi) Baskerville
bw (broadway) Broadway
by (berkeley) Berkeley

c0 (carolina) Carolina
c1 (calisto) Calisto
c2 (casln224) CaslonTwoTwentyFour
c3 (caslon3) CaslonThree
c4 (chicago) Chicago
c5 (casln540) CaslonFiveForty
c6 (clarion) Clarion
c7 (compacta) Compacta
c8 (calvert) Calvert
c9 (codex) Codex
b:Calligraphic421BT
ca (caslon) Caslon
cb (cooper) Cooper
cc (cascade) Cascade
b:Freehand471BT
cd (clarendo) Clarendon
ce (centenni) Centennial
cf (clearfac) Clearface
cg (charlema) Charlemagne
ch (charter) Charter
ci (candida) Candida
cl (cloister) Cloister
cm (cm) Computer Modern
cn (century) Century
co (cochin) Cochin
cp (copperpl) Copperplate
cq (cheq) Cheq
cr (courier) Courier
m:MCourierPS
m:CourierTwelveMT
u:NimbusMonL
cs (cntursbk) CenturySchoolbook
ct (cheltenh) Cheltenham
cu (centuros) CenturyOldStyle
cv (clairvx) Clairvaux
cw (cottonwd) Cottonwood
cx (caxton) Caxton

cy (city) City
b:SquareSlabserif711BT
cz (choc) Choc
b:Staccato555BT
d0 (davida) DavidaBold
dc (domcasl) Dom
DomCasual
DomDiagonal
dd (ducberry) DucDeBerry
de (dellarob) DellaRobbia
di (diotima) Diotima
b:Calligraphic810BT
dl (dolores) Dolores
dm (delima) Delima
dr (doric) Doric
ds (dorchesc)
DorchesterScript
dt (dante) Dante
e0 (embassy) Embassy
e1 (empire) Empire
ea (lectra) Electra
b:Transitional521BT
eb (baskerbe)
BaskervilleBE
ec (applecha)
e:AppleChancery
ee (egyptian) Egyptian
EgyptianExtended
ef (egyptien) EgyptienneF
eg (stgaramo)
StempelGaramond
b:OriginalGaramondBT
eh (engsft) Engschrift
ei (esprit) Esprit
ek (eckmann) Eckmann
b:Freeform710
el (ellingtn) Ellington
en (engraver) Engravers
eo (amerigo) Amerigo
b:AmerigoBT
er (eras) Eras
es (eurostil) Eurostile
b:Square721BT
k:Minima
et (ehrhardt) Ehrhardt
eu (euler) Euler
ex (exclsior) Excelsior
f0 (flemishs)
FlemishScript
f1 (fournier) Fournier
f2 (facade) Facade
f3 (festival) Festival
f4 (footligh) Footlight
f5 (french) French
fa (formata) Formata
fe (forte) Forte
ff (fairfld) Fairfield
b:Transitional551BT
fg (frankgth)
FranklinGothic
fi (figaro) Figaro
fl (flora) Flora
fn (fenice) Fenice
fo (folio) Folio
fq (frizquad)
FrizQuadrata
fr (frutiger) Frutiger
b:Humanist777BT
fs (falstaff) Falstaff
ft (fettefrk) FetteFraktur
Fraktur
fu (futura) Futura
fx (felixttl) FelixTitling
fy (flyer) Flyer
g0 (gando) Gando
g1 (gothic13)
GothicThirteen
g2 (giddyup) Giddyup
g3 (garamon3)
GaramondThree
b:AmercianGaramondBT
g4 (gymodern)
GoudyModern
g5 (gycatalg)
GoudyCatalogue
g6 (gorilla) GorillaITC
g9 (ugaramon)
GaramondRetrospective
gb (gothicbb) GothicBBB
ge (geneva) e:Geneva
gg (garthgra)
GarthGraphic
gj (granjon) Granjon
b:ElegantGaramondBT
gl (galliard) Galliard
gm (garamond) Garamond
go (gyoldsty)
GoudyOldStyle
gp (glypha) Glypha
gq (grotesq) Grotesque
u:URWGrotesk
gr (gloucest) Gloucester
gs (gillsans) GillSans
m:GillAltOneMT
b:Humanist521BT
gt (goudytxt) GoudyText
gu (guardi) Guardi
gv (giovanni) Giovanni
gy (goudy) Goudy
h0 (handelgo)
HandelGothic
hb (hobo) Hobo Tramp
hc (herculnm)
Herculanum
hd (headline) Headline
hg (hiroshig) Hiroshige
hi (hvinsera)
HelveticaInserat
b:Swiss921BT
hm (charme) Charme
hn (hvneue) HelveticaNeue
ho (horleyos)
HorleyOldStyle
hr (hvround)
HelveticaRounded
hs (hnseati) Hanseatic
B:Swiss924BT
ht (hoeflert) e:HoeflerText
hv (helvetic) Helvetica
u:NimbusSans
u:NimbusSanL
b:Swiss721BT
b:Monospace821BT
b:Swiss911BT d:Sans
Newton
n:SonoranSansSerif
hw (schadow) Schadow
hx (huxleyvr)
HuxleyVertical
i0 (commscr)
CommercialScript
i1 (impress) Impress
i2 (impuls) Impuls
ia (insignia) Insignia
ib (ibycus) Ibycus
ic (impact) Impact
if (inflex) Inflex

ig (scgaramo) SimonciniGaramond b:ItalianGaramond
ii (imprint) Imprint
il (caecilia) Caecilia
im (imago) Imago
in (industri) Industria
io (ionic) Ionic b:News701BT
ip (impressm) Impressum
is (italnos) ItalianOldStyle
it (italia) Italia
iw (ironwood) Ironwood
iy (iowanold) IowanOldStyle
jf (jeffersn) Jefferson b:Freehand575BT
jn (janson) Janson
jo (joanna) Joanna
jp (juniper) Juniper
kb (kabel) Kabel
kg (bankgoth) BankGothic
kl (klang) Klang
kn (kino) Kino
ko (bookmano) BookmanOldStyle
kr (korinna) Korinna
ks (kuenstsc) KuenstlerScript
ku (kaufmann) Kaufmann
l1 (liberty) Liberty
l2 (libra) Libra
l3 (lucia) Lucia
l4 (lucian) Lucian
lb (lubaling) LubalinGraph
lc (lucida) Lucida
lf (life) Life
lg (lettrgth) LetterGothic ISILGothic
lh (lubright) LucidaBright
li (latin) Latin
ln (lino) Lino
lo (lithos) Lithos
ls (lucsans) LucidaSans
lt (lutetia) Lutetia
lw (leawood) Leawood
lx (lucidfax) LucidaFax
ly (lydian) Lydian
m0 (monospac) Monospace
m1 (monolisc) MonolineScript
m2 (metro) Metro b:Geometric415BT
m3 (mirarae) Mirareae
m4 (mrearl) MisterEarl
m5 (murrayhl) MurrayHill
m6 (mittlsft) Mittelschrift
m7 (matura) Matura
m8 (monotyos) MonotypeOldStyle
ma (mandate) Mandate b:Freehand521BT
mc (monaco) e:Monaco
md (meridien) Meridien
me (mercursc) MercuriusScript
mg (marigold) Marigold
mh (machine) Machine
mi (medici) Medici
ml (melior) Melior b:ZapfElliptical711BT
mn (minion) Minion
mo (modern) Modern b:Modern20BT
mp (memphis) Memphis b:GeometricSlab703BT
mq (mesquite) Mesquite
mr (madrone) Madrone
ms (mistral) Mistral b:Staccato222BT
mt (minister) Minister
mv (malvern) Malvern
mx (maximus) Maximus
mz (amazone) Amazone
n0 (normande) Normande
n1 (binner) Binner
na (newaster) NewAster
nb (nbaskerv) NewBaskerville
nc (ncntrsbk) NewCenturySchlbk u:CenturySchL
nd (ncaledon) NewCaledonia
nf (nofret) Nofret
ng (newsgth) NewsGothic
ni (nimrod) Nimrod
nk (neograph) Neographik
nl (neuland) Neuland b:Informal011BT
nm (nimbus) Nimbus
nn (nclarend) NewClarendon
np (nsplanti) NewsPlantin
nr (nberolin) NewBerolina
ns (timesnps) TimesNewRomanPS
nt (timesnew) TimesNewRoman m:TimesNR n:SonoranSerif
nu (nuptial) Nuptial
nv (novarese) Novarese
nz (neuzeits) NeuzeitS b:Geometric706BT
o0 (oldreadf) OldDreadfulNo7
o1 (oranda) Oranda
o2 (orbitb) OrbitB
o3 (colonna) Colonna
o4 (oceansan) OceanSans
o7 (oldsty7) OldStyleSeven
oa (ocra) OCR-A
ob (ocrb) OCR-B
oc (concorde) Concorde
oe (oldengli) OldEnglish b:English157
oi (ondine) Ondine Mermaid b:FormalScript421BT
ol (colmcill) Colmcille
om (omnia) Omnia
on (corona) Corona
op (optima) Optima d:Optimum b:ZapfHumanist601BT Omega
or (orator) Orator
os (oldstyle) OldStyle
ot (coronet) Coronet b:Ribbon131BT
ov (octavian) Octavian
ox (onyx) Onyx

oy (olympian) Olympian
l:OlympicLH
oz (ozhandcr)
OzHandicraft
p0 (poplar) Poplar
p1 (palacesc) PalaceScript
p2 (pepita) Pepita
p3 (palette) Palette
b:Brush445BT
p4 (pioneer) PioneerITC
p5 (piranesi) Piranesi
p6 (blueprin) Blueprint
pa (parkave) ParkAvenue
pb (ptbarnum) PtBarnum
pc (poetica)
PoeticaChancery
s:Scriptoria
pd (placard) Placard
pe (prestige) PrestigeElite
pf (profile) Profile Profil
b:Decorated035BT
pg (peignot) Peignot
b:Exotic350BT
ph (photina) Photina
pi (pi) pi fonts
pl (palatino) Palatino
d:Palton
b:ZapfCalligraphic801BT
u:URWPalladioL
Palmspring
k:Palindrome
pn (plantin) Plantin
b:Aldine721BT
po (ponderos) Ponderosa
pp (perpetua) Perpetua
b:Lapidary333BT
pq (postatqa) PostAntiqua
ps (parisian) Parisian
pt (present) Present
pu (popplaud)
PopplLaudatio
pw (perrywoo) Perrywood
px (popplpfx)
PopplPontifex
py (playbill) Playbill
pz (poliphil) Poliphilus
qr (quorum) Quorum
qt (boulevar) Boulevard
r0 (romana) Romana
r1 (rotis) RotisSemi
r2 (cantoria) Cantoria
ra (raleigh) Raleigh
ro (rotis) Rotis
rm (carmina) Carmina
rp (reporter) Reporter
rq (russelsq)
RussellSquare
rs (arrus) Arrus
rt (carta) Carta
ru (runic) Runic
rv (revue) Revue
rw (rockwell)
RockwellSlate
b:GeometricSlab712BT
ry (ryumin) Ryumin
s0 (sanmarco) SanMarco
s1 (scotchro)
ScotchRoman
s2 (script) Script
s3 (seagull) Seagull
s4 (shotgun) Shotgun
ShotgunBlanks
s5 (snell) Snell
s6 (stuyvesa) Stuyvesant
s7 (sassoon) Sassoon
sb (sabon) Sabon
b:ClassicalGaramondBT
sc (slimbach) Slimbach
sf (serifa) Serifa
sg (serifgth) SerifGothic
sh (shannon) Shannon
sk (skia) e:Skia
sl (stencil) Stencil
sm (spectrum) Spectrum
sn (spartan) Spartan
so (sonata) Sonata
sp (serpenti) Serpentine
sr (snellrnd)
SnellRoundhand
ss (stschnei)
StempelSchneidler
st (stone) Stone
sv (souvenir) Souvenir
sw (swing) Swing
sx (syntax) Syntax
sy (symbol) Symbol
m:MSymbolPS
u:StandardSymL
t0 (tango) Tango
t1 (thunderb)
Thunderbird
t2 (typo) TypoUpright
tb (bebodoni) BBodoni
te (timeseur)
TimesEuropa
tf (tiffany) Tiffany
tg (tradegth) TradeGothic
ti (techphon)
TechPhonetic
tj (trajan) Trajan
tk (tekton) Tekton
tl (castella) Castellar
tm (times) Times
u:NimbusRoman
u:NimbusRomNo9L
b:Dutch801BT
to (torino) Torino
b:Industrial736BT
tp (tempo) Tempo
tv (trumpmed)
TrumpMediaeval
b:Kuenstler480BT
tw (twentyc)
TwentiethCentury
ty (typewrit) Typewriter
u0 (columbus) Columbus
ub (babodoni)
BauerBodoni
ug (bengugth)
BenguiatGothic
uh (cushing) Cushing
ul (universa) Universal
um (umbra) Umbra
un (univers) Univers
b:ZurichBT
ur (centaur) Centaur
b:Venetian301BT
ut (utopia) Utopia
uw (usherwoo) Usherwood
uy (universi) University
v0 (vineta) Vineta
va (activa) Activa
vc (vectora) Vectora
vd (vandijck) VanDijck
vi (victoria) Victoria
vj (veljovic) Veljovic
vl (versaill) Versailles

`vr` (vagrnd) VAGRounded
`w0` (weddingt) WeddingT
`wb` (walbaum) Walbaum
`wd` (weideman) Weidemann
`wg` (wlklgth) WilhelmKlingsporGotisch
`wi` (wingding) Wingdings
`wk` (wilke) Wilke
`wn` (windsor) Windsor
`wr` (wittenbe) WitternbergerFrakt
`ws` (weiss) Weiss
`ww` (willow) Willow
`yg` (centuryg) CenturyGothic
`yh` (strayhor) Strayhorn
`ym` (stymie) Stymie
`za` (zantiqua) ZAntiqua
`zc` (zapfchan) ZapfChancery
m:ZapfChanceryPS
u:URWChanceryL
`zd` (zapfding) ZapfDingbats
m:ZapfDingbatsPS
u:Dingbats
`zg` (neuzeitg) NeuzeitGrotesk
`zt` (zeitgeis) Zeitgeist

A.6 Таблицы кодировки шрифтов

Таблица A.3. Таблица кодировки шрифтов, упорядоченная по именам глифов.

Имя	Глиф	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
A	A	65	65	65	65	65	65	65
AE	Æ	198	198	198	225	198	174	198
Aacute	Á	193	193	193		193	231	193
Abreve	Ă	128						
Acircumflex	Â	194	194	194		194	229	194
Adieresis	Ä	196	196	196		196	128	196
Agrave	À	192	192	192		192	203	192
Aogonek	Ą	129						
Aring	Å	197	197	197		197	129	197
Atilde	Ã	195	195	195		195	204	195
B	B	66	66	66	66	66	66	66
C	C	67	67	67	67	67	67	67
Cacute	Ć	130						
Ccaron	Č	131						
Ccedilla	Ç	199	199	199		199	130	199
D	D	68	68	68	68	68	68	68
Dcaron	Ď	132						
E	E	69	69	69	69	69	69	69
Eacute	É	201	201	201		201	131	201
Ecaron	Ě	133						
Ecircumflex	Ê	202	202	202		202	230	202
Edieresis	Ë	203	203	203		203	232	203
Egrave	È	200	200	200		200	233	200
Eng	Ŋ	141						

Таблица А.3. *Продолжение.*

Имя	Глиф	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
Eogonek	Ę	134						
Eth	Ð	208	208	208		208		208
F	F	70	70	70	70	70	70	70
G	G	71	71	71	71	71	71	71
Gbreve	Ğ	135						
Germandbls	SS	223						
H	H	72	72	72	72	72	72	72
I	I	73	73	73	73	73	73	73
IJ	IJ	156						
Iacute	Í	205	205	205		205	234	205
Icircumflex	Î	206	206	206		206	235	206
Idieresis	Ï	207	207	207		207	236	207
Idotaccent	İ	157						
Igrave	Ì	204	204	204		204	237	204
J	J	74	74	74	74	74	74	74
K	K	75	75	75	75	75	75	75
L	L	76	76	76	76	76	76	76
Lacute	Ĺ	136						
Lquoteright	Ľ	137						
Lslash	Ł	138	6		232			149
M	M	77	77	77	77	77	77	77
N	N	78	78	78	78	78	78	78
Nacute	Ń	139						
Ncaron	Ň	140						
Ntilde	Ñ	209	209	209		209	132	209
O	O	79	79	79	79	79	79	79
OE	Œ	215	140		234	140	206	150
Oacute	Ó	211	211	211		211	238	211
Ocircumflex	Ô	212	212	212		212	239	212
Odieresis	Ö	214	214	214		214	133	214
Ograve	Ò	210	210	210		210	241	210
Ohungarumlaut	Ő	142						
Oslash	Ø	216	216	216	233	216	175	216
Otilde	Õ	213	213	213		213	205	213
P	P	80	80	80	80	80	80	80
Q	Q	81	81	81	81	81	81	81
R	R	82	82	82	82	82	82	82
Racute	Ŕ	143						

Таблица А.3. *Продолжение.*

Имя	Глиф	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
Rcaron	Ř	144						
S	S	83	83	83	83	83	83	83
Sacute	Ś	145						
Scaron	Š	146	138			138		151
Scedilla	Ş	147						
T	T	84	84	84	84	84	84	84
Tcaron	Ť	148						
Tcedilla	Ţ	149						
Thorn	Þ	222	222	222		222		222
U	U	85	85	85	85	85	85	85
Uacute	Ú	218	218	218		218	242	218
Ucircumflex	Û	219	219	219		219	243	219
Udieresis	Ü	220	220	220		220	134	220
Ugrave	Ù	217	217	217		217	244	217
Uhungarumlaut	Ű	150						
Uring	Ů	151						
V	V	86	86	86	86	86	86	86
W	W	87	87	87	87	87	87	87
X	X	88	88	88	88	88	88	88
Y	Y	89	89	89	89	89	89	89
Yacute	Ý	221	221	221		221		221
Ydieresis	Ÿ	152	159			159	217	152
Z	Z	90	90	90	90	90	90	90
Zacute	Ź	153						
Zcaron	Ž	154	14					153
Zdotaccent	Ż	155						
a	a	97	97	97	97	97	97	97
aacute	á	225	225	225		225	135	225
abreve	ă	160						
acircumflex	â	226	226	226	ˊ	226	137	226
acute	´	1	180	180	194	180	171	180
adieresis	ä	228	228	228		228	138	228
ae	æ	230	230	230	241	230	190	230
agrave	à	224	224	224		224	136	224
ampersand	&	38	38	38	38	38	38	38
aogonek	ą	161						
aring	å	229	229	229		229	140	229
asciicircum	^	94	94	94	94	94	94	94

Таблица А.3. *Продолжение.*

Имя	Глиф	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
asciitilde	~	126	126	126	126	126	126	126
asterisk	*	42	42	42	42	42	42	42
at	@	64	64	64	64	64	64	64
atilde	ã	227	227	227		227	139	227
b	b	98	98	98	98	98	98	98
backslash	\	92	92	92	92	92	92	92
bar	|	124	124	124	124	124	124	124
blank							202	
braceleft	{	123	123	123	123	123	123	123
braceright	}	125	125	125	125	125	125	125
bracketleft	[	91	91	91	91	91	91	91
bracketright	]	93	93	93	93	93	93	93
breve	˘	8	11	150	198	24	249	24
brokenbar	¦		166	166		166		166
bullet	•		149		183		165	
c	c	99	99	99	99	99	99	99
cacute	ć	162						
caron	ˇ	7	16	159	207	25	255	25
ccaron	č	163						
ccedilla	ç	231	231	231		231	141	231
cedilla	¸	11	184	184	203	184	252	184
cent	¢		162	162	162	162	162	162
circumflex	ˆ	2	136	147	195	136	246	26
colon	:	58	58	58	58	58	58	58
comma	,	44	44	44	44	44	44	44
compoundwordmark		23						
copyright	©		169	169		169	169	169
currency	¤		164	164	168	164	219	164
d	d	100	100	100	100	100	100	100
dagger	†		134		178	134	160	129
daggerdbl	‡		135		179	135	224	130
dbar	đ	158						
degree	°		176	176		176	161	176
dieresis	¨	4	168	168	200	168	172	168
divide	÷		247	247		247	214	247
dollar	$	36	36	36	36	36	36	36
dotaccent	˙	10	1	151	199	27	250	27
dotlessi	ı	25	17	144	245		245	154

Таблица A.3. *Продолжение.*

Имя	Глиф	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
dotlessj	ȷ	26	18					
dquoteright	”	164						
e	e	101	101	101	101	101	101	101
eacute	é	233	233	233		233	142	233
ecaron	ě	165						
ecircumflex	ê	234	234	234		234	144	234
edieresis	ë	235	235	235		235	145	235
egrave	è	232	232	232		232	143	232
eight	8	56	56	56	56	56	56	56
ellipsis	...		133		188	133	201	131
emdash	—	22	151		208	151	209	132
endash	–	21	150		177	150	208	133
eng	ŋ	173						
eogonek	ę	166						
equal	=	61	61	61	61	61	61	61
eth	ð	240	240	240		240		240
exclam	!	33	33	33	33	33	33	33
exclamdown	¡	189	161	161	161	161	193	161
f	f	102	102	102	102	102	102	102
ff	ff	27	19					
ffi	ffi	30	20					
ffl	ffl	31	21					
fi	fi	28	2		174		222	147
five	5	53	53	53	53	53	53	53
fl	fl	29	3		175		223	148
florin	f		131		166	131	196	134
four	4	52	52	52	52	52	52	52
fraction	/		4		164		218	135
g	g	103	103	103	103	103	103	103
gbreve	ğ	167						
germandbls	ß	255	223	223	251	223	167	223
grave	`	0	30	145	193		96	96
greater	>	62	62	62	62	62	62	62
guillemotleft	«	19	171	171	171	171	199	171
guillemotright	»	20	187	187	187	187	200	187
guilsinglleft	‹	14	139		172	139	220	136
guilsinglright	›	15	155		173	155	221	137
h	h	104	104	104	104	104	104	104

Таблица А.3. *Продолжение.*

Имя	Глиф	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
hungarumlaut	˝	5	5	157	205	28	253	28
hyphen	-	127	173		45	173	45	45
i	i	105	105	105	105	105	105	105
iacute	í	237	237	237		237	146	237
icircumflex	î	238	238	238		238	148	238
idieresis	ï	239	239	239		239	149	239
igrave	ì	236	236	236		236	147	236
ij	ij	188						
j	j	106	106	106	106	106	106	106
k	k	107	107	107	107	107	107	107
l	l	108	108	108	108	108	108	108
lacute	Í	168						
less	<	60	60	60	60	60	60	60
logicalnot	¬		172	172		172	194	172
lquoteright	ľ	169						
lslash	ł	170	7		248			155
m	m	109	109	109	109	109	109	109
macron	¯	9	175	175	197	175	248	175
minus	−		12	45				138
mu	µ		181	181		181	181	181
multiply	×		215	215		215		215
n	n	110	110	110	110	110	110	110
nacute	ń	171						
nbspace				160				
ncaron	ň	172						
nine	9	57	57	57	57	57	57	57
ntilde	ñ	241	241	241		241	150	241
numbersign	#	35	35	35	35	35	35	35
o	o	111	111	111	111	111	111	111
oacute	ó	243	243	243		243	151	243
ocircumflex	ô	244	244	244		244	153	244
odieresis	ö	246	246	246		246	154	246
oe	œ	247	156		250	156	207	156
ogonek	˛	12	8	158	206	29	254	29
ograve	ò	242	242	242		242	152	242
ohungarumlaut	ő	174						
one	1	49	49	49	49	49	49	49
onehalf	½		189	189		189		189

Таблица A.3. *Продолжение.*

Имя	Глиф	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
onequarter	¼		188	188		188		188
onesuperior	¹		185	185		185		185
ordfeminine	ª		170	170	227	170	187	170
ordmasculine	º		186	186	235	186	188	186
oslash	ø	248	248	248	249	248	191	248
otilde	õ	245	245	245		245	155	245
p	p	112	112	112	112	112	112	112
paragraph	¶		182	182	182	182	166	182
parenleft	(	40	40	40	40	40	40	40
parenright	)	41	41	41	41	41	41	41
percent	%	37	37	37	37	37	37	37
period	.	46	46	46	46	46	46	46
periodcentered	·		183	183	180	183	225	183
perthousand	‰	24	137		189	137	228	139
plus	+	43	43	43	43	43	43	43
plusminus	±		177	177		177	177	177
q	q	113	113	113	113	113	113	113
question	?	63	63	63	63	63	63	63
questiondown	¿	190	191	191	191	191	192	191
quotedbl	"	34	34	34	34	34	34	34
quotedblbase	„	18	132		185	132	227	140
quotedblleft	“	16	147		170	147	210	141
quotedblright	”	17	148		186	148	211	142
quoteleft	‘	96	145	96	96	145	212	143
quoteright	’	39	146	39	39	146	213	144
quotesinglbase	‚	13	130		184	130	226	145
quotesingle	'		31		169	39	39	39
r	r	114	114	114	114	114	114	114
racute	ŕ	175						
rcaron	ř	176						
registered	®		174	174		174	168	174
ring	˚	6	9	154	202	30	251	30
s	s	115	115	115	115	115	115	115
sacute	ś	177						
scaron	š	178	154			154		157
scedilla	ş	179						
section	§	159	167	167	167	167	164	167
semicolon	;	59	59	59	59	59	59	59

Таблица А.3. *Продолжение.*

Имя	Глиф	EC	8r	ISO Latin 1	Adobe Standard	Windows ANSI	Mac	PDF
seven	7	55	55	55	55	55	55	55
sfthyphen				173				
six	6	54	54	54	54	54	54	54
slash	/	47	47	47	47	47	47	47
space			32	32	32	160	32	32
sterling	£	191	163	163	163	163	163	163
t	t	116	116	116	116	116	116	116
tcedilla	ţ	181						
thorn	þ	254	254	254		254		254
three	3	51	51	51	51	51	51	51
threequarters	¾		190	190		190		190
threesuperior	³		179	179		179		179
tilde	~	3	152	148	196	152	247	31
tquoteright	ť	180						
trademark	™		153			153	170	146
two	2	50	50	50	50	50	50	50
twosuperior	²		178	178		178		178
u	u	117	117	117	117	117	117	117
uacute	ú	250	250	250		250	156	250
ucircumflex	û	251	251	251		251	158	251
udieresis	ü	252	252	252		252	159	252
ugrave	ù	249	249	249		249	157	249
uhungarumlaut	ű	182						
underscore	_	95	95	95	95	95	95	95
uring	ů	183						
v	v	118	118	118	118	118	118	118
visiblespace	␣	32						
w	w	119	119	119	119	119	119	119
x	x	120	120	120	120	120	120	120
y	y	121	121	121	121	121	121	121
yacute	ý	253	253	253		253		253
ydieresis	ÿ	184	255	255		255	216	255
yen	¥		165	165	165	165	180	165
z	z	122	122	122	122	122	122	122
zacute	ź	185						
zcaron	ž	186	15					158
zdotaccent	ż	187						
zero	0	48	48	48	48	48	48	48

Таблица А.4. Таблица кодировки шрифтов, упорядоченная по номерам символов.

Десятичный	Восьмеричный	Шестнадцатеричный	Глиф в EC	EC	TeXBase1	ISO Latin 1	Standard	Windows ANSI	Mac Roman	PDF
0	0	0	`	grave						
1	1	1	´	acute	dotaccent					
2	2	2	ˆ	circumflex	fi					
3	3	3	˜	tilde	fl					
4	4	4	¨	dieresis	fraction					
5	5	5	˝	hungarumlaut	hungarumlaut					
6	6	6	°	ring	Lslash					
7	7	7	ˇ	caron	lslash					
8	10	8	˘	breve	ogonek					
9	11	9	¯	macron	ring					
10	12	a	˙	dotaccent						
11	13	b	¸	cedilla	breve					
12	14	c	˛	ogonek	minus					
13	15	d	,	quotesinglbase						
14	16	e	‹	guilsinglleft	Zcaron					
15	17	f	›	guilsinglright	zcaron					
16	20	10	“	quotedblleft	caron					
17	21	11	”	quotedblright	dotlessi					
18	22	12	„	quotedblbase	dotlessj					
19	23	13	«	guillemotleft	ff					
20	24	14	»	guillemotright	ffi					
21	25	15	–	endash	ffl					
22	26	16	—	emdash						
23	27	17		compoundwordmark						
24	30	18	˚	perthousand				breve		breve
25	31	19	ı	dotlessi				caron		caron
26	32	1a	ȷ	dotlessj				circumflex		circumflex
27	33	1b	ff	ff				dotaccent		dotaccent
28	34	1c	fi	fi				hungarumlaut		hungarumlaut

Таблица А.4. *Продолжение.*

Десятичный	Восьмеричный	Шестнадцатеричный	Глиф в EC	EC	TeXBase1	ISO Latin 1	Standard	Windows ANSI	Mac Roman	PDF
29	35	1d	fl	fl				ogonek		ogonek
30	36	1e	ffi	ffi	grave			ring		ring
31	37	1f	ffl	ffl	quotesingle			tilde		tilde
32	40	20	␣	visiblespace	space	space	space	space	space	space
33	41	21	!	exclam	exclam	exclam	exclam	exclam	exclam	exclam
34	42	22	"	quotedbl	quotedbl	quotedbl	quotedbl	quotedbl	quotedbl	quotedbl
35	43	23	#	numbersign	numbersign	numbersign	numbersign	numbersign	numbersign	numbersign
36	44	24	$	dollar	dollar	dollar	dollar	dollar	dollar	dollar
37	45	25	%	percent	percent	percent	percent	percent	percent	percent
38	46	26	&	ampersand	ampersand	ampersand	ampersand	ampersand	ampersand	ampersand
39	47	27	’	quoteright	quoteright	quoteright	quoteright	quotesingle	quotesingle	quotesingle
40	50	28	(	parenleft	parenleft	parenleft	parenleft	parenleft	parenleft	parenleft
41	51	29	)	parenright	parenright	parenright	parenright	parenright	parenright	parenright
42	52	2a	*	asterisk	asterisk	asterisk	asterisk	asterisk	asterisk	asterisk
43	53	2b	+	plus	plus	plus	plus	plus	plus	plus
44	54	2c	,	comma	comma	comma	comma	comma	comma	comma
45	55	2d	-	hyphen	hyphen	minus	hyphen	hyphen	hyphen	hyphen
46	56	2e	.	period	period	period	period	period	period	period
47	57	2f	/	slash	slash	slash	slash	slash	slash	slash
48	60	30	0	zero	zero	zero	zero	zero	zero	zero
49	61	31	1	one	one	one	one	one	one	one
50	62	32	2	two	two	two	two	two	two	two
51	63	33	3	three	three	three	three	three	three	three
52	64	34	4	four	four	four	four	four	four	four
53	65	35	5	five	five	five	five	five	five	five
54	66	36	6	six	six	six	six	six	six	six
55	67	37	7	seven	seven	seven	seven	seven	seven	seven
56	70	38	8	eight	eight	eight	eight	eight	eight	eight
57	71	39	9	nine	nine	nine	nine	nine	nine	nine

58	72	3a	:	colon	colon	colon	colon	colon	colon	colon
59	73	3b	;	semicolon	semicolon	semicolon	semicolon	semicolon	semicolon	semicolon
60	74	3c	<	less	less	less	less	less	less	less
61	75	3d	=	equal	equal	equal	equal	equal	equal	equal
62	76	3e	>	greater	greater	greater	greater	greater	greater	greater
63	77	3f	?	question	question	question	question	question	question	question
64	100	40	@	at	at	at	at	at	at	at
65	101	41	A	A	A	A	A	A	A	A
66	102	42	B	B	B	B	B	B	B	B
67	103	43	C	C	C	C	C	C	C	C
68	104	44	D	D	D	D	D	D	D	D
69	105	45	E	E	E	E	E	E	E	E
70	106	46	F	F	F	F	F	F	F	F
71	107	47	G	G	G	G	G	G	G	G
72	110	48	H	H	H	H	H	H	H	H
73	111	49	I	I	I	I	I	I	I	I
74	112	4a	J	J	J	J	J	J	J	J
75	113	4b	K	K	K	K	K	K	K	K
76	114	4c	L	L	L	L	L	L	L	L
77	115	4d	M	M	M	M	M	M	M	M
78	116	4e	N	N	N	N	N	N	N	N
79	117	4f	O	O	O	O	O	O	O	O
80	120	50	P	P	P	P	P	P	P	P
81	121	51	Q	Q	Q	Q	Q	Q	Q	Q
82	122	52	R	R	R	R	R	R	R	R
83	123	53	S	S	S	S	S	S	S	S
84	124	54	T	T	T	T	T	T	T	T
85	125	55	U	U	U	U	U	U	U	U
86	126	56	V	V	V	V	V	V	V	V
87	127	57	W	W	W	W	W	W	W	W
88	130	58	X	X	X	X	X	X	X	X
89	131	59	Y	Y	Y	Y	Y	Y	Y	Y
90	132	5a	Z	Z	Z	Z	Z	Z	Z	Z
91	133	5b	[	bracketleft	bracketleft	bracketleft	bracketleft	bracketleft	bracketleft	bracketleft
92	134	5c	\	backslash	backslash	backslash	backslash	backslash	backslash	backslash
93	135	5d	]	bracketright	bracketright	bracketright	bracketright	bracketright	bracketright	bracketright
94	136	5e	^	asciicircum	asciicircum	asciicircum	asciicircum	asciicircum	asciicircum	asciicircum

Таблица А.4. *Продолжение.*

Десятичный	Восьмеричный	Шестнадцатеричный	Глиф в EC	EC	TeXBase1	ISO Latin 1	Standard	Windows ANSI
95	137	5f	_	underscore	underscore	underscore	underscore	underscore
96	140	60	‘	quoteleft	quoteleft	quoteleft	quoteleft	quoteleft
97	141	61	a	a	a	a	a	a
98	142	62	b	b	b	b	b	b
99	143	63	c	c	c	c	c	c
100	144	64	d	d	d	d	d	d
101	145	65	e	e	e	e	e	e
102	146	66	f	f	f	f	f	f
103	147	67	g	g	g	g	g	g
104	150	68	h	h	h	h	h	h
105	151	69	i	i	i	i	i	i
106	152	6a	j	j	j	j	j	j
107	153	6b	k	k	k	k	k	k
108	154	6c	l	l	l	l	l	l
109	155	6d	m	m	m	m	m	m
110	156	6e	n	n	n	n	n	n
111	157	6f	o	o	o	o	o	o
112	160	70	p	p	p	p	p	p
113	161	71	q	q	q	q	q	q
114	162	72	r	r	r	r	r	r
115	163	73	s	s	s	s	s	s
116	164	74	t	t	t	t	t	t
117	165	75	u	u	u	u	u	u
118	166	76	v	v	v	v	v	v
119	167	77	w	w	w	w	w	w
120	170	78	x	x	x	x	x	x
121	171	79	y	y	y	y	y	y
122	172	7a	z	z	z	z	z	z
123	173	7b	{	braceleft	braceleft	braceleft	braceleft	braceleft
124	174	7c	|	bar	bar	bar	bar	bar
125	175	7d	}	braceright	braceright	braceright	braceright	braceright
126	176	7e	~	asciitilde	asciitilde	asciitilde	asciitilde	asciitilde
127	177	7f	-	hyphen				
128	200	80	Ă	Abreve				
129	201	81	Ą	Aogonek				
130	202	82	Ć	Cacute	quotesinglbase			quotesinglbase
131	203	83	Č	Ccaron	florin			florin
132	204	84	Ď	Dcaron	quotedblbase			quotedblbase
133	205	85	Ě	Ecaron	ellipsis			ellipsis
134	206	86	Ę	Eogonek	dagger			dagger
135	207	87	Ğ	Gbreve	daggerdbl			daggerdbl
136	210	88	Ĺ	Lacute	circumflex			circumflex
137	211	89	Ľ	Lquoteright	perthousand			perthousand
138	212	8a	Ł	Lslash	Scaron			Scaron
139	213	8b	Ń	Nacute	guilsinglleft			guilsinglleft
140	214	8c	Ň	Ncaron	OE			OE
141	215	8d	Ŋ	Eng				
142	216	8e	Ő	Ohungarumlaut				
143	217	8f	Ŕ	Racute				
144	220	90	Ř	Rcaron		dotlessi		
145	221	91	Ś	Sacute	quoteleft	grave		quoteleft
146	222	92	Š	Scaron	quoteright	acute		quoteright
147	223	93	Ş	Scedilla	quotedblleft	circumflex		quotedblleft
148	224	94	Ť	Tcaron	quotedblright	tilde		quotedblright
149	225	95	Ţ	Tcedilla	bullet	macron		
150	226	96	Ű	Uhungarumlaut	endash	breve		endash
151	227	97	Ů	Uring	emdash	dotaccent		emdash
152	230	98	Ÿ	Ydieresis	tilde	dieresis		tilde
153	231	99	Ź	Zacute	trademark			trademark
154	232	9a	Ž	Zcaron	scaron	ring		scaron
155	233	9b	Ż	Zdotaccent	guilsinglright	cedilla		guilsinglright
156	234	9c	IJ	IJ	oe			oe
157	235	9d	İ	Idotaccent		hungarumlaut		
158	236	9e	đ	dbar		ogonek		
159	237	9f	§	section	Ydieresis	caron		Ydieresis
160	240	a0	ă	abreve		nbspace		space

Таблица А.4. *Продолжение.*

Десятичный	Восьмеричный	Шестнадцатеричный	Глиф в EC	EC	TeXBase1	ISO Latin 1	Standard	Windows ANSI	Mac Roman	PDF
161	241	a1	ą	aogonek	exclamdown	exclamdown	exclamdown	exclamdown	degree	exclamdown
162	242	a2	ć	cacute	cent	cent	cent	cent	cent	cent
163	243	a3	č	ccaron	sterling	sterling	sterling	sterling	sterling	sterling
164	244	a4	ď	dquoteright	currency	currency	fraction	currency	section	currency
165	245	a5	ě	ecaron	yen	yen	yen	yen	bullet	yen
166	246	a6	ę	eogonek	brokenbar	brokenbar	florin	brokenbar	paragraph	brokenbar
167	247	a7	ğ	gbreve	section	section	section	section	germandbls	section
168	250	a8	ĺ	lacute	dieresis	dieresis	currency	dieresis	registered	dieresis
169	251	a9	ľ	lquoteright	copyright	copyright	quotesingle	copyright	copyright	copyright
170	252	aa	ł	lslash	ordfeminine	ordfeminine	quotedblleft	ordfeminine	trademark	ordfeminine
171	253	ab	ń	nacute	guillemotleft	guillemotleft	guillemotleft	guillemotleft	acute	guillemotleft
172	254	ac	ň	ncaron	logicalnot	logicalnot	guilsinglleft	logicalnot	dieresis	logicalnot
173	255	ad	ŋ	eng	hyphen	sfthyphen	guilsinglright	hyphen	notequal	
174	256	ae	ő	ohungarumlaut	registered	registered	fi	registered	AE	registered
175	257	af	ŕ	racute	macron	macron	fl	macron	Oslash	macron
176	260	b0	ř	rcaron	degree	degree		degree	infinity	degree
177	261	b1	ś	sacute	plusminus	plusminus	endash	plusminus	plusminus	plusminus
178	262	b2	š	scaron	twosuperior	twosuperior	dagger	twosuperior	lessequal	twosuperior
179	263	b3	ş	scedilla	threesuperior	threesuperior	daggerdbl	threesuperior	greaterequal	threesuperior
180	264	b4	ť	tquoteright	acute	acute	periodcentered	acute	yen	acute
181	265	b5	ţ	tcedilla	mu	mu		mu	mu	mu
182	266	b6	ű	uhungarumlaut	paragraph	paragraph	paragraph	paragraph	partialdiff	paragraph
183	267	b7	ů	uring	periodcentered	periodcentered	bullet	periodcentered	summation	periodcentered
184	270	b8	ÿ	ydieresis	cedilla	cedilla	quotesinglbase	cedilla	product	cedilla
185	271	b9	ź	zacute	onesuperior	onesuperior	quotedblbase	onesuperior	pi	onesuperior
186	272	ba	ž	zcaron	ordmasculine	ordmasculine	quotedblright	ordmasculine	integral	ordmasculine
187	273	bb	ż	zdotaccent	guillemotright	guillemotright	guillemotright	guillemotright	ordfeminine	guillemotright
188	274	bc	ij	ij	onequarter	onequarter	ellipsis	onequarter	ordmasculine	onequarter
189	275	bd	¡	exclamdown	onehalf	onehalf	perthousand	onehalf	Omega	onehalf

190	276	be	¿	questiondown	threequarters	threequarters		threequarters	ae	threequarters
191	277	bf	£	sterling	questiondown	questiondown	questiondown	questiondown	oslash	questiondown
192	300	c0	À	Agrave	Agrave	Agrave		Agrave	questiondown	Agrave
193	301	c1	Á	Aacute	Aacute	Aacute	grave	Aacute	exclamdown	Aacute
194	302	c2	Â	Acircumflex	Acircumflex	Acircumflex	acute	Acircumflex	logicalnot	Acircumflex
195	303	c3	Ã	Atilde	Atilde	Atilde	circumflex	Atilde	radical	Atilde
196	304	c4	Ä	Adieresis	Adieresis	Adieresis	tilde	Adieresis	florin	Adieresis
197	305	c5	Å	Aring	Aring	Aring	macron	Aring	approxequal	Aring
198	306	c6	Æ	AE	AE	AE	breve	AE	Delta	AE
199	307	c7	Ç	Ccedilla	Ccedilla	Ccedilla	dotaccent	Ccedilla	guillemotleft	Ccedilla
200	310	c8	È	Egrave	Egrave	Egrave	dieresis	Egrave	guillemotright	Egrave
201	311	c9	É	Eacute	Eacute	Eacute		Eacute	ellipsis	Eacute
202	312	ca	Ê	Ecircumflex	Ecircumflex	Ecircumflex	ring	Ecircumflex	blank	Ecircumflex
203	313	cb	Ë	Edieresis	Edieresis	Edieresis	cedilla	Edieresis	Agrave	Edieresis
204	314	cc	Ì	Igrave	Igrave	Igrave		Igrave	Atilde	Igrave
205	315	cd	Í	Iacute	Iacute	Iacute	hungarumlaut	Iacute	Otilde	Iacute
206	316	ce	Î	Icircumflex	Icircumflex	Icircumflex	ogonek	Icircumflex	OE	Icircumflex
207	317	cf	Ï	Idieresis	Idieresis	Idieresis	caron	Idieresis	oe	Idieresis
208	320	d0	Ð	Eth	Eth	Eth	emdash	Eth	endash	Eth
209	321	d1	Ñ	Ntilde	Ntilde	Ntilde		Ntilde	emdash	Ntilde
210	322	d2	Ò	Ograve	Ograve	Ograve		Ograve	quotedblleft	Ograve
211	323	d3	Ó	Oacute	Oacute	Oacute		Oacute	quotedblright	Oacute
212	324	d4	Ô	Ocircumflex	Ocircumflex	Ocircumflex		Ocircumflex	quoteleft	Ocircumflex
213	325	d5	Õ	Otilde	Otilde	Otilde		Otilde	quoteright	Otilde
214	326	d6	Ö	Odieresis	Odieresis	Odieresis		Odieresis	divide	Odieresis
215	327	d7	Œ	OE	multiply	multiply		multiply	lozenge	multiply
216	330	d8	Ø	Oslash	Oslash	Oslash		Oslash	ydieresis	Oslash
217	331	d9	Ù	Ugrave	Ugrave	Ugrave		Ugrave	Ydieresis	Ugrave
218	332	da	Ú	Uacute	Uacute	Uacute		Uacute	fraction	Uacute
219	333	db	Û	Ucircumflex	Ucircumflex	Ucircumflex		Ucircumflex	currency	Ucircumflex
220	334	dc	Ü	Udieresis	Udieresis	Udieresis		Udieresis	guilsinglleft	Udieresis
221	335	dd	Ý	Yacute	Yacute	Yacute		Yacute	guilsinglright	Yacute
222	336	de	Þ	Thorn	Thorn	Thorn		Thorn	fi	Thorn
223	337	df	SS	Germandbls	germandbls	germandbls		germandbls	fl	germandbls
224	340	e0	à	agrave	agrave	agrave		agrave	daggerdbl	agrave
225	341	e1	á	aacute	aacute	aacute	AE	aacute	periodcentered	aacute
226	342	e2	â	acircumflex	acircumflex	acircumflex		acircumflex	quotesinglbase	acircumflex

Таблица А.4. *Продолжение.*

Десятичный	Восьмеричный	Шестнадцатеричный	Глиф в EC	EC	TeXBase1	ISO Latin 1	Standard	Windows ANSI	Mac Roman	PDF
227	343	e3	ã	atilde	atilde	atilde	ordfeminine	atilde	quotedblbase	atilde
228	344	e4	ä	adieresis	adieresis	adieresis		adieresis	perthousand	adieresis
229	345	e5	å	aring	aring	aring		aring	Acircumflex	aring
230	346	e6	æ	ae	ae	ae		ae	Ecircumflex	ae
231	347	e7	ç	ccedilla	ccedilla	ccedilla		ccedilla	Aacute	ccedilla
232	350	e8	è	egrave	egrave	egrave	Lslash	egrave	Edieresis	egrave
233	351	e9	é	eacute	eacute	eacute	Oslash	eacute	Egrave	eacute
234	352	ea	ê	ecircumflex	ecircumflex	ecircumflex	OE	ecircumflex	Iacute	ecircumflex
235	353	eb	ë	edieresis	edieresis	edieresis	ordmasculine	edieresis	Icircumflex	edieresis
236	354	ec	ì	igrave	igrave	igrave		igrave	Idieresis	igrave
237	355	ed	í	iacute	iacute	iacute		iacute	Igrave	iacute
238	356	ee	î	icircumflex	icircumflex	icircumflex		icircumflex	Oacute	icircumflex
239	357	ef	ï	idieresis	idieresis	idieresis		idieresis	Ocircumflex	idieresis
240	360	f0	ð	eth	eth	eth		eth	apple	eth
241	361	f1	ñ	ntilde	ntilde	ntilde	ae	ntilde	Ograve	ntilde
242	362	f2	ò	ograve	ograve	ograve		ograve	Uacute	ograve
243	363	f3	ó	oacute	oacute	oacute		oacute	Ucircumflex	oacute
244	364	f4	ô	ocircumflex	ocircumflex	ocircumflex		ocircumflex	Ugrave	ocircumflex
245	365	f5	õ	otilde	otilde	otilde	dotlessi	otilde	dotlessi	otilde
246	366	f6	ö	odieresis	odieresis	odieresis		odieresis	circumflex	odieresis
247	367	f7	œ	oe	divide	divide		divide	tilde	divide
248	370	f8	ø	oslash	oslash	oslash	lslash	oslash	macron	oslash
249	371	f9	ù	ugrave	ugrave	ugrave	oslash	ugrave	breve	ugrave
250	372	fa	ú	uacute	uacute	uacute	oe	uacute	dotaccent	uacute
251	373	fb	û	ucircumflex	ucircumflex	ucircumflex	germandbls	ucircumflex	ring	ucircumflex
252	374	fc	ü	udieresis	udieresis	udieresis		udieresis	cedilla	udieresis
253	375	fd	ý	yacute	yacute	yacute		yacute	hungarumlaut	yacute
254	376	fe	þ	thorn	thorn	thorn		thorn	ogonek	thorn
255	377	ff	ß	germandbls	ydieresis	ydieresis		ydieresis	caron	ydieresis

A.7 Справочная информация по fontinst

A.7.1 Файлы кодировки

Файл кодировки является TEX'овским документом следующей структуры, в котором *encoding commands* описывают кодировку шрифта:

```
\relax
  комментарии (игнорируются)
\encoding
  encoding commands
\endencoding
комментарии (игнорируются)
```

Поскольку в файле кодировки любой материал, помещенный между `\relax` и `\encoding`, игнорируется, то этот файл может также иметь структуру LATEX'овского файла и обрабатываться как обычный файл. Для документирования инструкций кодировки в этом случае используется пакет fontdoc:

```
\documentclass{article}
\usepackage{fontdoc}

\begin{document}
В этом файле описана кодировка.
\encoding
   ...
\endencoding
А вот и конец
\end{document}
```

В разделе *encoding commands* для глифов в шрифте вводятся «слоты», которым с помощью следующих команд приписываются определенные позиции в кодировке:

`\nextslot{`*number*`}`
: Установить номер следующего слота явным образом. Если команда `\nextslot` не указана, то используется номер, следующий за номером предыдущего слота.

`\setslot{`*glyph*`}` *slot commands* `\endsetslot`
: Установить слот для глифа *glyph*. Блок *slot commands* описывает глиф и его использование в TEX'е.

`\inputetx{`*file*`}`
: Вводит *encoding commands* из файла *file*`.etx`.

В файлах кодировки могут устанавливаться переменная `\codingscheme` (схема кодировки шрифта), а также следующие переменные целого типа (с помощью команды `\setint`): `\boundarychar` (номер слота для символа, используемого как граница слова), `\letterspacing` (размер дополнительного пространства, вставляемого между каждыми двумя глифами) и, кроме того,

`\fontdimen(`*n*`)`
: Устанавливает значение n-го параметра размера шрифта.

Для каждого слота файл кодировки генерирует целую переменную, на которую можно ссылаться отовсюду; команда `\int{glyph}` позволяет вычислить номер слота для глифа *glyph*.

Возможными командами в блоке *slot commands* могут быть[1]:

`\usedas{`*type*`}{`*control sequence*`}`
Устанавливает управляющую последовательность, используемую в TEX'е для символа, тип *type* которого выбирается из списка: *accent*, *char*, *mathaccent*, *mathbin*, *mathclose*, *mathdelim*, *mathopen*, *mathord*, *mathpunct*, *mathrel*, *mathvariable*.

`\nextlarger{`*glyph*`}`
Добавляет запись *nextlarger*, указывающую на глиф *glyph* следующего размера. Используется при создании математических шрифтов.

`\varchar` *varchar commands* `\endvarchar`
Добавляет запись *varchar*, указывающую на формирование композитного математического глифа с помощью следующих команд:

`\vartop{`*glyph*`}`
Указывает глиф *glyph*, используемый для рисования верхней части композитного глифа *varchar*.

`\varmid{`*glyph*`}`
Указывает глиф *glyph*, используемый для рисования средней части композитного глифа *varchar*.

`\varbot{`*glyph*`}`
Указывает глиф *glyph*, используемый для рисования нижней части композитного глифа *varchar*.

`\varrep{`*glyph*`}`
Устанавливает глиф *glyph*, используемый для повтороного наращивания размеров средней части композитного глифа *varchar*.

A.7.2 Метрические файлы

Метрический файл является TEX'овским документом, описывающим глифы в шрифте. Он имеет следующую структуру:

```
\relax
комментарии (игнорируются)
\metrics
metric commands
\endmetrics
комментарии (игнорируются)
```

Командами *metric commands* являются следующие:

`\setglyph{`*name*`}` *glyph commands* `\endsetglyph`
Определить глиф с именем *name* с помощью команд *glyph commands*, если он ранее не был определен.

[1]Здесь перечислены не все возможные команды в этом блоке. В частности, не упомянута широко используемая команда `comment`, служащая для добавления комментариев, выводимых с помощью пакета fontdoc. — *Прим. перев.*

\resetglyph{*name*} *glyph commands* \endsetglyph
Определить глиф с именем *name* с помощью *glyph commands*, даже если он уже был определен.

\unsetglyph{*name*}
Считать глиф с именем *name* неопределенным.

\setrawglyph{*name*}{*font*}{*size*}{*slot*}{*width*}{*height*}{*depth*}{*italic correction*}
Установить глиф с именем *name* из шрифта *font*, который имеет размер (size), номер слота (slot), ширину (width), высоту (height), глубину (depth) и поправку на курсив (italic correction). Эта команда обычно генерируется автоматически из файлов `afm` или `pl`.

\setkern{*glyph*}{*glyph*}{*amount*}
Установить керн между двумя указанными глифами.

\setleftkerning{*glyph*}{*glyph*}{*amount*}
Установить пропорцию (по отношению к 1000), на которую первый глиф должен скопировать левый керн второго глифа.[1] В следующем примере указывается, что глиф Asmall (капительная буква A) должен использовать в качестве своего левого кернинга кернинг обычной буквы A в пропорции 85%, а глиф IJ должен иметь точно такой же левый кернинг, как обычная буква I.

```
\setleftkerning{Asmall}{A}{850}
\setleftkerning{IJ}{I}{1000}
```

\setrightkerning{*glyph*}{*glyph*}{*amount*}
Установить долю, на которую первый глиф должен копировать правый керн второго глифа.

\inputmtx{*file*}
Ввести/считать команды *metric commands* из файла *file*`.mtx`.

Метрические файлы позволяют также установить такие размеры, как `\ascender` (высоту наиболее высокой строчной буквы), `\capheight` (высоту наиболее высокой прописной буквы), `\descender` (глубину прописных букв), `\italicslant` (величину наклона букв, задаваемую в единицах смещения направо на каждые 1000 единиц смещения вверх), `\minimumkern` (величину размера керна, ниже которой он игнорируется), `\underlinethickness` (ширину линии подчеркивания), `\xheight` (высоту строчных букв) и `\designsize` (дизайнерские размеры шрифта), а также параметр `\monowidth` (устанавливается для моноширинного шрифта).

Указанными выше командами *glyph commands* могут быть:

\glyph{*glyph*}{*scale*}
Устанавливает указанный глиф с заданным масштабом. При этом

- Смещает указатель текущей позиции по горизонтали на ширину текущего глифа.
- Высоту текущего глифа устанавливает равной по крайней мере высоте указанного глифа, подгоняя под текущее смещение по вертикали.
- Глубину текущего глифа устанавливает равной по крайней мере глубине указанного глифа, подгоняя под текущее смещение по вертикали.
- Устанавливает поправку на курсив текущего глифа такой же, как у указанного глифа.

[1] В действительности речь идет о размерах апрошей. — *Прим. перев.*

Указываемый глиф должен быть уже определен, иначе возникает ошибка. Например:

```
\setglyph{fi}
 \glyph{f}{1000}
 \glyph{i}{1000}
\endsetglyph
\setglyph{Asmall}
 \glyph{A}{700}
\endsetglyph
```

\glyphrule{*integer*}{*integer*}
Задает линейку заданной ширины и высоты. Например:

```
\setglyph{underline}
 \glyphrule{333}{40}
\endsetglyph
```

\glyphspecial{*text*}
Устанавливает зависящую от устройства \special. Например:

```
\setglyph{crest}
  \glyphspecial{Filename: crest.eps}
\endsetglyph
```

\glyphwarning{*text*}
Устанавливает предупреждение о \special и вызывает вывод предупреждения каждый раз, когда глиф используется. Например:

```
\setglyph{missingglyph}
 \glyphrule{500}{500}
 \glyphwarning{missing glyph}
\endsetglyph
```

\movert{*integer expression*}
Сдвигает указатель текущей позиции направо и смещает глиф на указанную величину. Например:

```
\setglyph{Asmall}
 \movert{50}
 \glyph{A}{700}
 \movert{50}
\setendglyph
```

\moveup{*integer expression*}
Сдвигает указатель текущей позиции вверх на указанную величину. Каждый глиф должен заканчиваться с вертикальным смещением, равным нулю. Например:

```
\setglyph{onehalf}
 \moveup{500}
 \glyph{one}{700}
 \moveup{-500}
 \glyph{slash}{1000}
 \moveup{-200}
 \glyph{two}{700}
 \moveup{200}
\endsetglyph
```

\push *glyph commands* \pop
Исполняет *glyph commands* без подгонки текущего положения или ширины глифа. Например:

```
\setglyph{aacute}
 \push
 \movert{\div{\sub{\width{a}}{\width{acute}}}{2}}
 \glyph{acute}{1000}
 \pop
 \glyph{a}{1000}
\endsetglyph
```

\resetwidth{*integer expression*}
Устанавливает ширину текущего глифа, равную *integer expression*.

\resetheight{*integer expression*}
Устанавливает высоту текущего глифа, равную *integer expression*.

\resetdepth{*integer expression*}
Устанавливает глубину текущего глифа, равную *integer expression*.

\resetitalic{*integer expression*}
Устанавливает поправку на курсив текущего глифа, равную *integer expression*.

Внутри определения *glyph* вы можете использовать выражения, такие, как \width{*glyph*}, которое ссылается на ранее определенный глиф. Например, знак суммы в режиме displaystyle можно определить как текстовый знак суммы $\sum$, который увеличен до 120% его первоначального размера и имеет высоту и глубину, увеличенные на половину пункта:

```
\setglyph{sumdisplay}
 \glyph{textsum}{1200}
 \resetheight{\add{\height{sumdisplay}}{50}}
 \resetdepth{\add{\depth{sumdisplay}}{50}}
\endsetglyph
```

Внутри \resetglyph эти выражения относятся к предыдущему определению глифа. Например, вы можете добавить апроши к букве «A» с помощью

```
\resetglyph{A}
 \movert{25}
 \glyph{A}{1000}
 \movert{25}
\endresetglyph
```

A.7.3 Низкоуровневые команды fontinst

Для создания управляющих файлов fontinst существует множество команд:

\ifiscommand{*command*}\then
Выполнить \iftrue, если команда *command* определена, и \iffalse в противном случае.

\ifisdim{*dim*}\then
Выполнить \iftrue, если переменная размера *dim* определена, и \iffalse в противном случае.

`\ifisglyph{`*glyph*`}\then`
Выполнить `\iftrue`, если глиф *glyph* определен, и `\iffalse` в противном случае.

`\ifisint{`*int*`}\then`
Выполнить `\iftrue`, если строковая целая переменная *int* определена, и `\iffalse` в противном случае.

`\ifisstr{`*str*`}\then`
Выполнить `\iftrue`, если строковая переменная *str* определена, и `\iffalse` в противном случае.

`\needsfontinstversion{`*version*`}`
Выдать предупреждение, если номер текущей версии пакета `\fontinst` меньше, чем *version*.

`\resetcommand{`*command*`}{`*definition*`}`
Синоним `\def`.

`\resetdim{`*dim*`}{`*dimension*`}`
Присвоить переменной размера *dim* текущее значение *dimension*.

`\resetint{`*int*`}{`*integer expression*`}`
Присвоить целой переменной *int* текущее значение *integer expression*.

`\resetstr{`*str*`}{`*string*`}`
Присвоить строковой переменной *str* текущее значение *string*.

`\setcommand{`*command*`}{`*definition*`}`
Если команда *command* не определена, она устанавливается равной *definition*. Команда имеет такой же синтаксис, как TEX'овская команда `\def`.

`\setdim{`*dim*`}{`*dimension*`}`
Если переменная размера *dim* не определена, она устанавливается равной текущему значению *dimension*.

`\setint{`*int*`}{`*integer expression*`}`
Если целая переменная *int* не определена, она устанавливается равной текущему значению *integer expression*.

`\setstr{`*str*`}{`*string*`}`
Если строковая переменная *str* остается неопределенной, она устанавливается равной текущему значению *string*.

`\unsetcommand{`*command*`}`
Делает команду *command* неопределенной.

`\unsetdim{`*dim*`}`
Делает размер *dim* неопределенным.

`\unsetint{`*int*`}`
Делает *int* неопределенным целым.

`\unsetstr{`*str*`}`
Делает строковую переменную *str* неопределенной.

`\int{`*int*`}`
Значение целой переменной *ini*.

`\width{`*glyph*`}`
Ширина глифа *glyph*.

`\height{`*glyph*`}`
Высота глифа *glyph*.

`\depth{`*glyph*`}`
Глубина глифа *glyph*.

`\italic{`*glyph*`}`
Поправка на курсив для глифа *glyph*.

`\kerning{`*left*`}{`*right*`}`
Величина кернинга между глифами *left* и *right*.

`\neg{`*integer expression*`}`
Отрицательное значение *integer expression*.

`\add{`*integer expression*`}{`*integer expression*`}`
Сумма двух указанных *integer expression*.

`\sub{`*integer expression*`}{`*integer expression*`}`
Величина, равная разности первого и второго *integer expression*.

`\mul{`*integer expression*`}{`*integer expression*`}`
Произведение двух указанных *integer expression*.

`\div{`*integer expression*`}{`*integer expression*`}`
Величина, равная первому *integer expression*, деленному на второе.

`\scale{`*integer expression*`}{`*integer expression*`}`
Величина произведения двух указанных *integer expression*, деленного на 1000.

`\declaresize{`*size*`}{`*fd-size-range*`}`
Декларирует новый размер и дает `fd` для него. Например, `fontinst.sty` декларирует следующие размеры:

```
\declaresize{}{<->}
\declaresize{5}{<5>}
\declaresize{6}{<6>}
\declaresize{7}{<7>}
\declaresize{8}{<8>}
\declaresize{9}{<9>}
\declaresize{10}{<10>}
\declaresize{11}{<10.95>}
\declaresize{12}{<12>}
\declaresize{14}{<14.4>}
\declaresize{17}{<17.28>}
\declaresize{20}{<20.74>}
\declaresize{25}{<24.88>}
```

`\declareencoding{`*string*`}{`*etx*`}`
Декларирует, какой файл `etx` соответствует какому названию кодировки. Например, `fontinst.sty` декларирует, такие названия кодировок:

```
\declareencoding{EXTENDED TEX FONT ENCODING - LATIN}{T1}
\declareencoding{TEX TEXT}{OT1}
\declareencoding{TEX TYPEWRITER TEXT}{OT1TT}
\declareencoding{TEX MATH ITALIC}{OML}
\declareencoding{TEX MATH SYMBOLS}{OMS}
\declareencoding{TEX MATH EXTENSION}{OMX}
```

A.8 Драйверы Ghostscript'a

Дисплеи DOS и Windows

`ega`	EGA (640×350, 16 цветов)
`vga`	VGA (640×480, 16 цветов)
`s3vga`	SuperVGA с микросхемой S3 86C911 (например, плата Diamond Stealth)

`svga16`	Общий SuperVGA в 16-цветном режиме 800×600
`tseng`	SuperVGA с микросхемами Tseng Labs ET3000/4000, 256 цветов
`tvga`	Trident SuperVGA, 256 цветов
`vesa`	SuperVGA с API-драйвером стандарта VESA
Дисплеи OS/2	
`os2pm`	OS/2 Presentation Manager
`os2dll`	OS/2 DLL bitmap
`os2prn`	Принтер OS/2
Дисплеи Unix и VMS	
`lvga256`	Linux vgalib, 256-цветные VGA-режимы
`vgalib`	Linux PC с vgalib
`x11`	X Windows System версии 11, номер выпуска >=4
`x11alpha`	X Windows с градациями серого
Принтеры	
`ap3250`	Принтер Epson AP3250
`bj10e`	Canon BubbleJet BJ10e
`bj200`	Canon BubbleJet BJ200
`bjc600`	Canon Color BubbleJet BJC-600, BJC-4000 и BJC-70
`bjc800`	Canon Color BubbleJet BJC-800
`ccr`	CalComp Raster format
`cdeskjet`	HP DeskJet 500C, 1 бит/пиксел (черно-белый)
`cdjcolor`	HP DeskJet 500C, 24 бит/пиксел с высококачественной диффузией цвета (Floyd-Steinberg); подходит для DeskJet 540C
`cdjmono`	HP DeskJet 500C (только черно-белый); подходит для DeskJet 510, 520 и 540C (только черно-белый)
`cdj500`	HP DeskJet 500C (то же, что и cdjcolor)
`cdj550`	HP DeskJet 550C/560C
`deskjet`	HP DeskJet и DeskJet Plus
`djet500`	HP DeskJet 500
`djet500c`	Другой драйвер для HP DeskJet 500C (не работает с 550C и 560C)
`dnj650c`	HP DesignJet 650C
`epson`	Epson-совместимые точечные матричные принтеры (9 и 24 иголки)
`eps9mid`	Epson-совместимый принтер с 9 иголками, чересстрочная печать (промежуточное разрешение)
`eps9high`	Epson-совместимый принтер с 9 иголками, чересстрочная печать (утроенное разрешение)
`epsonc`	Цветные принтеры Epson LQ-2550 и Fujitsu 3400/2400/1200
`ibmpro`	IBM Proprinter с 9 иголками
`imagen`	Принтеры Imagen ImPress
`iwhi`	Apple Imagewriter в режиме с высоким разрешением
`iwlo`	Apple Imagewriter в режиме с низким разрешением
`iwlq`	Apple Imagewriter LQ в режиме 320×216 dpi
`jetp3852`	Струйный цветной принтер IBM Jetprinter (модель 3852)
`laserjet`	HP LaserJet
`la50`	Принтер DEC LA50
`la70`	Принтер DEC LA70
`la70t`	Принтер DEC LA70 с улучшенной печатью текста в режиме с низким разрешением
`la75`	Принтер DEC LA75

`la75plus`	Принтер DEC LA75plus
`lbp8`	Лазерный принтер Canon LBP-8II
`lips3`	Лазерный принтер Canon LIPS III в английском (CaPSL) режиме
`ln03`	Принтер DEC LN03
`lj250`	Цветной принтер DEC LJ250 Companion
`ljet2p`	HP LaserJet IId/IIp/III с компрессией TIFF
`ljet3`	HP LaserJet III с компрессией Delta Row
`ljet3d`	HP LaserJet IIID с возможностью двусторонней печати
`ljet4`	HP LaserJet 4 (по умолчанию 600 точек на дюйм)
`lj4dith`	HP LaserJet 4 с диффузией по алгоритму Floyd-Steinberg
`ljetplus`	HP LaserJet Plus
`lp2563`	Принтер HP 2563B
`m8510`	Принтер C.Itoh M8510
`necp6`	Принтеры NEC P6/P6/P60 с разрешением 360×360 точек на дюйм
`nwp533`	Лазерный принтер Sony Microsystems NWP533
`oce9050`	Принтер OCE 9050
`oki182`	Okidata MicroLine 182
`okiibm`	IBM-совместимые принтеры Okidata MicroLine
`paintjet`	Другой драйвер для цветного принтера HP PaintJet
`pj`	Драйвер HP PaintJet
`pjetxl`	Другой драйвер HP PaintJet XL
`pjxl`	Цветной принтер для HP PaintJet XL
`pjxl300`	Цветной принтер HP PaintJet XL300; подходит для PaintJet 1200C
`r4081`	Лазерный принтер Ricoh 4081
`sj48`	Струйный принтер StarJet 48
`sparc`	SPARCprinter
`st800`	Принтер Epson Stylus 800
`stcolor`	Epson Stylus Color
`t4693d2`	Цветной принтер Tektronix 4693d, 2 бита на компоненту RGB
`t4693d4`	Цветной принтер Tektronix 4693d, 4 бита на компоненту RGB
`t4693d8`	Цветной принтер Tektronix 4693d, 8 бит на компоненту RGB
`tek4696`	Струйный плоттер Tektronix 4695/4696
`xes`	Принтеры Xerox XES (2700, 3700, 4045 и др.)

Форматы факсимильной печати (Fax systems) [1]

`dfaxhigh`	Формат DigiBoard, Inc.'s DigiFAX software (высокое разрешение)
`dfaxlow`	Низкое (обычное) разрешение DigiFAX
`faxg3`	Group 3 fax, с концами строк, но без заголовка и признака конца данных
`faxg32d`	Group 3 2D fax, с концами строк, но без заголовка и признака конца данных
`faxg4`	Group 4 fax, с концами строк, но без заголовка и признака конца данных
`tiffcrle`	TIFF "CCITT RLE 1-dim" (Group 3 fax без концов строк)
`tiffg3`	TIFF Group 3 fax (с концами строк)
`tiffg32d`	TIFF Group 3 2D fax
`tiffg4`	TIFF Group 4 fax

[1] Все эти драйверы подгоняют размер бумаги так, чтобы соответствовать одному из трех стандартных форматов CCITT (U.S. letter с шириной как у A4, A4 или B4).

Форматы растровых файлов

`bmpmono`	Монохромный BMP-формат (MS Windows)
`bmp16`	4-битовый (EGA/VGA) BMP
`bmp256`	8-битовый (256-цветный) BMP
`bmp16m`	24-битовый BMP
`cgmmono`	Монохромный (черно-белый) CGM
`cgm8`	8-битовый (256-цветный) CGM
`cgm24`	24-битовый цветной CGM
`cif`	Формат CIF для VLSI
`miff24`	Формат ImageMagick MIFF, 24-битовый (прямой) цвет, компрессия RLE
`mgrmono`	1-битовые монохромные MGR-устройства
`mgrgray2`	2-битовые серые (gray scale) MGR-устройства
`mgrgray4`	4-битовые серые MGR-устройства
`mgrgray8`	8-битовые серые MGR-устройства
`mgr4`	4-битовые (VGA) цветные MGR-устройства
`mgr8`	8-битовые цветные MGR-устройства
`pcxmono`	Монохромный формат PCX (1-битовый, черно-белый)
`pcxgray`	PCX, 8-битовый серый
`pcx16`	PCX, 4-битовый «плоский» цвет (EGA/VGA)
`pcx256`	PCX, 8-битовый цвет
`pcx24b`	PCX, 24-битовый цвет (три 8-битовые плоскости)
`pbm`	Portable Bitmap (неупакованный формат)
`pbmraw`	Portable Bitmap (упакованный формат)
`pgm`	Portable Graymap (неупакованный формат)
`pgmraw`	Portable Graymap (упакованный формат)
`pgnm`	Portable Graymap (неупакованный формат), при возможности преобразуемый в PBM
`pgnmraw`	Portable Graymap (упакованный формат), при возможности преобразуемый в PBM
`pnm`	Portable Pixmap (неупакованный формат) (RGB), при возможности преобразуемый в PGM или PBM
`pnmraw`	Portable Pixmap (упакованный формат) (RGB), при возможности преобразуемый в PGM или PBM
`ppm`	Portable Pixmap (неупакованный формат) (RGB)
`ppmraw`	Portable Pixmap (упакованный формат) (RGB)
`pngmono`	Монохромный формат Portable Network Graphics (PNG)
`pnggray`	8-битовый серый PNG
`png16`	4-битовый цветной PNG
`png256`	8-битовый цветной PNG
`png16m`	24-битовый цветной PNG
`psmono`	Монохромный рисунок в PostScript'е первого уровня
`sgirgb`	Формат SGI RGB pixmap
`tiff12nc`	TIFF 12-битовый RGB без компрессии
`tiff24nc`	TIFF 24-битовый RGB без компрессии (стандартный формат NeXT)
`tifflzw`	TIFF со сжатием LZW (монохромный)
`tiffpack`	TIFF PackBits (монохромный)

Форматы высокого уровня

`pdfwrite`	Portable Document Format (PDF)

Как все это получить

Мы надеемся, что, читая эту книгу, вы действительно попытаетесь найти один или несколько из описанных пакетов, опробуете их сами, оцените, насколько они полезны для вашей работы и, возможно, получите удовольствие.

Большинство пакетов и программ, описанных здесь, могут быть свободно получены из Интернета с помощью ftp (file transfer protocol) или через интерфейс WWW с одного из архивов CTAN (Comprehensive TEX Archive Network (Greenwade, 1993)). В настоящее время имеется два основных узла CTAN, один в Кембридже (Великобритания), а другой в Гейдельберге (Германия). Кроме того, имеется много «зеркальных» узлов в различных частях света. Для тех, кто имеет плохую связь с Интернетом или вовсе ее не имеет, выпущено несколько дисков CD-ROM, содержащих «слепки» архивов CTAN. В частности, CD-ROM под именем *TEX Live* доступен у различных групп пользователей TEX'а (см. Burbank and Goossens (1996) и `http://www.tug.org/usergroups.html` на предмет текущего состояния). Международная группа TEX Users Group (TUG), английская UK-TUG и французская GUTenberg в сотрудничестве с членами других групп и добровольными помощниками выпустили этот plug-and-play CD-ROM с дистрибутивом teTeX (TEX для Unix), который составлен Томасом Эссером и основывается на программе Web2c, созданной Карлом Берри. Его дерево каталогов соответствует предложенному стандарту TDS (TEX Directory Structure) (TUG Working Group on the TEX Directory Structure (TWG-TDS), 1995). Этот дистрибутив содержит сконфигурированную рабочую систему для более чем двадцати платформ, работающих под Unix. Благодаря тому что он записан в стандарте ISO 9660, CD-ROM может также использоваться в системах MSDOS и MS-Windows, хотя установка не оптимизирована для этих операционных систем. Детали, касающиеся этого CD-ROM'а, можно найти по адресу `http://www.tug.org/texlive.html`.[1]

[1] В настоящее время вариант teTeX под именем fpTeX перенесен и в среду MS-Windows. CD-ROM TEX Live 6, модифицированный под нужды отечественного TEX-пользователя, распространяется в комплекте с книгой М. Гуссенс, С. Ратц. Путеводитель по пакету LATEX и его Web-приложениям — М.: Мир, 2001. — *Прим. перев.*

В первой части этого приложения мы опишем различные способы доступа к архивам CTAN с помощью Интернета, а во второй части дадим перечень описанных в этой книге пакетов и прикладных программ, которые могут быть найдены в Интернете.

B.1 Подключение к архиву CTAN

Для тех читателей, кто имеет достаточно быстрое соединение с Интернетом, получение самой последней версии пакета прямо из архива CTAN или одного из его «зеркал», вероятно, является наилучшей альтернативой. Тем не менее мы считаем, что все равно стоит купить один из последних дисков с TEX'ом и родственными ему пакетами, о которых мы говорили в предыдущем разделе, поскольку такой CD-ROM представляет собой удобный и полный справочник. В частности, все пакеты, описанные в этой книге, имеются на CD-ROM'е *TEX Live*.

Если вы решили перекачать какой-то пакет, вам следует подключиться к Интернету. Заметим, что, хотя сетевые соединения все время становятся быстрее, будет разумнее подключиться к узлу, географическое расположение которого не очень далеко от вашего. В наше время наиболее удобным способом подключения к Интернету является использование навигатора, в особенности поскольку многие узлы с TEX'овскими архивами CTAN разработали дружественный интерфейс к этим архивам.

B.1.1 Поиск файлов в архиве

Одним из примеров Web-интерфейса является поисковая система CTAN `http://www.ucc.ie/cgi-bin/ctan`, созданная Питером Флинном (см. рис. B.1). На введенный запрос («pstricks» в нашем примере) поисковая система выводит список всех файлов в архиве CTAN, которые удовлетворяют заданному критерию поиска. DANTE — группа пользователей TEX'а в Германии — предлагает похожую поисковую систему по адресу `http://www.dante.de/cgi-bin/ctan-index`. Подобный интерфейс к структуре директорий имеется и на `http://www.tex.ac.uk/tex-archive/`; последний интерфейс автоматически показывает текст файла «README», если таковой присутствует в соответствующей директории.

Каталог относящихся к TEX'у и LATEX'у пакетов, который поддерживается Грэмом Уильямсом, может быть найден по адресу `http://www.dit.csiro.au/~gjw/texpkgs.html` (см. рис. B.2). Этот интерфейс особенно привлекателен, если вы хотите, «пролистывая» архив, узнать, что делает конкретный пакет. Кроме этого, интерфейс позволяет вам выбрать узел, с которого лучше копировать файлы, так что вы можете оптимизировать ваше соединение. Когда расширенное описание пакета доступно в Интернете, его URL показывается вместе с соответствующей записью. В Интернете есть несколько копий этого каталога, который регулярно обновляется автором.

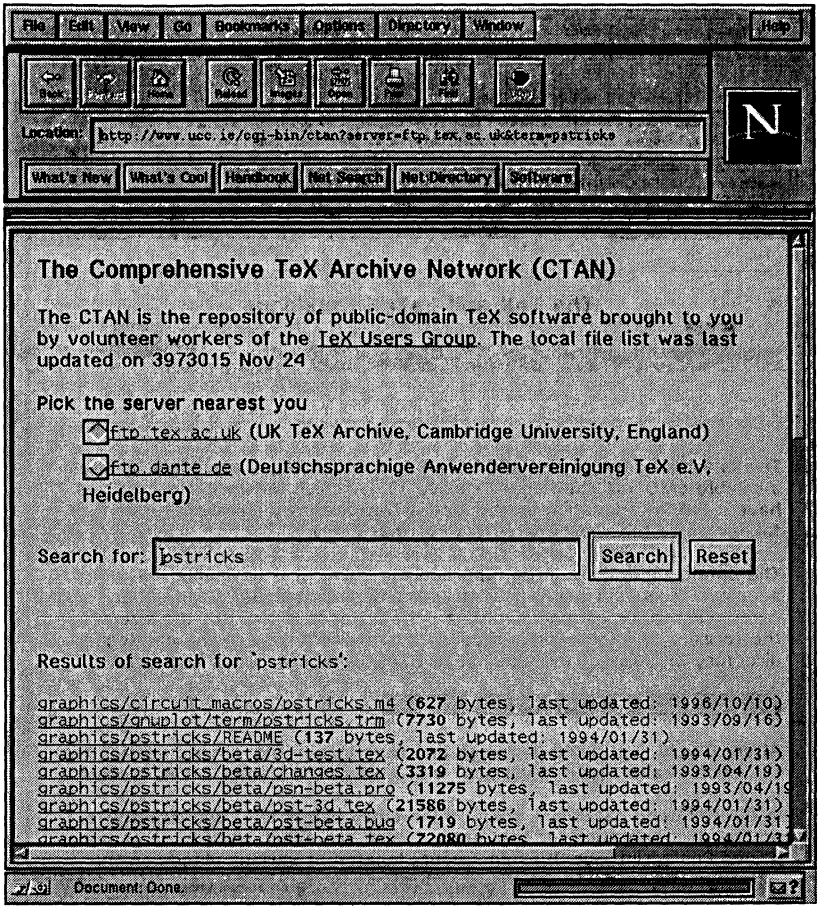


Рис. B.1. Web-интерфейс к CTAN Питера Флинна.

Эти интерфейсы оказываются очень уместными, если вы хотите проверить характеристики одного или нескольких конкретных файлов (дату последней модификации, размер, назначение и т. п.), но они не особенно удобны для копирования целых пакетов, содержащих большое число (иногда сотни) файлов; о копировании целых пакетов мы поговорим ниже.

B.2 Копирование пакета из архива

Хотя в предыдущих примерах мы пользовались навигатором Netscape, вы можете использовать любой навигатор как интерфейс к архивам CTAN. В этом разделе мы используем программу Internet Explorer фирмы Microsoft в операционной системе Microsoft Windows, чтобы показать различные этапы поиска и копирования целых пакетов.

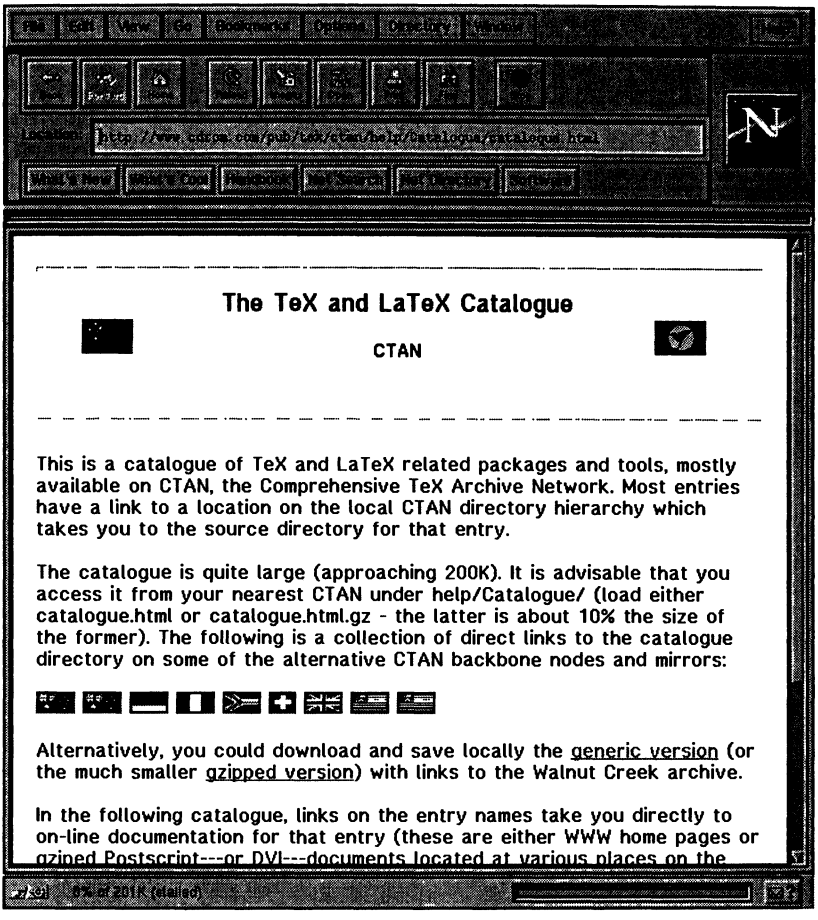


Рис. B.2. Каталог CTAN Грэма Уильямса в Интернете.

После подключения к поисковой системе WWW-узла DANTE CTAN в Интернете по адресу `http://www.dante.de/` мы запрашиваем имена файлов, содержащих строку «graphicx» (показанную внутри овала в верхней части рис. B.3). В средней части рисунка виден результат поиска. Затем мы выбираем одну строку с помощью указателя мыши (эта строка показана стрелкой), и навигатор показывает интересующий нас файл (в нижней части рисунка). После этого вы можете сохранить этот файл на диске.

Если вы хотите скопировать более чем один файл (например, целый пакет, состоящий из одной или большего числа директорий), вам следует подняться по дереву директорий на один уровень выше интересующей вас директории. В примере, показанном на рис. B.4, вам нужно ввести в окошко *Address*, содержащее текущее URL, имя директории с расширением `.zip` или `.tar.gz` (ftp-серверы узлов CTAN поддерживают автоматическое создание архивов типа

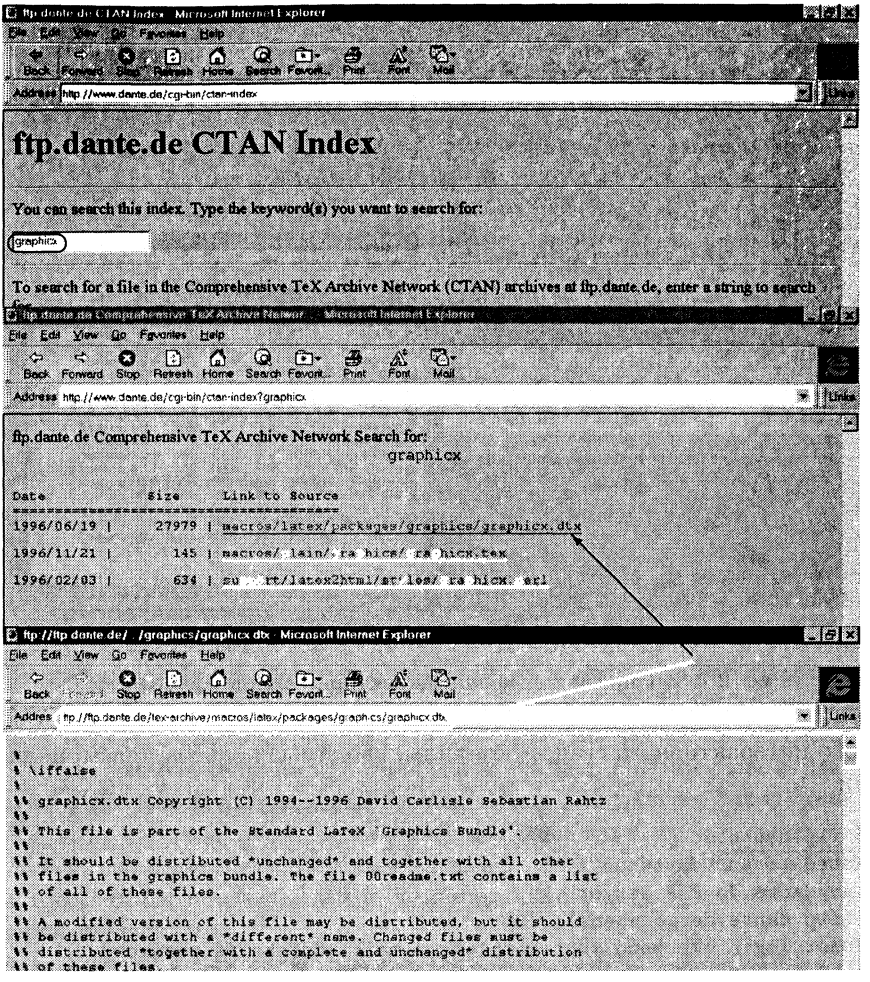


Рис. B.3. Просмотр файлов CTAN в Интернете.

`.zip`, `.zoo`, `.tar` и их сжатых с помощью программы gzip форм). При этом запрашиваемые в виде архива файлы будут скопированы. Результатом в нашем примере будет копирование в виде архива `zip` целой директории, содержащей все файлы, относящиеся к пакету graphics. В верхней части на рис. B.4 внутри окошка вы можете видеть команду, которая посылается на ftp-сервер. Снизу слева показан файл `graphics.zip` в вашей директории, а внизу справа — содержание архива `.zip` после завершения копирования в ваш персональный компьютер.

Вы также можете непосредственно соединиться с любым из архивов CTAN или его «зеркал», используя протокол ftp. Ниже приведена запись сеанса ти-

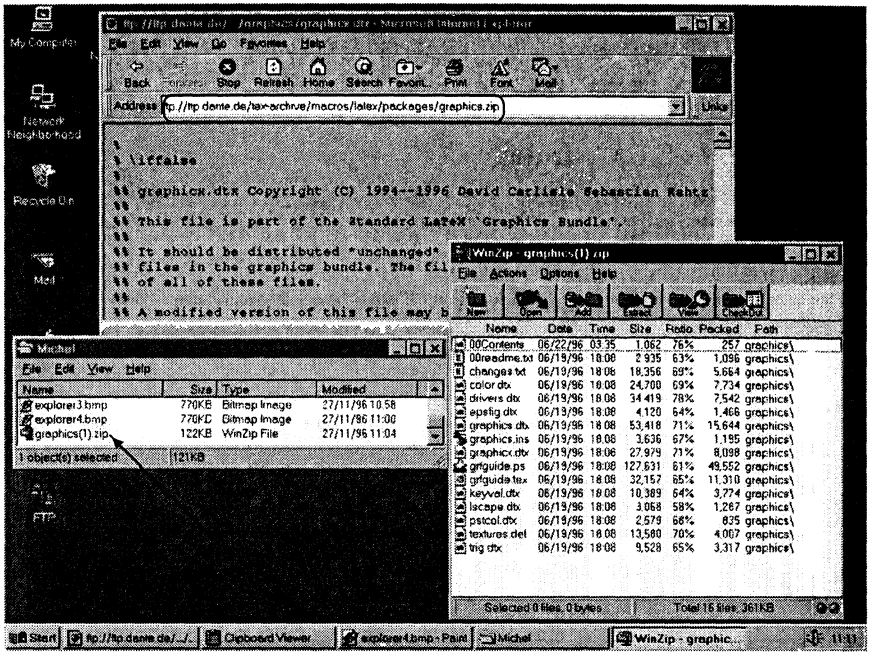


Рис. B.4. Копирование (целой) директории в виде архива `zip`.

пичного интерактивного сеанса (команды, вводимые пользователем, подчеркнуты):

```
> ftp ftp.dante.de
Connected to sun.dante.de.
220 sun.dante.de FTP server ...
Name (ftp.dante.de:goossens): ftp
331 Guest login ok, send your complete e-mail address as password.
Password: goossens@cern.ch (use your email address!)
230-Welcome, archive user!  This is an FTP server for the DANTE Archive.
   ...
ftp> quote site index graphicx
200-index graphicx
200-NOTE. This index shows at most 20 lines. for a full list of files,
200-retrieve /tex-archive/FILES.byname
200-1996/06/19 |      27979 | macros/latex/packages/graphics/graphicx.dtx
200-1996/11/21 |        145 | macros/plain/graphics/graphicx.tex
200-1996/02/03 |        634 | support/latex2html/styles/graphicx.perl
200  (end of 'index graphicx')
ftp> bin
200 Type set to I.
ftp> cd ctan:
250-Machine specific implementations        --> systems
250-Original Knuthian sources:              --> systems/knuth.
250-LaTeX styles, plain macros, MusicTeX:   --> macros.
```

```
250-LaTeX2e:                                --> macros/latex.
   ...
ftp> cd macros/latex/packages
250 CWD command successful.
ftp> get graphics.zip
200 PORT command successful.
150 Opening BINARY mode data connection for /bin/ZIP.
226 Transfer complete.
124742 bytes received in 4.88 seconds (24.94 Kbytes/s)
ftp> get graphics.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for /bin/tar.
226 Transfer complete.
117666 bytes received in 5.57 seconds (20.61 Kbytes/s)
ftp> quit
221 Goodbye.
```

После соединения с узлом (`ftp.dante.de` в примере выше) вы должны указать имя `ftp` как регистрационное имя (login name) и ввести *ваш* адрес email в качестве пароля. После этого вы можете послать запрос с помощью команды «`query site index` ⟨*term*⟩», где ⟨*term*⟩ — строка поиска. В нашем примере мы ввели тот же запрос «`graphicx`», что и на рис. B.3, и, разумеется, получили результат, идентичный по содержанию тому, который показан в средней части обсуждаемого рисунка. Поскольку мы решили скопировать весь пакет graphics, то сначала мы должны перейти в корень архива CTAN, подав команду «`cd ctan:`» (с двоеточием в конце!). Из ответа на наш запрос мы знаем, где в дереве каталогов расположены интересующие нас файлы. Поэтому мы переходим в директорию, лежащую на один уровень выше (`cd macros/latex/packages`). После этого мы копируем директорию дважды, один раз как архив `.zip`, а другой раз как компрессированный архив `.tar` (сделано это дважды только для того, чтобы показать, какие команды надо набирать). Команда «`quit`» завершает сеанс ftp.

B.3 Список пакетов и программ

В следующей таблице приводится либо полный адрес в Сети (Internet Uniform Resource Locator (URL)) или положение в архивах CTAN, либо указание, в каком из пакетов находится искомый ресурс. Ссылки на архив CTAN начинаются с префикса `CTAN:`

Таблица B.1. Адреса пакетов и программ, упомянутых в этой книге.

Имя	*Адрес*
2up	часть дистрибутива seminar
abc2mtex	`CTAN:support/abc2mtex`
Acrobat Distiller	`http://www.adobe.com/prodindex/acrobat/main.html`
Acrobat Reader	`http://www.adobe.com/prodindex/acrobat/readstep.html`
Adobe Acrobat	`http://www.adobe.com/prodindex/acrobat/main.html`
Adobe File Utilities	`http://www.adobe.com/prodindex/fileutilities/main.html`

Таблица В.1. *Продолжение.*

Имя	*Адрес*
Adobe Illustrator	`http://www.adobe.com/prodindex/illustrator/main.html`
Adobe Photoshop	`http://www.adobe.com/prodindex/photoshop/main.html`
Adobe Type Manager	`http://www.adobe.com/prodindex/atm/main.html`
afm2tfm	часть дистрибутива dvips
AlDraTex	`CTAN:macros/generic/dratex`
aliphat	часть пакета XeTeX
amstex	`CTAN:macros/amstex`
ArcInfo	`http://www.esri.com/base/products/arcinfo/arcinfo.html`
array	`CTAN:macros/latex/packages/tools`
aurora	`CTAN:support/aurora`
AutoCAD	`http://www.autodesk.com/products/autocad/autocad.htm`
avant	часть пакета PSNFSS
	`CTAN:fonts/psfonts/adobe/avantgar`
	`CTAN:fonts/psfonts/monotype/avantgar`
axodraw	`CTAN:graphics/axodraw`
bakoma	`CTAN:fonts/cm/type1/bakoma`
bar	`CTAN:macros/latex209/contrib/barkom`
basker	часть пакета psnfss-extra
	`CTAN:fonts/psfonts/adobe/basker`
bdfchess	`CTAN:fonts/chess/bdfchess`
bembo	часть пакета psnfss-extra
	`CTAN:fonts/psfonts/monotype/bembo`
	`CTAN:fonts/psfonts/xadobe/bembo`
	`CTAN:fonts/psfonts/xmonotype/bembo`
bg	`CTAN:macros/latex/contrib/other/bg`
bit2spr	`CTAN:graphics/bit2spr`
bitpxl	`CTAN:dviware/bitpxl`
bm2font	`CTAN:graphics/bm2font`
bookman	часть пакета psnfss-extra
	`CTAN:fonts/psfonts/adobe/bookman`
	`CTAN:fonts/psfonts/monotype/bookman`
boxes	часть дистрибутива METAPOST
bridge	`CTAN:macros/latex/contrib/other/bridge`
carom	часть пакета XeTeX
cchess	`CTAN:macros/latex/contrib/other/cchess`
ccycle	часть пакета XeTeX
chancery	часть пакета PSNFSS
charter	часть пакета PSNFSS
	`CTAN:fonts/charter`
	`CTAN:fonts/psfonts/bitstream/charter`
ChemDraw	`http://www.camsci.com/products/chemdraw/chemdraw.html`
chemstr	часть пакета XeTeX
chemsym	`CTAN:macros/latex/contrib/other/chemsym`
chem	часть Unix Documentors Workbench
cheq	`CTAN:fonts/cheq`
chesskey	`CTAN:fonts/chess/bdfchess`
chess	`CTAN:fonts/chess`
circ	`CTAN:macros/generic/diagrams/circ`
CMacTeX	`CTAN:systems/mac/cmactex`
colordvi	`CTAN:dviware/dvips`
colortab	`CTAN:macros/generic/misc`
colortbl	`CTAN:macros/latex/contrib/supported/carlisle`
color	`CTAN:macros/latex/packages/graphics`
Corel Draw	`http://www.corel.com/products/graphicsandpublishing/draw7/index.htm`
crosswrd	`CTAN:macros/latex/contrib/other/crosswrd`

Таблица B.1. *Продолжение.*

Имя	*Ад ес*
curves	CTAN:macros/latex/contrib/supported/curves
cwpuzzle	CTAN:macros/latex/contrib/other/gene/crossword
dcolumn	CTAN:macros/latex/packages/tools
draftcopy	CTAN:macros/latex/contrib/supported/draftcopy
DraTex	CTAN:macros/generic/dratex
dvi2ps	CTAN:dviware/dvi2ps
dvialw	CTAN:drivers/beebe
dvicopy	часть дистрибутива TEX
dvidrv	часть дистрибутива emTeX TEX
dvipsone	http://www.YandY.com/yydvips.htm
dvips	CTAN:dviware/dvips
dvitomp	часть дистрибутива METAPOST
dvitops	CTAN:dviware/dvitops
dviwindo	http://www.YandY.com/yydviwin.htm
dviwin	CTAN:dviware/dviwin
ecltree	CTAN:macros/latex209/contrib/eclbip
Edmetrics	http://www.bluesky.com/textures.html
eepic	CTAN:macros/latex/contrib/other/eepic
emTeX	CTAN:systems/msdos/emtex
epic	CTAN:macros/latex/contrib/other/epic/
epsffit	часть коллекции утилит psutils
epsfig	CTAN:macros/latex/packages/graphics
epstool	CTAN:support/ghostscript/rjl
Excel	http://www.microsoft.com/msexcel/
extractres	часть коллекции утилит psutils
fancybox	часть дистрибутива PSTricks
Feynarts	http://www.mathsource.com/cgi-bin/MathSource/Applications/Physics/0202-194
feynman	CTAN:macros/latex209/contrib/feynman
feynmf	CTAN:macros/latex/contrib/supported/feynmf
feynmp	часть пакета feynmf
fig2mf	CTAN:graphics/fig2mf
fixdlsrps	часть коллекции утилит psutils
fixfmps	часть коллекции утилит psutils
fixmacps	часть коллекции утилит psutils
fixpsditps	часть коллекции утилит psutils
fixpspps	часть коллекции утилит psutils
fixscribeps	часть коллекции утилит psutils
fixtpps	часть коллекции утилит psutils
fixwfwps	часть коллекции утилит psutils
fixwpps	часть коллекции утилит psutils
fixwwps	часть коллекции утилит psutils
flow	CTAN:support/flow
foiltex	CTAN:macros/latex/contrib/supported/foiltex
fontdoc	часть пакета fontinst
fontinst	CTAN:fonts/utilities/fontinst
FrameMaker	см. http://www.adobe.com/prodindex/framemaker/main.html
FreeHand	http://www.macromedia.com/software/freehand/index.html
garamond	часть пакета psnfss-extra CTAN:fonts/psfonts/adobe/garamond CTAN:fonts/psfonts/monotype/garamond
getafm	часть коллекции утилит psutils
gftodvi	часть дистрибутива TEX
gftopk	часть дистрибутива TEX
gftopxl	часть дистрибутива TEX
gftype	часть дистрибутива TEX

Таблица B.1. *Продолжение.*

Имя	*Адрес*
Ghostscript	CTAN:support/ghostscript/aladdin
ghostview	CTAN:support/ghostscript/gnu
gnuplot	CTAN:graphics/gnuplot
go	CTAN:fonts/go
gpic	часть дистрибутива GNU groff
graphics	CTAN:macros/latex/packages/graphics
graphicx	часть пакета graphics
graph	часть дистрибутива METAPOST
grap	часть Unix Documentors Workbench
gsftopk	CTAN:fonts/utilities/gsftopk
GSview	CTAN:support/ghostscript/rjl
hcycle	часть пакета XeTeX
helvet	часть пакета PSNFSS CTAN:fonts/psfonts/adobe/helvetic
hetaromh	часть пакета XeTeX
hetarom	часть пакета XeTeX
hhline	CTAN:macros/latex/packages/tools
Hijaak	http://www.csd.net/%7Earcturus/xhijks.htm
hp2xx	CTAN:support/hp2xx
hptomf	CTAN:support/hptomf
ifthen	CTAN:macros/latex/base
Image Alchemy	http://www.alchemy.com.au/welcome.html
includeres	часть коллекции утилит psutils
inputenc	CTAN:macros/latex/base
Internet Explorer	http://www.microsoft.com/ie/ie.htm
keyval	часть пакета graphics
latex2html	CTAN:support/latex2html
lcircuit	CTAN:graphics/lcircuit
letter	CTAN:macros/latex/base
locant	часть пакета XeTeX
longtable	CTAN:macros/latex/packages/tools
lowcycle	часть пакета XeTeX
lpr	часть системы Unix
lscape	часть пакета graphics
lucbr	CTAN:systems/mac/textures/fonts
m-ch-en	часть пакета ppchtex
m4	часть дистрибутива Unix
Mac GS Viewer	http://www.glyphic.com/glyphic/projects/macgs.html
makeindex	часть дистрибутива TeX
makeinstance	CTAN:fonts/utilities/mm
makempx	часть дистрибутива METAPOST
MakeTeXPK	часть основанных на Web2c дистрибутивов TeX
MakeTeXTFM	часть основанных на Web2c дистрибутивов TeX
Maple	http://www.maplesoft.on.ca/Products/MapleV/MapleV.html
Mathematica	http://www.mathematica.com
mathptm	часть пакета PSNFSS
MATLAB	http://www.mathworks.com
METAFONT	часть дистрибутива TeX
METAPOST	часть дистрибутива TeX или CTAN:graphics/metapost
mfpic	CTAN:graphics/mfpic
mftoeps	CTAN:graphics/MF-PS/mftoeps
midi2tex	CTAN:support/midi2tex
mmafm	CTAN:fonts/utilities/mm
mpp	CTAN:support/mpp
mptotex	часть дистрибутива METAPOST
mtimes	часть пакета psnfss-extra

Таблица B.1. *Продолжение.*

Имя	*Адрес*
multido	часть дистрибутива PSTricks
MusicTEX	CTAN:macros/musictex
music	часть Unix Documentors Workbench
musixflx	часть пакета MusiXTEX
MusiXTEX	CTAN:macros/musixtex/taupin
nassflow	CTAN:macros/latex209/contrib/nassflow
Netscape	http://www.netscape.com
newcent	часть пакета PSNFSS
nfssfont.tex	CTAN:macros/latex/base
nimbus	CTAN:fonts/psfonts/urwvf CTAN:fonts/psfonts/urw/nimbus
OzTeX	CTAN:systems/mac/oztex
palatino	часть пакета PSNFSS CTAN:fonts/psfonts/adobe/palatino
pbmtopk	CTAN:graphics/pbmtopk
pctex32	http://www.pctex.com
pctexhp	http://www.pctex.com
pctexps	http://www.pctex.com
pctexwin	http://www.pctex.com
pdf2ps	часть дистрибутива Ghostscript
pict2e	CTAN:macros/latex/base
PICTEX	CTAN:graphics/pictex
pic	часть Unix Documentors Workbench
pie-chart.sh	часть дистрибутива PSTricks
pifont	часть пакета PSNFSS
pktogf	часть дистрибутива TEX
pktype	часть дистрибутива TEX
pltotf	часть дистрибутива TEX
PMX	CTAN:support/pmx
pnmcrop	часть пакета netpbm
ppchtex	CTAN:macros/context/ppchtex
ppmtogif	часть пакета netpbm
ps2ascii	часть дистрибутива Ghostscript
ps2epsi	часть дистрибутива Ghostscript
ps2pdf	часть дистрибутива Ghostscript
ps2pk	CTAN:fonts/utilities/ps2pk
psbook	часть коллекции утилит psutils
PSfrag	CTAN:macros/latex/contrib/supported/psfrag
psmerge	часть коллекции утилит psutils
PSNFSS	CTAN:macros/latex/packages/psnfss
psnup	часть коллекции утилит psutils
pspicture	CTAN:macros/latex/contrib/other/misc/pspicture.sty
psprint	CTAN:dviware/psprint
psresize	часть коллекции утилит psutils
psselect	часть коллекции утилит psutils
pst-3d	часть дистрибутива PSTricks
pst-char	часть дистрибутива PSTricks
pst-coil	часть дистрибутива PSTricks
pst-eps	часть дистрибутива PSTricks
pst-fill	часть дистрибутива PSTricks
pst-grad	часть дистрибутива PSTricks
pst-node	часть дистрибутива PSTricks
pst-plot	часть дистрибутива PSTricks
pst-text	часть дистрибутива PSTricks
pst-tree	часть дистрибутива PSTricks
pstcol	часть пакета graphics

Таблица B.1. *Продолжение.*

Имя	*Ад ес*
pstops	часть коллекции утилит psutils
pstotext	http://www.research.digital.com/SRC/virtualpaper/pstotext.html
pstpoly	часть дистрибутива PSTricks
pstricks	часть дистрибутива PSTricks
PSTricks	CTAN:graphics/pstricks
psutils	CTAN:support/psutils
ps_view	CTAN:support/ps_view
pxltopk	часть дистрибутива TEX
rboxes	часть дистрибутива METAPOST
rotating	CTAN:macros/latex/contrib/supported/rotating
Scientific Word	http://www.thomson.com/tcisoft/products/swver25.html
semcolor	часть дистрибутива seminar
seminar	CTAN:macros/latex/contrib/other/seminar
showchar	часть коллекции утилит psutils
slidesec	часть дистрибутива seminar
slides	CTAN:macros/latex/base
SliTEX	CTAN:macros/latex209/distribs/latex/general
sprite	CTAN:graphics/bit2spr
SPSS	http://www.spss.com/
t1utils	CTAN:fonts/utilities/t1utils
teTeX	CTAN:systems/unix/teTeX
TeX-PS	CTAN:macros/generic/TeX-PS
TeXcad	часть дистрибутива emTeX
Textures	http://www.bluesky.com/textures.html
tftopl	часть дистрибутива TEX
times	часть пакета PSNFSS CTAN:fonts/psfonts/adobe/times
timing	CTAN:macros/latex/contrib/other/timing
trees	CTAN:macros/latex/contrib/supported/treesvr
trig	часть пакета graphics
troff	часть дистрибутива GNU groff (Unix)
TrueTeX	http://styx.ios.com/~kinch/truetex.htm
utopia	часть пакета PSNFSS CTAN:fonts/psfonts/adobe/utopia CTAN:fonts/psfonts/xadobe/utopia CTAN:fonts/utopia
vftovp	часть дистрибутива TEX
vptovf	часть дистрибутива TEX
VTeX	http://www.micropress-inc.com
Web2c	CTAN:systems/web2c
xdvi	CTAN:dviware/xdvi
xfig	CTAN:graphics/xfig
XYMTEX	CTAN:macros/latex/contrib/other/xymtex
XY-pic	CTAN:macros/generic/diagrams/xypic

Список литературы[1]

Adobe Systems 1990. *PostScript Language Reference Manual.* 2nd edition. Reading, MA: Addison-Wesley.

Aho, A. V., Sethi, R. and Ullman, J. D. 1986. *Compilers: Principles, Techniques and Tools.* Reading, MA: Addison-Wesley.

Appelt, W. 1988. Typesetting chess. *TUGboat*, **9** (3), 284–287.

Arbortext Inc. 1986. *Manual for DVIPS.* (См. заметки LPEXT, распространяемые с этим программным обеспечением).

Bentley, J. 1986. Little languages. *Communications of the ACM*, **29**, 711–721.

Bentley, J. and Kernighan, B. 1984. *GRAP — a language for typesetting graphs.* Computing Science Technical Report 114, AT&T Bell Laboratories, Murray Hill, NJ.

Brüggemann-Klein, A. and Wood, D. 1989. Drawing trees nicely with TEX. *Electronic publishing — origin, dissemination and design*, **3**, 101–115.

Burbank, M. and Goossens, M. 1996. Electronic news from the family. *TUGboat*, **17** (1), 37–42.

Carlisle, D. 1995. *Packages in the 'graphics' bundle.* Электронный документ, распространяемый с пакетом `graphics`.

Clark, A. 1987. Halftone output from TEX. *TUGboat*, **8** (3), 270–274.

Clark, J. 1989. *DVITOPS user manual.* Электронный документ, распространяемый с этим пакетом.

Clark, M. (ed.) 1990. *TEX Applications, Uses, Methods.* Chichester: Ellis Horwood.

Déribéré, M. 1993. *La Couleur — Collection: que sais-je ?, Nr. 220.* Eighth edition. Paris: Presses universitaires de France.

Duggan, A. 1994. Colour separation and PostScript. *TUGboat*, **15** (3), 213–217.

Duplan, P. and Jauneau, R. 1986. *Maquette et mise en page.* Paris: Éditions du Moniteur.

Eck, H. and Küblbeck, S. 1991. *Generating Feynman graphs and amplitudes with* FeynArts. Технический отчет, Physikalisches Institut am Hubland, D-8700 Würzburg. Доступно на сервере MathSource по URL `http://www.wolfram.com/mathsource/`.

Foley, J. D., van Dam, A., Feiner, S. K. and Hughes, J. F. 1990. *Computer graphics, principles and practice.* 2nd edition. Reading, MA: Addison-Wesley.

Fujita, S. 1993. X$^{\text{f}}_{\text{f}}$MTEX*: a macro package for typesetting chemical structural formulas.* Manual of X$^{\text{f}}_{\text{f}}$MTEX for Version 1.00. Электронный документ, распространяемый с этим пакетом.

Fujita, S. 1995. X$^{\text{f}}_{\text{f}}$MTEX for drawing chemical structural formulas. *TUGboat*, **16** (1), 80–88.

Gerritsen, F. 1988. *Evolution in color.* West Chester, PA: Schiffer Publishing Ltd.

Girou, D. 1994. Présentation de PSTricks. *Cahiers GUTenberg*, **16**, 21–70.

Goossens, M. and van Herwijnen, E. 1992. The elementary Particle Entity Notation (PEN) scheme. *TUGboat*, **13** (2), 201–207.

[1] Знаком * отмечены работы, добавленные при переводе. — *Прим. ред.*

Goossens, M., Mittelbach, F. and Samarin, A. 1994. *The LaTeX companion.* Reading, MA: Addison-Wesley. [Имеется русский перевод: Гуссенс М., Миттельбах Ф., Самарин А. Путеводитель по пакету LaTeX и его расширению LaTeX 2ε. — М.: Мир, 1999.]

*Goossens, M. and Rahtz, S. (with Gurari, E., Moore, R. and Sutor, R.) 1999. *The LaTeX Web Companion. Integrating TeX, HTML and XML.* Reading, MA: Addison-Wesley. [Имеется русский перевод: Гуссенс М., Ратц С. Путеводитель по пакету LaTeX и его Web-приложениям. — М.: Мир, 2001.]

*Grätzer, G. 1999. *First Steps in LaTeX.* Boston, MA: Birkhäuser. [Имеется русский перевод: Грэтцер Г. Первые шаги в LaTeX'e. — М.: Мир, 2000.]

Greenwade, G. D. 1993. The Comprehensive TeX Archive Network (CTAN). *TUGboat*, **14** (3), 342–351.

Greenwood, T. G. 1993. International cultural differences in software. *Digital Technical Journal*, **5** (16), 8–20.

Gurari, E. M. 1994. *TeX and LaTeX: Drawing and Literate Programming.* New York: McGraw-Hill.

Haas, R. T. and O'Kane, K. C. 1987. Typesetting chemical structure formulas with the text formatter TeX/LaTeX. *Computers and Chemistry*, **11** (4), 251–271.

Hafner, J. L. 1992. FoilTeX, a LaTeX-like system for typesetting foils. *TUGboat*, **13** (3), 347–356.

Hagen, J. and Otten, A. F. 1996. PPCHTeX: typesetting chemical formulas in TeX. *TUGboat*, **17** (1), 54–66.

Hamilton Kelly, B. 1990. Some macros to draw crosswords. *TUGboat*, **11** (1), 103–119.

Haralambous, Y. and Rahtz, S. 1995. LaTeX, hypertext and PDF, *or* the entry of TeX into the world of hypertext. *TUGboat*, **16** (2), 162–173.

Hassel, F. 1993. Schachfigurensatz mit TeX und LaTeX. *Die TeXnische Komödie*, **5** (4), 11–25. Английский перевод распространяется с этим пакетом как электронный документ.

Hobby, J. D. 1992. *A user's manual for MetaPost.* Computing Science Technical Report 162, AT&T Bell Laboratories.

Hobby, J. D. 1993. *Drawing graphs with MetaPost.* Computing Science Technical Report 164, AT&T Bell Laboratories.

Hoenig, A. 1997. *TeX Unbound: Strategies for Fonts, Graphics, and More.* New York: Oxford University Press.

International Union of Pure and Applied Chemistry 1979. *Nomenclature of Organic Chemistry.* Oxford: Pergamon.

International Union of Pure and Applied Physics 1987. Symbols, units, nomenclature and fundamental constants in physics. *Physica*, **146A**, 1–67.

Itten, J. 1974. *Art of Colour.* New York: Von Nostrand Reinhold.

Jackowski, B. 1995. A METAFONT-EPS interface. *TUGboat*, **16** (4), 388–395.

Jackson, R., MacDonald, L. and Freeman, K. 1994. *Computer Generated Colour.* New York: Wiley.

Jalbert, F. 1989. *MuTeX user's guide.* Электронный документ, распространяемый с этим пакетом.

Judd, D. B. and Wyszecki, G. 1963. *Color in Business, Science, and Industry.* 2nd edition. New York: Wiley.

Karney, C. 1988. *Getting TeX labelling on graphics.* Электронное сообщение в texhax, 24 May 1989.

*Karow, P. 1994. *Font Technology. Description and Tools.* Berlin: Springer Verlag. [Имеется русский перевод: Каров П. Шрифтовые технологии. Описание и инструментарий. — М.: Мир, 2001.]

Kernighan, B. 1984. *PIC — a graphics language for typesetting.* Computing Science Technical Report 116, AT&T Bell Laboratories, Murray Hill, NJ.

Knappen, J. 1992. Changing the appearance of math. In Zlatuška (1992), pp. 212–216.

Knuth, D. E. 1986a. *The TEXbook*, volume A of *Computers and Typesetting.* Reading, MA: Addison-Wesley. [Имеется русский перевод: Кнут Д. Все про TEX. — Протвино: RDTEX, 1993.]

Knuth, D. E. 1986b. *TEX: The Program*, volume B of *Computers and Typesetting.* Reading, MA: Addison-Wesley.

Knuth, D. E. 1986c. *The METAFONTbook*, volume C of *Computers and Typesetting.* Reading, MA: Addison-Wesley.

Knuth, D. E. 1986d. *METAFONT: The Program*, volume D of *Computers and Typesetting.* Reading, MA: Addison-Wesley.

Knuth, D. E. 1986e. *Computer Modern Typefaces*, volume E of *Computers and Typesetting.* Reading, MA: Addison-Wesley.

Knuth, D. E. 1987. Fonts for digital halftones. *TUGboat*, **8** (2), 135–160. (См. также Knuth (1999).)

Knuth, D. E. 1989. Typesetting *Concrete Mathematics*. *TUGboat*, **10** (1), 31–36. (См. также Knuth (1999).)

Knuth, D. E. 1990. Virtual fonts: more fun for grand wizards. *TUGboat*, **11** (1), 13–23. (См. также Knuth (1999).)

Knuth, D. E. 1993. Icons for TEX and METAFONT. *TUGboat*, **14** (4), 387–389. (См. также Knuth (1999).)

Knuth, D. E. 1999. *Digital Typography.* Los Angeles, CA: CSLI. [Имеется русский перевод: Кнут Д. Компьютерная типография. — М.: Мир, 2002.]

Kunkel, G. 1990. *Graphic Design with PostScript.* Glenview, IL: Scott, Foresman.

Kuykens, H. J. P. and Verbruggen, A. H. 1992. *MIDI2TEX, a MIDI to* MusicTEX *translation program.* Электронный документ, распространяемый с этим пакетом.

Lamport, L. 1994. *LATEX: A Document Preparation System.* 2nd edition. Reading, MA: Addison-Wesley.

Levine, M. J. S. 1990. A LATEX graphics routine for drawing Feynman diagrams. *Computer Physics Communications*, **58**, 181–198.

Maclenan, M. P. and Burns, G. M. 1991. An approach to drawing circuit diagrams for text books. *TUGboat*, **12** (1), 66–69.

McGilton, H. and Campione, M. 1992. *PostScript by Example.* Reading, MA: Addison-Wesley.

Merz, T. 1996. *PostScript & Acrobat/PDF.* Berlin: Springer Verlag.

Murray, J. D. and van Ryper, W. 1996. *Encyclopedia of Graphics File Formats.* 2nd edition. Sebastopol, CA: O'Reilly & Associates, Inc.

Nemcsics, A. 1993. *Colour Dynamics — Environmental Colour Design.* New York: Ellis Horwood.

Nieuwenhuizen, J. and Nienhuys, H.-W. 1996. *MusiXTEX pre-processor—using TEX and the MusiXTEX macro package to write parts and scores of music.* Электронный документ, распространяемый с этим пакетом.

Norris, A. C. and Oakley, A. L. 1990. Electronic publishing and chemical text processing. In Clark (1990).

Ohl, T. 1995. Drawing Feynman Diagrams with LATEX and METAFONT. *Computer Physics Communications*, **90**, 340–354.

Ohl, T. 1996. *feynMF, Drawing Feynman Diagrams with LATEX and METAFONT.* Version 1.05. Электронный документ, распространяемый с этим пакетом.

Olejniczak-Burkert, R. 1989. texpic—design and implementation of a picture graphics language in TEX. *TUGboat*, **10** (4), 627–637.

Podar, S. 1986. *Enhancements to the picture environment of LATEX.* Технический отчет 86-17, Dept. of Computer Science, State University of New York, Stony Brook, NY. version 1.2.

Rahtz, S. 1987. The Protestant Cemetery, Rome: a study undertaken under the auspices of the Unione Internazionale degli Istituti di Archeologia, Storia e Storia dell'Arte in Roma. *Opuscula Romana*, **16**, 149–167.

Ramek, M. 1990. Chemical structure formulae and x/y diagrams with TEX. In Clark (1990).

Reid, G. C. 1988. *PostScript Language Program Design.* Reading, MA: Addison-Wesley.

Reid, T. and Hosek, D. 1989. Report from the DVI Driver Standards Committee. *TUGboat*, **10** (2), 188–191.

Rogers, D. 1989. Computer graphics and TEX—a challenge. *TUGboat*, **10** (1), 39–44.

Rokicki, T. 1994. Advanced 'special' support in a dvi driver. *TUGboat*, **15** (3), 205–212.

Rose, K. H. 1992. How to typeset pretty diagram arrows with TEX—design decisions used in XY-pic. In Zlatuška (1992), pp. 183–190.

Rose, K. H. 1995. XY-pic *user's guide. Version 3.0.* Электронный документ, распространяемый с этим пакетом.

Rose, K. H. and Moore, R. 1995. XY-pic *reference manual. Version 3.2.* Электронный документ, распространяемый с этим пакетом.

Rubinstein, Z. 1989. Chess printing via METAFONT and TEX. *TUGboat*, **10** (2), 170–172.

Salmon, R. and Slater, M. 1987. *Computer Graphics — Systems & Concepts.* Amsterdam: Addison-Wesley Europe.

Schofer, A. and Steinbach, A. 1987. *Automatisierter Notensatz mit TEX.* Технический отчет, Rheinische Friedrich-Wilhelms-Universität, Bonn.

Simons, D. 1995. *PMX, a preprocessor for MusiXTEX. Version 0.92.* Электронный документ, распространяемый с этим пакетом.

Sofka, M. 1994. Color book production using TEX. *TUGboat*, **15** (3), 228–238.

Sowa, F. 1994. Printing colour pictures. *TUGboat*, **15** (3), 223–227.

Tannert, S. and Tille, A. 1996. *The CIRC package.* Электронный документ, распространяемый с этим пакетом.

Taupin, D. 1992. MusicTEX: using TEX to write polyphonic or instrumental music. In Zlatuška (1992), pp. 257–272.

Taupin, D. 1993a. MusicTEX: using TEX to write polyphonic or instrumental music. *TUGboat*, **14** (3), 203–211.

Taupin, D. 1993b. Using TEX and METAFONT to build complicated maps. *TUGboat*, **14** (3), 196–202.

Taupin, D. 1995a. MusiXTEX, even more beautiful than MusicTEX for music typesetting. In W. Dol (ed.), *Proceedings of the 9th European TEX Conference, September 4–8 1995, Arnhem, The Netherlands*, pp. 351–358. Nederlandstalige TEX Gebruikersgroep.

Taupin, D. 1995b. *MusicTEX, using TEX to write polyphonic and instrumental music, Version 5.14.* Электронный документ, распространяемый с этим пакетом.

Taupin, D., Mitchell, R. and Egler, A. 1996. *MusiXTEX, using TEX to write polyphonic and instrumental music, Version T.42.* Электронный документ, распространяемый с этим пакетом.

TUG Working Group on the TEX Directory Structure (TWG-TDS) 1995. A Directory Structure for TEX Files (Version 0.999). *TUGboat*, **16** (4), 401–413.

Tutelaers, P. 1992. A font and a style for typesetting chess using LATEX or TEX. *TUGboat*, **13** (1), 85–90.

Unicode Consortium 1996. *Unicode Standard; worldwide character encoding, version 2.0.* Reading, MA: Addison-Wesley.

Valiente Feruglio, G. 1994. Typesetting commutative diagrams. *TUGboat*, **15** (4), 466–484.

van der Laan, C. G. 1989. Typesetting bridge via LaTeX. *TUGboat*, **10** (1), 113–116.

van Zandt, T. 1993a. *PSTricks user's guide.* Неопубликованная документация к версии 0.93a этого пакета.

van Zandt, T. 1993b. *seminar.sty A LaTeX style for slides and notes. User's Guide. Version 1.0.* Неопубликованная документация.

van Zandt, T. 1993c. *Documentation for colortab.tex/colortab.sty: shading and coloring TeX tables. Version 0.9.* Неопубликованная документация.

van Zandt, T. and Girou, D. 1994. Inside PSTricks. *TUGboat*, **15** (3), 239–246.

Vanderburg, G. L. and Reid, T. J. 1987. `\special` issues. *TUGboat*, **8** (2), 291–300.

Vermaseren, J. A. M. 1994. Axodraw. *Computer Physics Communications*, **83**, 45–58.

Vulis, M. 1996. VTeX user guide. Руководство по пакету VTeX.

Waldschmidt, H. 1988. An editor for constructing graphics with TeX. *Computer Graphics Forum*, **6** (4), 359–364.

Walshaw, C. 1996. *ABC2MTeX, an easy way of transcribing folk and traditional music.* Электронный документ, распространяемый с этим пакетом.

Wanske, H. 1990. Notenproduktion im Umbruch. Gedanken zur gegenwärtigen und zukünftigen Musikalienherstellung. In H.-J. Koppitz (ed.), *Gutenberg-Jahrbuch 1990*, pp. 237–243. Mainz, Germany: Gutenberg-Gesellschaft, Internationale Vereinigung für Geschichte und Gegenwart der Druckkunst e.V.

White, J. V. 1990. *Color for the Electronic Age.* New York: Watson-Guptil Publications.

Wichura, M. J. 1987. *The* PiCTeX *manual.* Number 6 in TeXniques: publications for the TeX community. Providence, RI: TeX Users Group.

Wichura, M. J. 1988. Macros for drawing PiCtures. *TUGboat*, **9** (2), 193–197.

Zlatuška, J. (ed.) 1992. *EuroTeX '92: Proceedings of the 7th European TeX Conference, Prague, Czechoslovakia, September 14–18, 1992.* Brno: Masarykova Universita.

Котельников И. А., Чеботаев П. З. 1998. *Издательская система LaTeX 2ε.* — Новосибирск: Сибирский хронограф.

Львовский С. М. 1995. *Набор и верстка в пакете LaTeX*, 2-е изд. — М.: Космосинформ.

Именной указатель

Предметный указатель

Этот указатель состоит из двух частей. В первой части перечислены по алфавиту все упомянутые в книге пакеты и программы вместе с использованными в них командами, опциями, нотациями и т. д. Вторая часть, начиная со с. 609, разбита на разделы по пакетам, программам, файлам и т. п.

По всему указателю полужирным шрифтом выделены страницы, на которых имеется важная информация, например точное определение команды или подробное описание пакета. Обычные ссылки набраны светлым шрифтом.

* * *

Список таблиц

Список иллюстраций

Цветные вклейки

Оглавление

Справочное издание

Мишель Гуссенс, Себастиан Ратц, Франк Миттельбах

Путеводитель по пакету LaTeX
и его графическим расширениям

Иллюстрирование документов
при помощи TeX'а и PostScript'а

Зав. редакцией И. А. Маховая
Художник И. И. Куликова
Художественный редактор В. П. Григорьев
Корректор Е. Н. Клитина
Оригинал-макет подготовлен О. Г. Лапко в пакете LaTeX 2_ε
с использованием семейства шрифтов Computer Modern
с кириллическим расширением LH

Подписано к печати 22.03.02 г. Формат 70 × 100/16.
Печать офсетная. Объем 19,5 + 0,5 вкл. бум. л.
Усл.-печ. л. 50,7 + 1,3 вкл. Уч.-изд. л. 43,85 + 0,75 вкл.
Изд. № 20/9743. Тираж 5000 экз. Заказ 2732

Издательство «Мир»
Министерства РФ по делам печати,
телерадиовещания и средств массовых коммуникаций
107996, ГСП-6, Москва, 1-й Рижский пер., 2
при участии издательства «Бином ЛЗ»
Адрес для переписки: 103473, Москва, а/я 9
Телефон (095)955-0398. E-mail ibz@aha.ru

Диапозитивы изготовлены
в издательстве «Мир»

Отпечатано с готовых диапозитивов в
полиграфической фирме «Полиграфист»
160001, г. Вологда, ул. Челюскинцев, 3

Лучшие книги мира — в издательстве «МИР»

Серия: Библиотека издательских TEXнологий

Выпускаемая издательством «Мир» серия посвящена знаменитой издательской системе TEX и родственным вопросам. Все программное обеспечение, относящееся к этой системе, является свободным или условно свободным, что делает ее особенно привлекательной для всех, кто имеет доступ к Интернету.

По книгам этой серии новичок научится полиграфически грамотно и красиво оформлять свои научные работы с формулами, графиками и таблицами; он также узнает, в каком виде удобнее и быстрее пересылать по электронной почте статью в научный журнал или письмо о научном открытии коллеге.

Тот, кто уже знаком с издательской системой TEX, найдет здесь много новых и полезных сведений о том, как подготовить оригинал-макет научного издания с таблицами, формулами, указателем, списком литературы, иллюстрациями, в том числе цветными, о технологиях изготовления компьютерных шрифтов, о возможностях представления научных работ в сети Интернет и о многом другом.

Готовится к выпуску:

Кнут Д. **КОМПЬЮТЕРНАЯ ТИПОГРАФИЯ:** Пер. с англ. — 52 л., ил.

Сборник работ Дональда Кнута, написанных им за время работы над созданием всемирно известных систем TEX и METAFONT, в который вошли также статьи последних лет по этой тематике. Книга состоит из 34 глав, разделенных условно на три раздела: TEX и относящиеся к нему темы; METAFONT и родственные вопросы; TEX и METAFONT в историческом аспекте. Представлен богатый иллюстративный материал (более 500 рисунков, среди них фотографии, графики, факсимиле). Приводятся тексты программ.

Как и во всех книгах Дональда Кнута весьма серьезные вопросы излагаются просто и увлекательно, что, учитывая междисциплинарную направленность книги, делает ее доступной для специалистов в разных областях науки.

Книга представляет интерес для научных работников всех специальностей, самостоятельно готовящих свои работы к публикации, для специалистов в области информатики и издательских систем, а также для математиков, интересующихся нестандартными приложениями.

Адрес: 107996, ГСП-6, Москва, 1-й Рижский пер., 2
Контактные телефоны: (095)286-83-88, 286-82-33, 286-25-50
Факс: (095)286-84-55 ✧ **E-mail:** victor@mir.msk.su ✧ **http://www.mir-pubs.dol.ru**