

Компьютерная графика и издательские системы

к.ф.-м.н., доц. Журенков Олег Викторович, zhur@pie-aael.ru

кафедра прикладной информатики в экономике, государственном и
муниципальном управлении (314С), тел. 296–547

Оглавление

1 Компьютерная графика	5
1. Основные понятия	5
1.1. История компьютерной графики	5
1.2. Основные области применения	6
1.3. Классификация	7
1.4. Графические редакторы	8
1.5. Выводы	9
2. Векторная графика	9
2.1. Достоинства и недостатки	9
3. Растровая графика	10
3.1. Дополнительные характеристики	10
3.2. Цветовые палитры	11
3.3. Интенсивность тона	14
3.4. Динамический диапазон	15
3.5. Гамма-коррекция	16
3.6. Альфа-композиция	16
4. Фрактальная графика	17
5. Форматы файлов	18
5.1. Векторные файлы	18
5.2. Алгоритмы сжатия	19
5.3. Растровые файлы	20
6. Трёхмерная графика	32
6.1. Рендеринг	33
6.2. Методы визуализации	33
6.3. Шейдеры	34
7. Конвертеры файлов	39
7.1. NetPBM	39
7.2. ImageMagick	40
8. Деловая и научная графика	41
8.1. Редактор научной графики GNUplot	42
9. Полезные ссылки	45
2 Издательские системы	49
1. Обзор издательских систем (ИС)	49
1.1. Системы визуального проектирования	49
1.2. Системы логического проектирования	50
1.3. \LaTeX	50
3 Введение в \LaTeX	52
1. Рабочий процесс \LaTeX	52
2. Команды и аргументы	55
2.1. Логосы и декларации	56
2.2. Правила скобок	56
3. Буквы и символы	56

3.1.	Специальные символы	57
3.2.	Акценты	57
3.3.	Тире, кавычки, многоточия	59
4.	Шрифты	60
4.1.	Пользовательские команды выбора шрифтов	60
4.2.	Пользовательские команды изменения размера шрифтов	61
5.	Низкоуровневые команды задания шрифтов	61
5.1.	Кодировка	62
5.2.	Непосредственное задание символа	62
5.3.	Гарнитура	63
5.4.	Насыщенность	63
5.5.	Начертание	65
5.6.	Кегль	65
5.7.	Переключение шрифтов	65
4	Вёрстка и форматирование	67
1.	Управление строками	67
1.1.	Горизонтальные пробелы	67
1.2.	Переносы	69
1.3.	Разрыв строки	69
1.4.	Абзацы	69
2.	Управление страницами	70
2.1.	Вертикальные пробелы	70
2.2.	Страницы	70
3.	Форматирование текста	71
3.1.	Выравнивание	71
3.2.	Цитаты и стихи	72
3.3.	Списки	72
3.4.	Неформатированный текст	72
3.5.	Сноски и заметки	73
4.	Боксы	73
4.1.	Строковые боксы	73
4.2.	Текстовые боксы	74
4.3.	Линейные боксы	75
4.4.	Сохранение бокса	75
5.	Команды в L ^A T _E X 2 _ε	76
5.1.	Командные длины	76
5.2.	Счётчики	77
6.1.	Определение новых команд	79
6.2.	Определение новых окружений	80
6.3.	Определение команд типа «теорема»	80
5	Печатный документ	82
1.	Структура документа	82
1.1.	Преамбула	82
1.2.	Титульный лист	82

1.3.	Аннотация (abstract)	83
1.4.	Секционирование (рубрикация)	83
1.5.	Оглавление (содержание)	84
1.6.	Библиография (список литературы)	84
2.	Механизм перекрёстного цитирования	85
3.	Большие документы	86
3.1.	Условная компиляция	86
3.2.	Включение файлов	86
4.	Стиль документа	87
4.1.	Стиль страницы	87
4.2.	Стандартные классы	89
6	Математика в \LaTeX'е	92
1.	Основные понятия	92
1.1.	Математические моды (режимы)	92
1.2.	Пробелы в математических формулах	93
2.	Алфавит математики	94
2.1.	Математические акценты	94
2.2.	Греческие буквы	95
2.3.	Бинарные операторы	95
2.4.	Символы сравнения	95
2.5.	Большие операторы и символы переменного размера	95
2.6.	Разделители	98
2.7.	Стрелки	99
2.8.	Функции	99
2.9.	Прочие символы	101
3.	Основные структуры	101
3.1.	Верхние и нижние индексы	101
3.2.	Корни	102
3.3.	Дроби	102
3.4.	Биномиальные коэффициенты	103
3.5.	Размещение объектов друг над другом	103
3.6.	Знаки пунктуации и многоточия	105
3.7.	Формулы в рамке	105
3.8.	Условный выбор	105
3.9.	Матрицы	105
4.	Большие формулы	107
4.1.	Уравнения	107
4.2.	Сложные формулы	108
4.3.	Позиционирование	109
4.4.	Нумерация и ссылки	109
5.	Шрифты	110
5.1.	Классы \mathcal{AMS} - \LaTeX	111

7	Таблицы в \LaTeX'е	112
1.	Табулятор	112
2.	Таблица	113
2.1.	Параметры настройки	115
3.	Размещение таблиц	115
8	Графика в $\LaTeX 2_{\epsilon}$	118
1.	Основные понятия	118
2.	Импортирование графики	118
3.	Размещение рисунков	122
4.	Псевдографика	124
4.1.	Позиционирование	124
4.2.	Линии	125
4.3.	Круги	125
4.4.	Овалы	126
4.5.	Боксы	126
5.	Цвет	128
5.1.	Определение цвета	128
5.2.	Использование цветов	128

Глава 1

Компьютерная графика

1. Основные понятия

Компьютерная графика (машинная графика) — область деятельности, в которой компьютеры используются в качестве инструмента как для создания изображений, так и для обработки визуальной информации, полученной из реального мира.

Также **компьютерной графикой** называют результат такой деятельности.

1.1. История компьютерной графики

Первые вычислительные машины не имели отдельных средств для работы с графикой, однако уже использовались для получения и обработки изображений. Программируя память первых ЭВМ, построенную на основе матрицы ламп, можно было получать узоры.

В 1961 г. программист *Стив Рассел (Stephen “Slug” Russell)* возглавил проект по созданию первой компьютерной игры с графикой. Создание игры «Spacewar!» («Космическая война») заняло около 200 человеко-часов. Игра была создана на машине PDP-1.

В начале 1960-х гг. американский учёный *Айвен Сазерленд (Ivan Edward Sutherland)* создал программно-аппаратный комплекс **Sketchpad**, который позволял рисовать точки, линии и окружности на трубке цифровым пером. Поддерживались базовые действия с примитивами: перемещение, копирование и др. По сути, это был первый векторный редактор, реализованный на компьютере. Также программу можно назвать первым графическим интерфейсом, причём она являлась таковой ещё до появления самого термина.

В 1960-х гг. появились разработки в промышленных приложениях **компьютерной графики**. *Норман Тейлор (Norman Taylor)*, *Джек Гилмор (Jack Gilmore)* и др. из фирмы Itek в 1962 г. разработали *цифровую электронную чертёжную машину EDM*, основанную на PDP-1.



Рис. 1.1. «Spacewar!» на компьютере PDP-1

В 1964 г. General Motors совместно с IBM представила *систему автоматизированного проектирования DAC-1*.

В 1968 г. группой под руководством *Н. Н. Константинова* была создана компьютерная математическая модель движения кошки. БЭСМ-4, выполняя написанную программу решения дифференциальных уравнений, рисовала мультфильм «Кошечка», и это для своего времени было прорывом. Для визуализации использовался алфавитно-цифровой принтер.



Рис. 1.2. Кадр мультфильма «Кошечка»

Стремительный прогресс **компьютерной графики** начался с появлением возможности запоминать изображения и выводить их на компьютерном дисплее (электронно-лучевой трубке).

1.2. Основные области применения

Разработки в области **компьютерной графики** сначала развивались лишь в научных учреждениях. Постепенно **компьютерная графика** прочно вошла в по-

вседневную жизнь, стало возможным вести коммерчески успешные проекты в этой области.

Основные сферы применения технологий **компьютерной графики**:

- *графический интерфейс пользователя;*
- *спецэффекты, визуальные эффекты (VFX), цифровая кинематография;*
- *компьютерная графика для кино и телевидения;*
- *цифровое телевидение, Всемирная паутина, видеоконференции;*
- *компьютерные игры, системы виртуальной реальности (например, тренажёры управления самолётом);*
- *цифровая фотография и существенно возросшие возможности по обработке фотографий;*
- *визуализация научных и деловых данных;*
- *системы автоматизированного проектирования;*
- *компьютерная томография;*
- *лазерная графика.*

1.3. Классификация

По способам задания изображений можно выделить категории:

- **двумерная (2D) графика**:
 - **растровая**;
 - **векторная**;
 - **фрактальная**;
- **трёхмерная (3D) графика**.

Все графические файлы и программы для работы с ними можно разделить на *векторные* и *растровые*.

Всякое изображение в **растровой графике** рассматривается как совокупность точек разного цвета.

*Графическая информация в **растровой графике** — это совокупность данных о цвете каждого пикселя.*

***Пиксель** (от *picture element* или *picture cell*) — наименьший логический элемент двумерного цифрового изображения в **растровой графике**.*

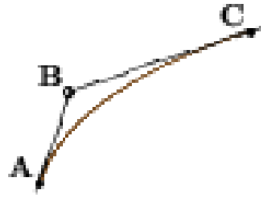


Рис. 1.3. Растровый рисунок с демонстрацией пикселизации (хорошо видны пиксели изображения)

Пиксель представляет собой неделимый объект прямоугольной (обычно квадратной) или круглой формы, обладающий определённым цветом и, возможно, прозрачностью. Растровое компьютерное изображение состоит из пикселей, расположенных по строкам и столбцам.

Векторный подход рассматривает изображение как совокупность простых элементов: отрезков, дуг, эллипсов, прямоугольников и пр., которые называются **графическими примитивами**.

Графическая информация в векторной графике — это данные, однозначно определяющие все графические примитивы, составляющие рисунок.

Например, кривая на рис. 1.4 задана командой `\qBezier(5,5)(15,35)(75,55)`.

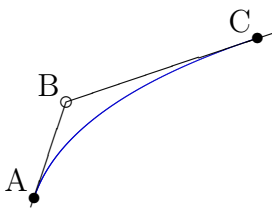


Рис. 1.4. Векторный рисунок, из которого был получен рис. 1.3

1.4. Графические редакторы

Для создания и редактирования рисунков на компьютере используются *графические редакторы*. Графические редакторы также разделяются на растровые и векторные.

Редактирование растровых файлов заключается в изменении значений цветов пикселей с помощью различных инструментов и графических функций (т. н. *фильтров*), а также вырезания/копирования/вставки фрагментов растрового изображения.

Наиболее известные представители этого семейства программ — Adobe Photoshop, Corel Photopaint, GIMP.

Работа в векторных редакторах напоминает работу с конструктором: в любое время можно внести изменения в рисунок, изменив свойства объектов, добавив или удалив объект.

Наиболее известные представители данного класса ПО — Corel Draw, Adobe Illustrator, Adobe InDesign, Inkscape, sK1, Adobe Flash, f4l.

3D редакторы — тоже векторные.

Наибольшую известность получили такие редакторы, как 3Ds Max, Maya, Cinema 4D, Bryce, Modo, Blender, Unity.

1.5. Выводы

Рисовать сложные графические изображения, особенно когда не известно разрешение окончательного устройства вывода, удобнее в векторном редакторе. Размер такого файла, как правило, в несколько раз меньше растрового.

Обрабатывать полноцветные рисунки, редактировать фотоизображения (с уже заданными размерами и разрешением) лучше в редакторе растровой графики.

2. Векторная графика

В графических файлах векторного формата содержатся описания графических примитивов, составляющих рисунок.

В векторном представлении буква К — это три линии (см. пример .1). Всякая линия описывается указанием координат её концов, например, `line(X1,Y1,X2,Y2)`. Тогда изображение буквы К можно описать следующим образом: `line(4,2,4,8) line(5,5,8,2) lin`

Для цветного изображения, кроме координат, указывается ещё один параметр — цвет линии.

Графические файлы векторных форматов содержат информацию о линиях и областях в виде уравнений кривых разного порядка и различных графических примитивов с указанием необходимых параметров.

Положение и форма графических примитивов задаются в **системе графических координат**, связанных с экраном.

Обычно начало координат расположено в верхнем левом углу экрана. Горизонтальная ось X направлена слева направо; вертикальная ось Y — сверху вниз.

Отрезок прямой линии однозначно определяется указанием координат его концов; окружность — координатами центра и радиусом; многоугольник — координатами его углов; закрашенная область — граничной линией, типами линий и заливки, цветом линии и заливки и пр.

2.1. Достоинства и недостатки

К достоинствам векторной графики можно отнести следующие её свойства:

- изображения в векторных форматах *не зависят от разрешения* устройства вывода;
- графические файлы векторного типа имеют относительно *небольшие размеры*;
- векторные изображения легко *масштабируются без потери качества*.

Основным недостатком векторной графики является то, что она *не позволяет получать изображения фотографического качества*.

Замечание: Различие в представлении графической информации в растровом и векторном форматах существует лишь для графических файлов. При выводе на экран любого изображения в видеопамяти формируется информация растрового типа, содержащая сведения о цвете каждого пикселя.

3. Растровая графика

ГОСТ 27459-87:

***Пиксель** — наименьший элемент поверхности визуализации, которому может быть независимым образом заданы цвет, интенсивность и другие характеристики изображения.*

Растровые файлы содержат последовательный набор «цветовых описаний» всех точек.

Для монитора эти точки называются **пикселями** (pixels), а для принтера и сканера — *точками* (dots), заполняющими «холст».

В связи с этим **разрешения** устройств выражают в **ppi** (pixels per inch) или в **dpi** (dots per inch).

3.1. Дополнительные характеристики

***Зернистость** — размер пикселя монитора.*

***Растр** — изображение, построенное из отдельных элементов (точек), как правило, расположенных регулярно.*

В большинстве приложений компьютерной графики растровое изображение представляется двумерным массивом пикселей.

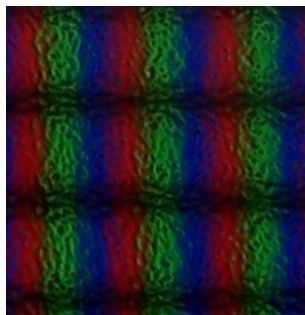


Рис. 1.5. Фрагмент матрицы ЖК монитора (0,78 × 0,78 мм), увеличенный в 46 раз

Растр в технических устройствах (в системах отображения графической информации) — последовательность строк, возникающая в результате работы системы развёртки (печати).

Линиатура — плотность **растра** принтера или сканера, измеряется в *lpi* (*lines per inch*), или в «линиях-на-сантиметр».

Переводной коэффициент — 2,54 (150 lpi = 59 л/см).

Линиатура — параметр, характеризующий растровую структуру количеством линий на единицу длины.

Глубина цвета (качество цветопередачи, битность изображения) — количество бит для задания любого цвета палитры при кодировании одного пикселя.

Измеряется **глубина цвета** в bpp (*bits per pixel*), задающее точное количество используемых бит для представления цвета.

В связи с этой характеристикой, **цветовые палитры** характеризуются **глубиной цвета**.

3.2. Цветовые палитры

Существует несколько основных **цветовых палитр**:

- **BW** (чёрно-белая) — 1 бит.
- **CGA** (4 градации серого) — 2 бита.
- 8-цветная — 3 бита.

Эту палитру использовали устаревшие персональные компьютеры с TV-выходом.

- **EGA** (16-цветная) — 4 бита.
- 256 цветов — 8 бит = 1 байт.

8-битные видеорежимы появились вместе с ростом объёмов памяти компьютеров. Основное своё распространение получили с конца 1980-х гг. В середине 1990-х гг., с появлением доступных 1–2-мегабайтных видеоплат, на рабочих столах ОС 8-битные режимы уступили пальму первенства 16-битным.

В играх они продержались несколько дольше из-за высокой скорости, например StarCraft (1998) работал в режиме $640 \times 480 \times 8$ и не замедлялся на компьютерах класса Pentium-100 даже в массовых боях. Вышедший в 2000 г. Grand Prix 3 использовал 8-битные режимы в программном **рендеринге**.

Широкое распространение получили лишь некоторые 8-битные **палитры**.

- **Индексированная** (≤ 256 цветов) — ≤ 8 бит.

Из широкого цветового пространства выбираются любые цвета. Их значения хранятся в специальной таблице — **палитре**. В каждом из пикселей изображения хранится номер цвета в **палитре** (от 0 до 255).

- **Grayscale** (серая) — 256 оттенков серого.
- **Однородные палитры** — 256 оттенков одного цвета.

- **RGB**, **HSB**, ... (16 777 216 цветов) — 3 байта.

Red, Green, Blue — аддитивная цветовая модель, как правило описывающая способ синтеза цвета для цветовоспроизведения. Аддитивной она называется потому, что цвета получаются путём добавления к чёрному.

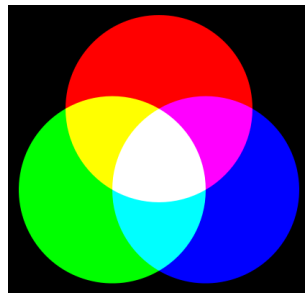


Рис. 1.6. Аддитивное смешение цветов

Иначе говоря, если цвет экрана, освещённого цветным прожектором, обозначается в **RGB** как (r_1, g_1, b_1) , а цвет того же экрана, освещённого другим прожектором, — (r_2, g_2, b_2) , то при освещении двумя прожекторами цвет экрана будет $(r_1+r_2, g_1+g_2, b_1+b_2)$.

Изображение в данной цветовой модели состоит из трёх каналов. При смешении основных цветов (основными цветами считаются красный, зелёный и синий), например синего (B) и красного (R), мы получаем пурпурный (M, magenta), при смешении зелёного (G) и красного (R) — жёлтый (Y, yellow), при смешении зелёного (G) и синего (B) — циановый (C, cyan). При смешении всех трёх цветовых компонентов мы получаем белый цвет (W).

В телевизорах и мониторах применяются три электронные пушки (светодиода, светофильтра) для красного, зелёного и синего каналов.

- **СМЮК** — (4 294 967 296 цветов) — 4 байта.

Cyan, Magenta, Yellow, black — *субтрактивная* схема формирования цвета, используемая прежде всего в полиграфии для стандартной триадной печати. Схема **СМЮК**, как правило, обладает сравнительно небольшим цветовым охватом.

«Субтрактивный» означает «вычитаемый» — из белого вычитаются первичные цвета.

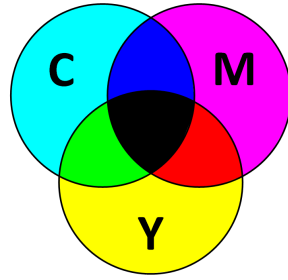


Рис. 1.7. Схема субтрактивного синтеза в СМΥК

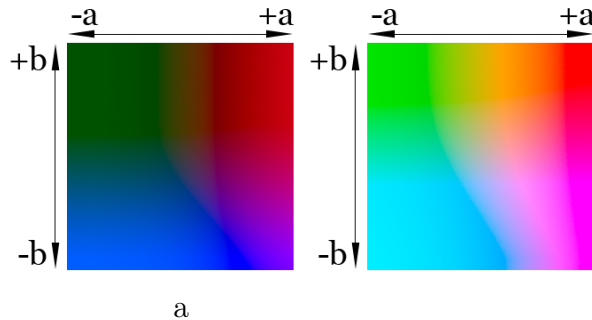


Рис. 1.8. Плоскость ab , соответствующая $L = 25\%$ (а) и $L = 75\%$ (б)

- **CIE Lab**.

В цветовом пространстве **CIE Lab** значение светлоты отделено от значения хроматической составляющей цвета (тон, насыщенность). Светлота задана координатой L (изменяется от 0 до 100, от самого тёмного до самого светлого), хроматическая составляющая — двумя полярными координатами a и b . Первая обозначает положение цвета в диапазоне от зелёного до пурпурного, вторая — от синего до жёлтого.

В отличие от цветовых пространств **RGB** или **СМΥК**, которые являются, по сути, набором аппаратных данных для воспроизведения цвета на бумаге или на экране монитора (цвет может зависеть от типа печатной машины, марки красок, влажности воздуха в цеху или производителя монитора и его настроек), **CIE Lab** однозначно определяет цвет.

Поэтому **CIE Lab** нашла широкое применение в программном обеспечении для обработки изображений в качестве промежуточного цветового пространства, через которое происходит конвертирование данных между другими цветовыми пространствами (например из **RGB** сканера в **СМΥК** печатного процесса). При этом особые свойства **CIE Lab** сделали редактирование в этом пространстве мощным инструментом цветокоррекции.

Благодаря характеру определения цвета в **CIE Lab** появляется возможность отдельно воздействовать на яркость, контраст изображения и на его цвет. Во многих случаях это позволяет ускорить обработку изображений, например, при допечатной подготовке.

Lab предоставляет возможность избирательного воздействия на отдельные цвета в изображении, усиления цветового контраста, незаменимыми являются и возможности, которые это цветовое пространство предоставляет для борьбы с шумом на цифровых фотографиях.

Для любой палитры количество всевозможных цветов $N_{\text{цв.}} = 2^b$, где b — число бит (**глубина цвета**), необходимых для кодирования цвета.

Размер файла тесно связан с **размером холста** (в пикселях по вертикали и горизонтали) и с **глубиной цвета**.

Размер растрового графического файла (без сжатия и заголовков): **высота × ширина × глубина**

Пример .1 (Размер растрового файла). Рассмотрим растровый файл размером 10×10 и чёрно-белым изображением буквы 'К'.

```

0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 1 0 0
0 0 0 1 0 0 1 0 0 0
0 0 0 1 0 1 0 0 0 0
0 0 0 1 1 0 0 0 0 0
0 0 0 1 0 1 0 0 0 0
0 0 0 1 0 0 1 0 0 0
0 0 0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

```

Растр с чёрно-белым изображением буквы 'К'

Для кодирования изображения в растровой форме на таком экране требуется 100 бит (1 бит на **пиксель**).

Представим этот код в виде битовой матрицы, в которой строки и столбцы соответствуют строкам и столбцам растровой сетки. Пусть 1 обозначает закрашенный **пиксель**, а 0 — не закрашенный.

Тот же рисунок в «серой шкале» займёт $100 \times 8 = 800$ бит = 100 Б, в **RGB**-палитре — 300 Б, в **СМУК**-палитре — 400 Б.

3.3. Интенсивность тона

Интенсивность тона или светлота (lightness) — это видимая степень заметности цветового тона в данном хроматическом цвете.

В компьютерной графике **интенсивность тона** имеет $N = 256$ градаций. Больше число не воспринимается. Для этого ячейка **растра** должна быть 16×16 точек. Вообще:

$$N = \left(\frac{\text{dpi}}{\text{lpi}} \right)^2 + 1 \quad \text{или} \quad \text{lpi} = \frac{\text{dpi}}{\sqrt{N - 1}}.$$

Абсолютно чёрный цвет (для серого цвета) соответствует 100% заполнению цветом ячейки **растра**.

Для промежуточных значений используется разный способ заполнения ячейки:

амплитудная модуляция — заполнение от центра (радиусом, соответствующим **интенсивности**);

частотная модуляция — периодическое заполнение (с частотой, соответствующей **интенсивности**);

стохастическое растрирование (квазислучайное заполнение) — хаотичное заполнение (со средней плотностью, соответствующей **интенсивности**).

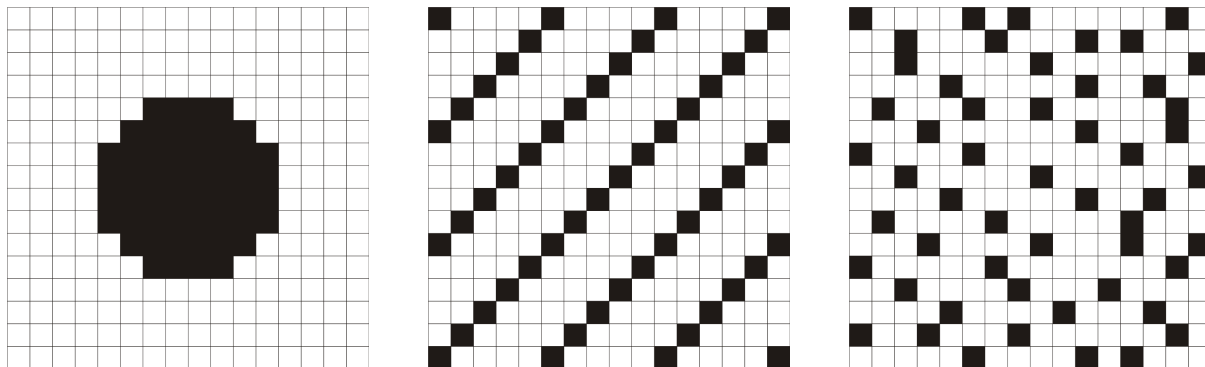


Рис. 1.9. Три способа заполнения ячейки **растра**: амплитудная модуляция, частотная модуляция и квазислучайное заполнение

При печати полноцветных изображений каждый последующий растр поворачивается на определённый угол:

С голубой — 105° ;

М пурпурный — 75° ;

У жёлтый — 90° ;

К чёрный — 45° .

При этом ячейка **растра** становится косоугольной, и для воспроизведения 256 градаций на устройстве с линиатурой 150 lpi уже недостаточно разрешения $16 \times 150 = 2400$ dpi. Для профессиональных фотоэкспонирующих устройств принято минимальное разрешение 2540 dpi (коэффициент поправки $\sim 1,06$).

3.4. Динамический диапазон

Динамический диапазон в компьютерной графике — отношение между максимальной и минимальной измеримой интенсивностью света.

Качество воспроизведения тоновых изображений оценивается **динамическим диапазоном** D :

$$D = \lg \frac{1}{\rho}, \quad \rho = \frac{J_\rho}{J_0}; \quad D = \lg \frac{1}{\sigma}, \quad \sigma = \frac{J_\sigma}{J_0}.$$

Здесь J_0 — максимальный (в идеале — падающий) отражённый световой поток, J_ρ — минимальный отражённый световой поток, ρ — коэффициент отражения, J_σ — минимальный прошедший световой поток, σ — коэффициент пропускания.

3.5. Гамма-коррекция

Гамма-коррекция — коррекция функции яркости в зависимости от характеристик устройства вывода.

Повышение показателя **гамма-коррекции** позволяет повысить контрастность, разборчивость тёмных участков изображения, не делая при этом чрезмерно контрастными или яркими светлые детали снимка.

Информация о яркости в аналоговом виде в телевидении, а также в цифровом виде, в большинстве распространённых графических форматов хранится в нелинейной шкале. Яркость пикселя I (или яркости составляющих цвета, красной, зелёной и синей по отдельности) на экране монитора можно считать

$$I \sim V^\gamma,$$

где V — численное значение интенсивности цвета, а γ — показатель **гамма-коррекции**.

Примером может служить **гамма-коррекция** изображения на электронно-лучевых трубках (ЭЛТ). Значение $\gamma = 1$ соответствует «идеальному» монитору, который имеет линейную зависимость отображения от белого к чёрному. Но таких мониторов не бывает — зависимость, в особенности для ЭЛТ, нелинейна. Большее значение γ означает более высокую нелинейность этой зависимости. Стандартное значение γ для стандарта видеоизображений NTSC — 2,2. Для дисплеев компьютера значение γ обычно находится в пределах от 1,5 до 2,0.

3.6. Альфа-композиция

Альфа-композиция обозначает процесс комбинирования изображения с фоном с целью создания эффекта частичной прозрачности.

Этот метод часто применяется для многопроходной обработки изображения по частям с последующей комбинацией этих частей в единое двумерное результирующее изображение.

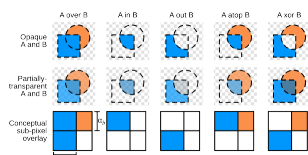


Рис. 1.10. Пример работы альфа-композиционных операторов **over**, **in**, **out**, **atop** и **xor**

Альфа-канал (**маска-канал**) позволяет объединить переходную прозрачность с изображением. Формат GIF поддерживает простую бинарную прозрачность (когда любой **пиксель** может быть либо полностью прозрачным, либо абсолютно непрозрачным). Формат *PNG* позволяет использовать 254 или 65534 уровня частичной прозрачности.

Все три типа *PNG* изображений («TrueColor», «GrayScale» и индексированная палитра) могут содержать **альфа-информацию**, хотя обычно она применяется лишь с «TrueColor» изображениями. Вместо того чтобы сохранять три байта для каждого пикселя (красный, зелёный и синий, **RGB**), сохраняются четыре: красный, зелёный, синий и **альфа**, таким образом получается палитра **RGBA**.

Такая переходная прозрачность позволяет создавать «спецэффекты», хорошо выглядящие на любом фоне. Например, эффекта фотовиньетки для портрета можно добиться путём установки полностью непрозрачной центральной области (для лица и плеч), прозрачной остальной обстановки и созданием плавного перехода между двумя этими различными областями. Соответственно, портрет будет плавно освещаться на белом фоне и затемняться на чёрном. Ещё один спецэффект с прозрачностью — это отбрасывание тени.

Прозрачность наиболее важна для маленьких изображений, обычно используемых на веб-страницах, вроде цветных (круглых) маркеров или причудливого текста. **Альфа-композиция** позволяет использовать **сглаживание (anti-aliasing)**, создавая иллюзию гладких кривых на сетке прямоугольных пикселей, плавно изменяя их цвета, что позволяет добиться округлых изображений, хорошо отображаемых как на белом, так и на любом другом фоне. Таким образом одно и то же изображение может быть многократно использовано в нескольких местах без «призрачного» эффекта, свойственного GIF-изображениям.

Windows XP поддерживает 32-битные значки (иконки) — 24-бита цвета **RGB** и 8-битный альфа канал. Это позволяет отображать значки со сглаженными (размытыми) краями и тенью, которые сочетаются с любым фоном.

4. Фрактальная графика

Фрактал — объект, отдельные элементы которого наследуют свойства родительских структур.

Поскольку более детальное описание элементов меньшего масштаба происходит по простому алгоритму, описать такой объект можно всего лишь несколькими математическими уравнениями.

Фракталы позволяют описывать целые классы изображений, для детального описания которых требуется относительно мало памяти. С другой стороны, к изображениям вне этих классов **фракталы** применимы слабо.

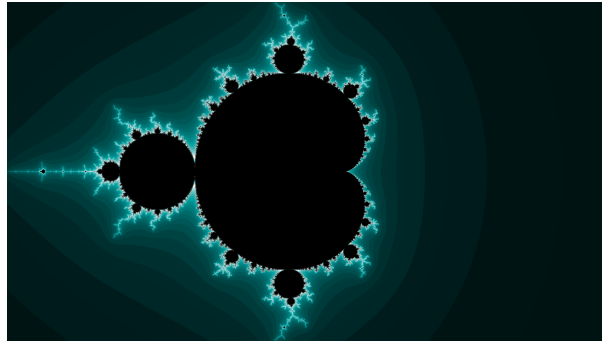
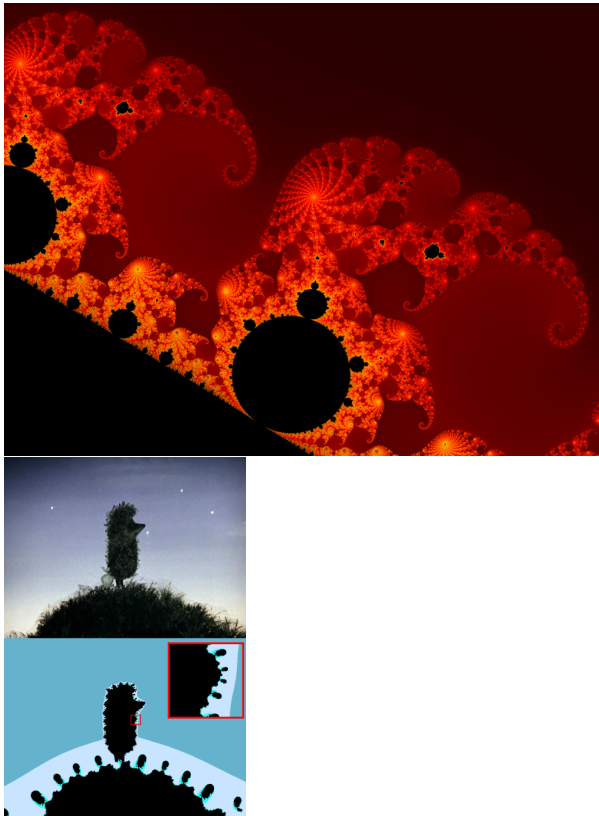


Рис. 1.11. Фрактальный рисунок



5. Форматы файлов

5.1. Векторные файлы

WMF (Windows MetaFile, .wmf) — формат MS Windows, цветовая палитра — 256 цветов, используется в галереях офисных и пр. пакетов.

EMF (Microsoft Enhanced MetaFile, .emf) — формат MS Windows, цветовая палитра — **RGB**, поддерживается далеко не всеми программами.

SVG (Scalable Vector Graphics, .svg) — формат, разработанный для внедре-

ния векторной графики в веб-документы, записывается в виде структурированного (XML) текста.

PS (*PostScript*, *.ps*) — платформенно независимый переносимый формат описания страниц фирмы Adobe, используется для описания многостраничных документов. Язык программирования высокого уровня со стековой организацией. Поддерживаются все линейные преобразования. Есть возможность создавать библиотеки цветов, шрифтов, форм, изображений, полутонов и узоров. Для сжатия используются алгоритмы JPEG и LZW.

EPS (*Encapsulated PostScript*, *.eps*) — платформенно независимый переносимый формат описания любых графических изображений в соответствии с соглашениями по структурированию документов в формате PostScript.

PDF (*Portable Document Format*, *.pdf*) — платформенно независимый переносимый формат описания документов фирмы Adobe, имеет два алгоритма сжатия: ZIP (без потерь) и JPEG (с потерями).

5.2. Алгоритмы сжатия

Сжатие без потерь (lossless data compression) — метод сжатия информации, при использовании которого закодированная информация может быть восстановлена с точностью до бита.

При этом оригинальные данные полностью восстанавливаются из сжатого состояния. Этот тип сжатия принципиально отличается от **сжатия данных с потерями**. Для каждого из типов цифровой информации, как правило, существуют свои оптимальные алгоритмы **сжатия без потерь**.

Сжатие данных без потерь используется во многих приложениях. Например, оно используется в популярном файловом формате ZIP и UNIX-утилите Gzip. Оно также используется как компонент в сжатии с потерями.

Сжатие без потерь используется, когда важна идентичность сжатых данных оригиналу. Обычный пример — исполняемые файлы и исходный код. Некоторые графические файловые форматы (*PNG*, *GIF* и др.) используют только **сжатие без потерь**, тогда как другие (*PS*, *PDF*, *TIFF*, *JPEG 2000*, *MNG* и др.) могут использовать *сжатие* как с потерями, так и без.

Сжатие с потерями — это метод сжатия данных (*data compression*), когда распакованный файл отличается от оригинального, но «достаточно близок» для того, чтобы быть полезным каким-то образом.

Этот тип сжатия часто используется для сжатия звука или изображений, а также в **Интернете**, особенно в потоковой передаче данных и телефонии. Эти методы часто называются **кодеками**.

Кодек (от кодер-декодер) — программный модуль, реализующий упаковку и распаковку звука или видео.

Существуют две основные схемы сжатия с потерями:

1. В **предсказывающих кодеках** предыдущие и/или последующие данные используются для того, чтобы предсказать текущий фрейм. Ошибка между предсказанными данными и реальными вместе с добавочной информацией, необходимой для производства предсказания, затем *квантизуется и кодируется*.
2. В **трансформирующих кодеках** берутся фреймы (изображений или звука), разрезаются на небольшие сегменты, трансформируются в новое базисное пространство и производится *квантизация*. Результат затем *сжимается энтропийными методами*.

В некоторых системах эти две техники комбинируются путём использования **трансформирующих кодеков** для сжатия ошибочных сигналов, сгенерированных на стадии предсказания.

Преимущество методов **сжатия с потерями** над методами **сжатия без потерь** состоит в том, что первые существенно превосходят по степени сжатия, продолжая удовлетворять поставленным требованиям.

Распакованный файл может очень сильно отличаться от оригинала на уровне сравнения «бит в бит», но практически неотличим для человеческого уха или глаза в большинстве практических применений.

Много методов основано на особенностях строения органов чувств человека. Психоакустическая модель определяет то, как сильно звук может быть сжат без ухудшения воспринимаемого качества звука.

*Дефекты, причинённые сжатием с потерями, которые заметны для человеческого глаза или уха, называются **артефакты сжатия**.*

Звуковые данные, прошедшие сжатие с потерями, не принимаются судами как вещественные доказательства (и даже не берутся во внимание) по причине того, что информация, прошедшая сжатие, приобретает артефакты сжатия и теряет естественные шумы среды, из которой производилась запись, в связи с чем невозможно установить, подлинная ли запись или синтезированная. Поэтому важные записи рекомендуется производить в форматах импульсно-кодовой модуляции (**ИКМ**, или Pulse Code Modulation, **PCM**) (см. стр. ??) или использовать плёночный диктофон.

Фотографии, записанные в формате *JPEG*, могут быть приняты судом (несмотря на то, что данные прошли сжатие с потерями). Но при этом должен быть предоставлен фотоаппарат, которым они сделаны, или соответствующая **фототаблица цветопередачи**.

5.3. Растровые файлы

BMP (Windows Bitmap, *.bmp*) — формат Microsoft Windows.

PCX (.pcx) — формат Z-Soft, имеет алгоритм **сжатия без потерь**, оптимизированный для **BW**-файлов.

TIFF (Tagged Image File Format, .tif, .tiff) — наилучший формат *хранения* растровых изображений, поддерживает различные **цветовые схемы**, алгоритм **сжатия без потерь** LZW и алгоритм **сжатия с потерями** JPEG. Поддерживается почти всеми издательскими и графическими пакетами.

RAW (.raw) — простой формат (сырые данные) растровых изображений **глубиной цвета** 256, в котором каждый **пиксель** представляется одним байтом (или символом).

В современных цифровых устройствах это формат цифровых файлов изображения, содержащий необработанные данные об электрических сигналах с фотоматрицы цифрового фотоаппарата, цифровой кинокамеры, а также сканеров неподвижных изображений или киноплёнки.

В таких файлах содержится информация, полученная непосредственно с АЦП, не имеющая какой-либо общепринятой спецификации.

GIF(87) (Graphics Interchange Format, .gif) — *выходной* формат растровых изображений (рисованного типа) для электронных публикаций, поддерживается почти всеми издательскими и графическими пакетами, сжатие достигается за счёт индексации цветов (до 256).

GIF(89a) (Graphics Interchange Format, .gif) — появилась возможность *чересстрочной загрузки*, задания *прозрачного цвета* и *покадровой анимации*.

PhotoCD (.pcd) — формат **Kodak**, имеет 5 фиксированных уровней разрешения: *Base* (512 × 768), *Base/4*, *Base × 4*, *Base/16*, *Base × 16*; имеет *алгоритм сжатия с потерями*.

JPEG (Joint Photographic Experts Group, .jpeg, .jpg, .jpe, .jfif) — *выходной* формат растровых изображений (*фотографического типа*) для электронных публикаций, поддерживается почти всеми издательскими и графическими пакетами, имеет *мощный регулируемый алгоритм сжатия с потерями*, возможность *чересстрочной загрузки*.

Поддерживается сжатие цветных (24 бит) и серых изображений. При сохранении можно указать степень качества (степень сжатия), которую обычно задают в некоторых условных единицах (например, от 1 до 100 или от 1 до 10). Большее число соответствует лучшему качеству, но при этом увеличивается размер файла. Чаще всего разница в качестве между 90% и 100% на глаз уже практически не воспринимается.

При сжатии изображение переводится в цветовую систему YCbCr (YUV) (подробнее см. на стр. ??, ??). Далее каналы изображения Cb и Cr, отвечающие за цвет, уменьшаются в 2 раза (по линейному масштабу) — формат 2:1:1. Уже на этом этапе необходимо хранить только четверть информации о цвете изображения.

Реже используется уменьшение цветовой информации в 4 раза (4:1:1) или сохранение размеров цветowych каналов как есть (1:1:1). Количество программ, которые поддерживают сохранение в таком виде, относительно невелико. Далее цветowe каналы изображения, включая чёрно-белый канал Y, разбиваются на блоки 8×8 пикселей. Каждый блок подвергается *дискретно-косинусному преобразованию*.

Полученные коэффициенты подвергаются квантованию и упаковываются с помощью **кодов Хаффмана**. Идея алгоритма состоит в следующем: зная вероятности символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством префиксности (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать.

Матрица, используемая для квантования коэффициентов, хранится вместе с изображением.

Это приводит к огрублению мелких деталей на изображении. Чем выше степень сжатия, тем более сильному квантованию подвергаются все коэффициенты.

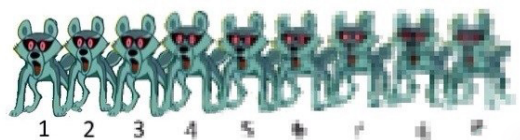


Рис. 1.12. Изображение в формате *JPEG* с увеличением степени сжатия слева направо

Progressive JPEG — способ записи сжатого изображения, при котором старшие (низкочастотные) коэффициенты находятся в начале файла.

В случае *progressive JPEG* сжатые данные записываются в выходной поток в виде набора сканов, каждый из которых описывает изображение полностью со всё большей степенью детализации. Это позволяет получить уменьшенное изображение при загрузке лишь небольшой части файла и повышать детализацию изображения по мере загрузки оставшейся части. **Progressive JPEG** получил широкое распространение в Интернете.

Демонстрация различной степени сжатия представлена на рис. 1.13.

В целом алгоритм основан на **дискретном косинусоидальном преобразовании** (ДКП), которое является разновидностью дискретного преобразования Фурье, применяемом к матрице изображения для получения некоторой новой матрицы коэффициентов. Для получения исходного изображения применяется обратное преобразование. ДКП раскладывает изображение по амплитудам некоторых частот. Таким образом, *при преобразовании мы получаем матрицу, в которой многие коэффициенты либо близки, либо равны нулю*.

Кроме того, *благодаря несовершенству человеческого зрения можно аппроксимировать коэффициенты более грубо без заметной потери качества изображения*.

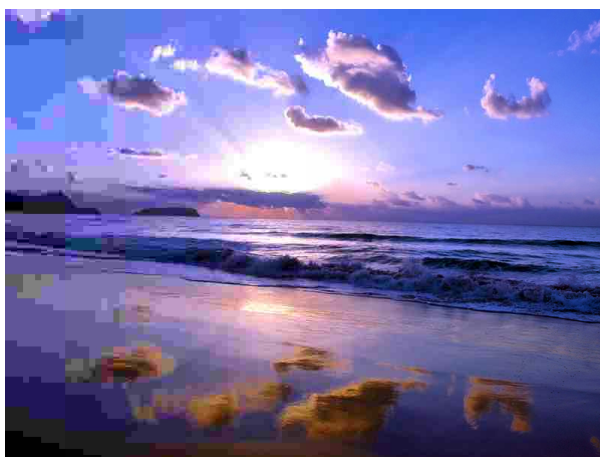


Рис. 1.13. Фотография заката в формате *JPEG* с уменьшением степени сжатия слева направо

Для этого используется квантование коэффициентов. В самом простом случае — это арифметический побитовый сдвиг вправо. При этом преобразовании теряется часть информации, но могут достигаться большие коэффициенты сжатия.

Процесс сжатия по схеме *JPEG* включает ряд этапов:

- преобразование изображения в оптимальное цветовое пространство;
- субдискретизация компонентов цветности усреднением групп пикселей;
- применение дискретных косинусных преобразований для уменьшения избыточности данных изображения;
- квантование каждого блока коэффициентов ДКП с применением весовых функций, оптимизированных с учётом визуального восприятия человеком;
- кодирование результирующих коэффициентов (данных изображения) с применением алгоритма группового кодирования и алгоритма Хаффмана для удаления избыточности информации.

JPEG 2000 (.j2k, .j2k, .jpf, .jpx, .jpm, .mj2, .jpg2 или .mjp2) — графический формат, который вместо дискретного косинусного преобразования, характерного для *JPEG*, использует технологию **вейвлет-преобразования**, основывающуюся на представлении сигнала в виде суперпозиции некоторых базовых функций — волновых пакетов.

Изображения *JPEG 2000* по сравнению с *JPEG* более гладкие и чёткие, а размер файла при одинаковом качестве уменьшается ещё на 30%. *JPEG 2000* полностью свободен от главного недостатка своего предшественника: благодаря использованию вейвлетов, изображения в этом формате не содержат знаменитой «решётки» из блоков по 8 пикселей. Новый формат также, как и *JPEG*, поддерживает так называемое

«прогрессивное сжатие», позволяющее по мере загрузки видеть сначала размытое, но затем всё более чёткое изображение.

JPEG 2000 во многом сходен с форматом сжатия изображений ICER, который используется NASA. Кодек изображений ICER был разработан для сжатия изображений на устройствах, работающих в открытом космосе.

Пока этот формат мало распространён и поддерживается не всеми современными браузерами. Среди поддерживающих JPEG 2000 браузеров — Konqueror, Safari и Mozilla Firefox (через Quicktime).

JPEG 2000 не является свободным от патентованных алгоритмов сжатия, но усилиями комитета *JPEG* достигнуто согласие, что в составе этого формата они могут использоваться бесплатно.

Всегда одним из самых больших преимуществ стандартов, выпущенных комитетом *JPEG*, было то, что они могут быть реализованы в базовой конфигурации без каких-либо лицензионных выплат. Новый стандарт *JPEG 2000* был подготовлен с учётом этой возможности, согласие было достигнуто между 20 большими организациями-держателями большинства патентов в области сжатия.

Разумеется, неопределённые и скрытые патенты могут всё ещё представлять опасность. Тем не менее *JPEG 2000* стоит рассматривать как более защищённый от притязаний формат, чем *JPEG* или *MP3*, для которых подобная работа велась на гораздо более низком уровне.

Однако, не взирая на свободу лицензирования патентов, *JPEG 2000* всё равно не может соответствовать *Debian Free Software Guidelines* (тест на свободу программного обеспечения). Это затрудняет адаптацию *JPEG 2000* к требованиям WWW, так исключает свободные веб-браузеры (особенно браузеры, основанные на Gecko) и популярные веб-приложения LAMP¹.

Артефакты, возникающие при сжатии алгоритмом *JPEG 2000*, отличаются от артефактов, возникающих при сжатии алгоритмом *JPEG* — присутствуют незначительные искажения на изображениях при высокой степени компрессии (см. рис. 1.14).

Часто фотографическое изображение может быть сжато в отношении 1/20 к оригинальному размеру без появления значительных искажений. Изображение справа демонстрирует различные искажения *JPEG 2000* при различных степенях сжатия (верхнее изображение — это оригинал без сжатия).

Основные области применения этого стандарта:

- цифровой кинематограф;
- мультимедийные устройства (цифровые камеры, КПК, смартфоны, цифровые факсы, принтеры, сканеры);
- клиент/серверные взаимодействия (Интернет, базы данных изображений, потоковое видео, видео-серверы);
- военное (HD-спутниковые изображения, обнаружение движения, распределённые сети и хранилища);

¹LAMP — акроним, обозначающий набор (комплекс) серверного программного обеспечения, широко используемый во Всемирной паутине. LAMP назван по первым буквам входящих в его состав компонентов: Linux, Apache, MySQL, PHP



Рис. 1.14. Артефакты компрессии *JPEG 2000* (числа показывают степень сжатия)

- медицинские изображения;
- хранение фотографии владельца в биометрических паспортах (на момент 08.03.2013 для сжатия используется JasPer 1.6);
- сенсорные устройства, цифровые устройства/архивы (Библиотека Конгресса США использует его как один из форматов для хранения оцифрованных версий географических карт).

Основные преимущества *JPEG 2000* по сравнению с *JPEG*:

- *Большая степень сжатия*: на высоких битрейтах, где артефакты незначительны, *JPEG 2000* имеет степень сжатия в среднем на 20% больше, чем *JPEG* (см. рис. 1.15). На низких битрейтах *JPEG 2000* также имеет преимущество над основными режимами *JPEG*. Большая степень сжатия достигается благодаря использованию дискретного **вейвлет-преобразования** и более сложного энтропийного кодирования.
- *Масштабируемость фрагментов изображений*: *JPEG 2000* обеспечивает бесповное сжатие разных компонентов изображения, с каждым компонентом хранится от 1 до 16 бит на сэмпл. Благодаря разбиению на блоки, можно хранить изображения разных разрешений в одном кодовом потоке.
- *Прогрессивное декодирование и масштабируемость отношения сигнал/шум*: *JPEG 2000* обеспечивает эффективную организацию кодового потока, которая позволяет просматривать файл с меньшей разрешающей способностью или с меньшим качеством.
- *Сжатие как с потерями, так и без потерь*. Сжатие без потерь обеспечивается путём использования обратимого (целочисленного) **вейвлет-преобразования**.
- *Произвольный доступ к кодовому потоку*, также иногда называемый доступом к «областям интереса» (Region of interest): кодовый поток *JPEG 2000* обеспечивает несколько механизмов для поддержки произвольного доступа, также поддерживается несколько степеней разбиения на части (области интереса).

- *Устойчивость к ошибкам: JPEG 2000* устойчив к битовым ошибкам, которые вносятся зашумлёнными каналами связи. Это достигается путём вставки маркеров ресинхронизации, кодирования данных в относительно небольшие независимые блоки и обеспечение механизмов для нахождения и локализации ошибок внутри каждого блока.
- *Возможность последовательной сборки: JPEG 2000* обеспечивает возможность последовательного декодирования и вывода изображения сверху вниз без необходимости буферизации всего изображения.
- *Гибкий формат файла:* обеспечивает хранение информации о цветовых пространствах, метаданных и информации для согласованного доступа в сетевых приложениях, взаимодействующих с помощью протокола *JPEG Part 9 JPIP*.

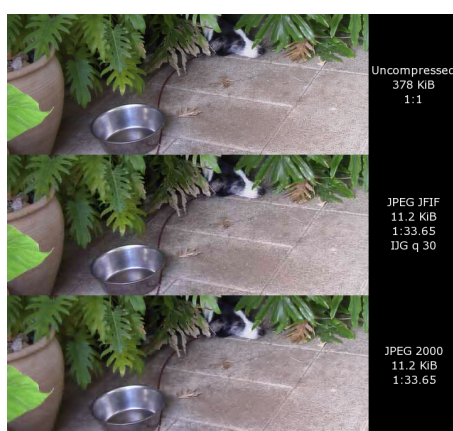


Рис. 1.15. Сравнение *JPEG* и *JPEG 2000*

JPEG XR (ранее назывался HD Photo и Windows Media Photo, *.jxr*, *.hdp*, *.wdp*) — формат кодирования и файловый формат для фотографий, разработанный и запатентованный Microsoft.

JPEG XR поддерживается операционной ОС Windows Vista, а также любой ОС с установленным .NET Framework 3.0.

В целом эффективность *JPEG XR* довольно высока: так, при 85%-ном качестве уже практически отсутствуют артефакты. При сравнении с *JPEG 2000* на 85% при меньшем размере файла *JPEG XR* даёт меньше артефактов.

Достоинства:

- Более эффективное сжатие чем *JPEG*, в разы быстрее открываются файлы тех же размеров что и *JPEG 2000*.
- Поддержка 16-bit позволяет хранить снимки с полным охватом цветов размером в разы меньше чем *TIFF*.
- Возможность сжатия без потерь (файл меньше аналогичного *PNG*).

- Возможность использовать альфа-канал.
- Можно использовать не только *R8G9B8*, но и другие форматы, такие как *R32G32B32*, *YUV*, *CMYK*.
- Стандартизирован *ISO* и *ECMA* для свободного использования.
- Корректное отображение метаданных (в *JPEG 2000* это реализовано только на бумаге).
- Поддерживают: XnView; Capture One; Photoshop (через плагин Windows и OS X).

JPEG XR запрещает создание реализаций, а также применение в ОС с лицензиями **copyleft**¹, например GPL и ОС GNU/Linux. Linux занимает очень крупную долю серверного и мобильного рынков.

JPEG XR поддерживается только браузерами IE7+ и выше. Открытые браузеры (например Mozilla Firefox) являются лицензионно несовместимыми с JPEG XR. Поддержка возможна только на основе плагина, но он будет работать исключительно в Windows.

Использование формата де факто принуждает пользователя продвигать ОС и технологии компании Microsoft.

PNG (Portable Network Graphics, *.png*) — *выходной* формат растровых изображений для электронных публикаций, поддерживается почти всеми издательскими и графическими пакетами. Поддерживаются палитры: серая 16 бит, индексированная 24 бит и полноцветная 48 бит; **Z-сжатие без потерь** (использует открытый, не запатентованный алгоритм сжатия DEFLATE); двумерная *чересстрочная развёртка*; *прозрачный цвет*; возможность **гамма-коррекции**; опциональная поддержка **альфа-канала**; возможность расширения формата пользовательскими блоками (на этом основан, в частности, формат *APNG*).

PNG был создан как для улучшения, так и для замены формата *GIF* графическим форматом, не требующим лицензии для использования, а также, в некоторой степени, для замены значительно более сложного формата *TIFF*.

Днём рождения *PNG* можно считать 4 января 1995 г., когда *Томас Бутелл* (*Thomas Boutell*) предложил в ряде конференций Usenet создать свободный формат, который был бы не хуже *GIF*. Через три недели после публикации идеи были разработаны четыре версии нового формата. Вначале он имел название *PBF* (Portable Bitmap Format), а нынешнее имя получил 23 января 1995 г. Уже в декабре того же года спецификация *PNG* версии 0.92 была рассмотрена консорциумом *W3C*, а с выходом 1 октября 1996 г. версии 1.0 *PNG* был рекомендован в качестве полноправного сетевого формата.

¹**Copyleft** позволяет использовать оригинальные (исходные) работы при создании новых (производных) работ без получения разрешения владельца авторского права.



Рис. 1.16. Визуализация изображения в формате *PNG* с 8-битным каналом прозрачности («шахматный» фон обычно используется в графических редакторах для обозначения «прозрачного» фона)



Рис. 1.17. Пример файла *APNG*

Хотя формат *JPEG 2000* поддерживает **сжатие без потерь**, он не предназначен для усовершенствования наилучшего формата **сжатия без потерь**.

Формат *PNG* более эффективен для изображений, содержащих одноцветные области (при небольшом количестве цветов — например, < 1000), и поддерживает специальные функциональные возможности, которых нет у *JPEG 2000* (см. рис. 1.16).

Считается, что в текущей реализации стандартов применение *PNG* более эффективно для сжатия диаграмм, а *JPEG 2000* — для сжатия фотографических изображений.

MNG (Multiple-image Network Graphics, *.mng*) — формат графических файлов для создания анимированных изображений, поддерживает все возможности алгоритмов сжатия *PNG* и *JPEG* (в том числе **альфа-канал** и **гамма-коррекцию**).

Поддерживается в браузере *Konqueror*, в браузере *Mozilla* — только с 2000 до 2003 г.

MNG близко связан с *PNG*. Когда в 1995 г. началась разработка формата *PNG*, разработчики решили не включать поддержку анимации, так как в то время эта особенность использовалась редко. Тем не менее, началась работа над *MNG* — версией *PNG* с поддержкой анимации. Первая версия спецификации *MNG* вышла 31 января 2001 г.

В настоящий момент MNG не поддерживается популярными ПО и браузерами. На его смену пришёл формат APNG, который намного проще MNG.

APNG (animated *PNG*, *.png*) — формат изображений, основанный на формате *PNG* с возможностью хранения анимации (аналогично *GIF*).

APNG — это расширенный формат *PNG*. Первый кадр *PNG* анимации хранится как обыкновенный поток *PNG*. Декодеры, не поддерживающие *APNG*, просто отобразят этот кадр. Все кадры, кроме первого, хранятся в дополнительных блоках *APNG*, который хранит информацию о количестве кадров и повторений анимации.

Чтобы уменьшить размер, *APNG* использует промежуточный буфер (спецификация называет его кадровым буфером). Каждый кадр имеет свой режим работы с

кадровым буфером:

None — сохранять кадр в кадровый буфер;

Background — очищать кадровый буфер;

Previous — не сохранять кадр в кадровый буфер.

Спецификация *APNG* была разработана *Стюартом Парментером (Stuart Parmenter)* и *Владимиром Вукичевичем (Vladimir Vukićević)* из *Mozilla Corporation* (*Mozilla Foundation*) для хранения элементов интерфейса, таких как анимация загрузки. *Mozilla* ранее отказалась от *MNG* (более мощного формата, поддерживающего все возможности *APNG*) из-за немалого размера *MNG*-библиотеки; декодер *APNG*, построенный прямо на библиотеке *PNG*, был намного меньше.

APNG был плохо встречен людьми, сопровождавшими спецификации *PNG* и *MNG*, они подчёркивали, что «*PNG* — это формат для неподвижных изображений». *APNG* хранит все кадры, кроме первого, в дополнительных блоках *PNG*-файла, и работающие с *PNG* программы будут игнорировать их.

В числе возражений — невозможно договориться с сервером о том, что выдавать: *PNG* или *APNG*, сложно отличить один от другого, а старая программа даже не предупредит о дополнительных кадрах. Таким образом, в *Mozilla* повторили ту же ошибку, которую совершили разработчики *GIF* 15 лет назад.

Глен Рэндерс-Пёрсон (Glenn Randers-Pehrson) предложил дать *APNG* новый MIME-тип (наподобие `video/png`), но *Mozilla* отказалась от этих предложений в пользу полной обратной совместимости.

20 апреля 2007 г. группа *PNG* официально отказалась признать *APNG*. Были и другие предложения простейшего анимационного формата, основанного на *PNG*, но не прошли и они.

В *Mozilla Firefox* *APNG* появился в версии 3 (23 марта 2007 г.). Но поскольку *libpng* поддерживается всё той же группой *PNG*, поддержки формата *APNG*, скорее всего, в ней никогда не будет.

Браузер *Iceweasel* в *Debian* долго не поддерживал *APNG*, но и он в 2011 г. перешёл с официальной библиотеки на модификацию *Mozilla*.

Роль *Mozilla* в продвижении формата *APNG* сравнивается с ролью *Netscape* в продвижении анимационного *GIF*.

APNG используется для слайдшоу во многих форматах цифрового радио.

Поддерживается ПО *KSquirrel*, *XnView*, *ImageJ*, *Imagine*, *TweakPNG*.

Не поддерживается ПО *Adobe*.

Поддержка браузерами:

- *Mozilla Firefox* (с 3.0) а также другое ПО, основанное на *Gecko* (например, *SeaMonkey*);
- *Opera* и *Opera Mobile* (с 9.5 до 15.0), *Opera Presto*;
- браузерами на основе *WebKit* (например, *Maxthon 3*, *Safari*, *Google Chrome* и *Chromium*) с 59.0.3042.0;
- *Iceweasel*.

Не поддерживается:

- Internet Explorer и др. на движке Microsoft Trident (например, Avant Browser, GreenBrowser);
- Konqueror;
- браузерами на основе Blink (например, Maxthon 3, Яндекс.Браузер, Vivaldi, Google Chrome и Chromium) (до версии 59);
- в связи с переходом на браузерный движок WebKit/Blink с Gecko поддержка *APNG* прекращена в браузерах Flock (с версии 3.0 и выше) и Eriphany (с версии 2.28 и выше), Opera Blink (с 15.0).
- Браузеры на основе старых версий WebKit (Konqueror, Rekonq, Midori).

WebP (web picture, *.webp*) — формат графических файлов, обеспечивающий возможность *сжатия* как **с потерями**, так и **без потерь** качества, предложенный компанией Google Inc. в 2010 г. Основан на алгоритме сжатия неподвижных изображений (ключевых кадров) из **видеокодека VP8**, использует **контейнер RIFF** (подробнее о сжатии см. на стр. 19).

Изображения в формате WebP, сжатые без потери качества, имеют размер на 28% меньший, чем *PNG*. Изображения в формате WebP с потерей качества имеют размер на 25–34% меньший, чем *JPEG* при равных значениях параметров. *WebP* также поддерживает прозрачность (**альфа-канал**).

Форматы *WebP* и *WebM* (web movie) продвигаются в качестве веб-стандартов компанией Google в рамках инициативы по уменьшению мирового интернет-трафика и улучшению качества интернет-технологий. *WebP* и *WebM* основаны на **кодеке VP8**, разработанном компанией On2 Technologies, впоследствии купленной компанией Google.

В настоящее время просмотр изображений в формате *WebP* поддерживается браузерами Google Chrome (начиная с 9 версии) и Opera (начиная с версии 11.10).

Android поддерживает чтение и запись *WebP* изображений, начиная с версии 4.0. С помощью специальной *JavaScript*-библиотеки возможно отображение в браузерах, поддерживающих видео в формате *WebM*, в частности в Firefox 4.0 и более новых.

Существует также порт библиотеки *libwebp* под названием *libwebpjs/libwebpas* на *JavaScript* и *ActionScript*, позволяющий использовать *WebP* во всех популярных браузерах (поддержка IE6+ осуществляется с помощью дополнительного модуля Adobe Flash).

MIFF (Magick Image File Format, *.miff*, *.mif*) — *платформенно независимый* формат растровых изображений, используемый программой отображения и анализа географических данных MapInfo. *MIFF* состоит из текстового заголовка файла и бинарной части с растром. Хранит визуализацию карты в текстовом формате, который может распознаваться сторонними приложениями. Используется в качестве формата обмена между приложениями геоинформационных систем. В файле содержатся графические данные (объекты), а также может содержаться описание таблицы данных, содержащей атрибутивную информацию, связанную с объектами.

PAM (NetPBM, .pam) — формат растровых изображений в виде 2-мерной целочисленной матрицы, параметры изображения определяются в заголовке файла; для BW палитры используются расширения .pbm, “Grayscale” — .pgm, **RGB** — .ppm, абстрактный формат для этих расширений — .pnm.

DjVu (déjà vu — «уже виденное», .djvu, .djv) — технология сжатия изображений с потерями, разработанная специально для хранения сканированных документов — книг, журналов, рукописей и прочее, где обилие формул, схем, рисунков и рукописных символов делает чрезвычайно трудоёмким их полноценное распознавание. Также является эффективным решением, если необходимо передать все нюансы оформления, например, исторических документов, где важное значение имеет не только содержание, но и цвет и фактура бумаги; дефекты пергамента: трещинки, следы от складывания; исправления, кляксы, отпечатки пальцев; следы, оставленные другими предметами.

DjVu стал основой для нескольких библиотек научных книг. Огромное количество книг в этом формате доступно в файлообменных сетях. Формат оптимизирован для передачи по сети таким образом, что страницу можно просматривать ещё до завершения скачивания. *DjVu*-файл может содержать текстовый (OCR) слой, что позволяет осуществлять полнотекстовый поиск по файлу. Кроме того, *DjVu*-файл может содержать встроенное интерактивное оглавление и активные ссылки, что позволяет реализовывать удобную навигацию в *DjVu*-книгах.

Для сжатия цветных изображений в *DjVu* применяется специальная технология, разделяющая исходное изображение на три слоя: передний план, задний план и чёрно-белую (однобитовую) маску. Маска сохраняется с разрешением исходного файла; именно она содержит изображение текста и прочие чёткие детали. Разрешение заднего плана, в котором остаются иллюстрации и текстура страницы, понижается для экономии места. Передний план содержит цветовую информацию о деталях, не попавших в задний план; его разрешение понижается ещё сильнее. Затем задний и передний планы сжимаются с помощью **вейвлет-преобразования**, а маска — алгоритмом JB2.

Особенностью алгоритма JB2 является то, что он ищет на странице повторяющиеся символы и сохраняет их изображение только один раз. В многостраничных документах каждые несколько подряд идущих страниц пользуются общим «словарём» изображений.

Для сжатия большинства книг можно обойтись только двумя цветами. В этом случае используется всего один слой, что позволяет достичь рекордной степени сжатия. В типичной книге с чёрно-белыми иллюстрациями, отсканированной с разрешением 600 dpi, средний размер страницы составляет около 15 Кб, т. е. приблизительно в 100 раз меньше, чем исходный файл.

DjVu используется сжатие данных с потерями. Для особо важных документов, возможно, будет разумнее использовать более «надёжные» форматы: *PNG*, *JPEG 2000*, *TIFF* и т. п. В общей сложности выигрыш объёма в этом случае составляет 4–10 раз.

В основе формата *DjVu* лежат несколько технологий, разработанных в АТ&Т. Это:

- алгоритм отделения текста от фона на отсканированном изображении;

- **вейвлетный** алгоритм сжатия фона IW44;
- алгоритм **сжатия** чёрно-белых изображений JB2;
- универсальный алгоритм **сжатия** ZP;
- алгоритм распаковки «по запросу»;
- алгоритм «маскировки» изображений.

6. Трёхмерная графика

Трёхмерная графика оперирует с объектами в трёхмерном пространстве. Обычно результаты **3D-графики** представляют собой плоскую картинку, проекцию.

Любое изображение на мониторе, в силу его плоскости, становится растровым, так как монитор — это матрица, он состоит из столбцов и строк. Трёхмерная графика существует лишь в нашем воображении, так как то, что мы видим на мониторе — это проекция трёхмерной фигуры, а уже создаём пространство мы сами. Таким образом, визуализация графики бывает только растровая и векторная, а способ визуализации — это только **растр** (набор пикселей), а от количества этих пикселей зависит способ задания изображения.

Трёхмерная компьютерная графика широко используется в кино, компьютерных играх.

В трёхмерной компьютерной графике все объекты обычно представляются как набор *поверхностей* или *частиц*.

*Минимальную поверхность называют **полигоном**.*

В качестве полигона обычно выбирают *треугольники*.

Всеми визуальными преобразованиями в **3D-графике** управляют *матрицы*. В компьютерной графике используется три вида *матриц*:

- поворота;
- сдвига;
- масштабирования.

Любой полигон можно представить в виде набора из координат его вершин. Так, у треугольника будет 3 вершины. Координаты каждой вершины представляют собой вектор (x, y, z) . Умножив вектор на соответствующую матрицу, мы получим новый вектор. Сделав такое преобразование со всеми вершинами полигона, получим новый полигон, а преобразовав все полигоны, получим новый объект, повернутый/сдвинутый/масштабированный относительно исходного.

6.1. Рендеринг

Статические и динамические изображения получают в проекции в результате рендеринга.

Рендеринг (*rendering* — перевод, изображение) — процесс получения изображения по модели с помощью компьютерной программы.

Здесь модель — это описание любых объектов или явлений на строго определённом языке или в виде структуры данных. Такое описание может содержать геометрические данные, положение точки наблюдателя, информацию об освещении, напряжённость физического поля, степени наличия какого-то вещества и пр.

Обычно в компьютерной графике (художественной и технической) под **рендерингом** понимают создание плоского изображения (картинки) по разработанной 3D-сцене. Синонимом в данном контексте является **визуализация**.

Существуют встроенные и отдельные программные продукты, выполняющие **рендеринг**. Обычно программные пакеты трёхмерного моделирования и анимации включают в себя также и функцию **рендеринга**.

В зависимости от цели различают **пре-рендеринг** как достаточно медленный процесс визуализации, применяющийся в основном при создании видео, и **рендеринг в реальном режиме** (времени), применяемый в компьютерных играх. Последний часто использует 3D-ускорители.

Компьютерная программа, производящая рендеринг, называется рендерером (*render*), *рендерером* (*renderer*) или *визуализатором*.

6.2. Методы визуализации

На текущий момент разработано множество алгоритмов визуализации. Существующее программное обеспечение может использовать несколько алгоритмов для получения конечного изображения.

Трассирование каждого луча света в сцене непрактично и занимает неприемлемо долгое время. Даже трассирование малого количества лучей, достаточного, чтобы получить изображение, занимает чрезмерно много времени, если не применяется аппроксимация (сэмплирование).

Вследствие этого было разработано четыре группы методов, более эффективных, чем моделирование всех лучей света, освещающих сцену.

1. *Растрезация* (*rasterization*) и *метод сканирования строк* (*scanline rendering*). Визуализация производится проецированием объектов сцены на экран без рассмотрения эффекта перспективы относительно наблюдателя.
2. *Ray casting*. Сцена рассматривается как наблюдаемая из определённой точки. Из точки наблюдения на объекты сцены направляются лучи, с помощью которых определяется цвет пикселя на двумерном экране. При этом лучи прекращают своё распространение (в отличие от метода обратного трассирования),

когда достигают любого объекта сцены либо её фона. Возможно используются какие-то очень простые техники добавления оптических эффектов. Эффект перспективы получается естественным образом в случае, когда бросаемые лучи запускаются под углом, зависящим от положения пикселя на экране и максимального угла обзора камеры.

3. *Глобальное освещение (global illumination, radiosity)*. Использует математику конечных элементов, чтобы симулировать диффузное распространение света от поверхностей и при этом достигать эффектов «мягкости» освещения.
4. *Трассировка лучей (ray tracing)*. Из точки наблюдения на объекты сцены направляются лучи, с помощью которых определяется цвет пикселя на двумерном экране. Но при этом луч не прекращает своё распространение, а разделяется на три компонента луча, каждый из которых вносит свой вклад в цвет пикселя на двумерном экране: отражённый, теневой и преломлённый. Количество таких разделений на компоненты определяет глубину трассирования и влияет на качество и фотореалистичность изображения. Благодаря своим концептуальным особенностям метод позволяет получить фотореалистичные изображения, но при этом он очень ресурсоёмкий, и процесс визуализации занимает значительные периоды времени.

6.3. Шейдеры

Шейдер (shader) — это программа для определения окончательных параметров объекта или изображения.

Она может включать в себя произвольной сложности описание поглощения и рассеяния света, наложения текстуры, отражение и преломление, затенение, смещение поверхности и эффекты пост-обработки.

Программируемые **шейдеры** обладают высокой эффективностью и гибкостью. Сложные с виду поверхности могут быть визуализированы при помощи простых геометрических форм. Например, **шейдеры** могут быть использованы для рисования поверхности из трёхмерной керамической плитки на абсолютно плоской поверхности.

В программных графических движках вся цепочка **рендеринга** — от определения видимых частей сцены до наложения текстуры — писалась разработчиком игры. В эту цепочку можно было включать собственные нестандартные видеоэффекты. Но с появлением видеоакселераторов разработчик оказался ограничен тем набором эффектов, который заложен в аппаратное обеспечение.

Вот два примера. Попробуйте нырнуть под воду в **Quake 2** на программном и на **OpenGL-рендеринге**. При всём качестве аппаратно ускоренной картинки, вода там — просто синий светофильтр, в то время как в программном есть эффект плеска воды. В **Counter-Strike** эффект ослепления от светошумовой гранаты на **аппаратном**

рендеринге — белая вспышка, на программном — белая вспышка и пикселизированный экран.

Для того чтобы составлять сложные видеоэффекты из атомарных операций, и были изобретены **шейдеры**. Предшественниками **шейдеров** были процедурная генерация текстур (широко применявшаяся в Unreal для создания анимированных текстур воды и огня) и мультитекстурирование (на нём был основан язык **шейдеров**, применявшийся в Quake 3). Но и эти механизмы не обеспечивают такой гибкости, как **шейдеры**.

В настоящее время **шейдеры** делятся на четыре типа:

- *вершинные*;
- *геометрические*;
- *параллаксные*;
- *фрагментные (пиксельные)*.

Вершинный шейдер оперирует данными, сопоставленными с вершинами многогранников. К таким данным, в частности, относятся координаты вершины в пространстве, текстурные координаты, тангенс-вектор, вектор бинормали, вектор нормали. **Вершинный шейдер** может быть использован для видового и перспективного преобразования вершин, генерации текстурных координат, расчёта освещения и т. д.

Геометрический шейдер, в отличие от вершинного, способен обработать не только одну вершину, но и целый примитив. Это может быть отрезок (две вершины) и треугольник (три вершины), а при наличии информации о смежных вершинах (adjacency) может быть обработано до шести вершин для треугольного примитива. Кроме того, **геометрический шейдер** способен генерировать примитивы «на лету», не задействуя при этом центральный процессор. Впервые данный **шейдер** начал использоваться на видеокартах nVidia серии 8. **Фрагментный шейдер** работает с фрагментами изображения. Под фрагментом изображения в данном случае понимается **пиксель**, которому поставлен в соответствие некоторый набор атрибутов, таких как цвет, **глубина**, текстурные координаты. **Фрагментный шейдер** используется на последней стадии графического конвейера для формирования фрагмента изображения.

Шейдерные языки обычно содержат специальные типы данных, такие как *цвет* и *нормаль*. Поскольку компьютерная графика имеет множество сфер приложения, для удовлетворения различных потребностей рынка было создано большое количество **шейдерных языков**.

Впервые использованные в системе RenderMan компании Pixar, **шейдеры** получили всё большее распространение со снижением цен на компьютеры. Основное преимущество от использования **шейдеров** — их гибкость, упрощающая и удешевляющая цикл разработки программы и при этом повышающая сложность и достоверность визуализируемых сцен.

Рассмотрим наиболее распространённые шейдерные языки.

Шейдерный язык RenderMan является фактическим стандартом для профессионального **рендеринга**. API RenderMan, разработанный *Робом Кукком (Rob Cook)*, используется во всех работах студии Pixar и не только. В 2004 г. этот пакет использовали в съёмках тридцати пяти из тридцати девяти фильмов, номинированных на «Оскар» в категории «Лучшие визуальные эффекты». *RenderMan* также является первым из реализованных **шейдерных языков**.

nVidia Gelato представляет собой оригинальную гибридную систему **рендеринга** изображений и анимации трёхмерных сцен и объектов, использующую для расчётов центральные процессоры и аппаратные возможности профессиональных видеокарт серии *Quadro FX*.

Шейдерный язык OpenGL носит название **GLSL** (The OpenGL Shading Language). **GLSL** основан на языке **ANSI C**. Большинство возможностей языка **ANSI C** сохранено, к ним добавлены векторные и матричные типы данных, часто применяющиеся при работе с трёхмерной графикой. В контексте **GLSL** шейдером называется независимо компилируемая единица, написанная на этом языке. Программой называется набор откомпилированных шейдеров, связанных вместе.

Низкоуровневый шейдерный язык DirectX (DirectX ASM) по синтаксису сходен с **Ассемблером**. Существует несколько версий, различающихся по набору команд, а также по требуемому оборудованию, есть разделение на **вершинные (vertex)** и **пиксельные (pixel)** шейдеры.

Высокоуровневый шейдерный язык DirectX HLSL (HLSL — High Level Shader Language) является надстройкой над **DirectX ASM**. По синтаксису сходен с **C**, позволяет использовать структуры, процедуры и функции.

Язык программирования Cg разработан nVidia совместно с Microsoft (такой же по сути язык от Microsoft — **HLSL**, включён в DirectX 9). **Cg** расшифровывается как «C for Graphics». Язык использует схожие с **C** типы (**int**, **float**), а также специальный 16-битный тип с плавающей запятой — **half**, обладает оптимизацией в виде упакованных массивов. Поддерживаются функции и структуры (см. рис. 1.18).

Несмотря на то, что язык разработан nVidia, он без проблем работает и с видеокартами ATI.

Следует учесть, что все **шейдерные программы** обладают своими особенностями, которые следует получить от разработчика.

Математическая модель

Передовое программное обеспечение обычно совмещает в себе несколько техник, чтобы получить достаточно качественное и фотореалистичное изображение за приемлемые затраты вычислительных ресурсов.

Реализация механизма **рендеринга** всегда основывается на физической модели. Производимые вычисления относятся к той или иной физической или абстрактной модели. Основные идеи просты для понимания, но сложны для применения.



Рис. 1.18. Изображение, отрендеренное в POV-Ray 3.6. Модель игровой кости создана в Cinema 4D, остальное — при помощи Rhinoceros 3D

Основное уравнение

Ключом к теоретическому обоснованию **моделей рендеринга** служит **уравнение рендеринга**. Оно является наиболее полным формальным описанием части **рендеринга**, не относящейся к восприятию конечного изображения. Все модели представляют собой какое-то приближённое решение этого уравнения.

$$L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + \int_{\Omega} f_r(x, \vec{\omega}', \vec{\omega}) L_i(x, \vec{\omega}') (\vec{\omega}' \cdot \vec{n}) d\vec{\omega}',$$

где L_o — количество светового излучения, исходящего из определённой точки в определённом направлении; L_e — собственное излучение; L_i — приходящее излучение; f_r — коэффициент отражения.

Иначе говоря, количество светового излучения, исходящего из определённой точки в определённом направлении, есть собственное излучение и отражённое излучение. Отражённое излучение есть сумма по всем направлениям приходящего излучения, умноженного на коэффициент отражения из данного угла.

Объединяя в одном уравнении приходящий свет с исходящим в одной точке, это уравнение составляет описание всего светового потока в заданной системе.

Рендереры

Ниже перечислены наиболее распространённые **рендереры**.

- 3Delight;
- AIR;
- ART;
- AQSIS;
- Angel;
- BMRT (Blue Moon Rendering Tools) (распространение прекращено);

- Brazil R/S;
- BusyRay;
- Entropy (продажи прекращены);
- finalRender;
- Fryrender;
- Gelato (разработка прекращена в связи с покупкой nVidia, mental ray);
- Holomatix Renditio (интерактивный raytracer);
- Indigo Renderer;
- mental ray;
- Kerkythea;
- LuxRender;
- Maxwell Render;
- Meridian;
- POV-Ray;
- Pixie;
- RenderDotC;
- RenderMan (PhotoRealistic RenderMan, Pixar's RenderMan);
- Sunflow;
- Turtle;
- V-Ray;
- YafRay;
- Octane Render;
- Arion Renderer.

Рендереры работающие в реальном времени

- VrayRT;
- FinalRender;
- iray;
- Shaderlight;
- Showcase;
- Rendition;
- Brazil IR.

Пакеты трёхмерного моделирования, имеющие собственные рендереры

- Autodesk 3ds Max (Scanline);
- Autodesk Maya (Software Hardware, Vector);
- Blender;
- NewTek LightWave 3D;
- Maxon Cinema 4D (Advanced Render);
- SketchUp;
- Daz3D Bryce;
- Luxology Modo;
- e-on Software Vue;
- SideFX Houdini;
- Terragen, Terragen 2.

7. Конвертеры файлов

7.1. NetPBM

NetPBM распространяется бесплатно. Автор — *Джеф Посканзер (Jef Poskanzer), Брайэн Хендерсон (Bryan Henderson)*.

Формат черно-белых изображений PBM был разработан Джефом Посканзером (Jef Poskanzer). Формат был достаточно простым, чтобы PBM-изображения могли пересылаться по электронной почте без порчи данных. В 1988 году Poskanzer выпустил Pbmplus — предшественника современного пакета Netpbm. К концу 1988 года Посканзер разработал форматы PGM (для полутоновых) PPM (для цветных) изображений, которые могли обрабатываться Pbmplus.

Последний релиз Pbmplus был выпущен 10 декабря 1991 года. Посканзер больше не развивал свой проект и в 1993 году на смену Pbmplus пришёл Netpbm. Поначалу это было не более, чем переименованный релиз Pbmplus, но он развивался вплоть до 1995, когда пакет вновь оказался заброшенным. В 1999 году развитие пакета Netpbm было подхвачено его нынешним меинтейнером, Брайэном Хендерсоном (Bryan Henderson).

Название Netpbm пошло от разработчиков, сотрудничавших при помощи сети Internet, что в то время было примечательно. (Аналогичные имена были даны операционной системе NetBSD и игре NetHack.)

Первый выпуск — 2000 (1988), последняя версия — 10.35.90 (26 сентября 2013).

Использует свой формат *PAM* для промежуточного хранения растра, понимает множество входных и выходных растровых форматов и *PS*.

Представляет из себя большой набор утилит, несколько динамических библиотек и скриптов на *shell* и *Perl*.

Пример .2 (NetPBM: GIF \mapsto EPS). Для конвертации GIF-файла в EPS надо использовать утилиты `giftopnm` и `pnmtops`:

```
giftopnm file.gif | pnmtops > file.eps
```

Полный синтаксис: `giftopnm [-alphaout=alpha-filename,-] [-verbose] [-comments] [-image=N,all] [GIFfile]`

`pnmtops [-scale=s] [-dpi=N[xN]] [-imagewidth=n] [-imageheight=n] [-width=N] [-height=N] [-equalpixels] [-turn|noturn] [-rle|runlength] [-flate] [-ascii85] [-nocenter] [-nosetpage] [-level=N] [-psfilter] [-noshowpage] [pnmfile]`

7.2. ImageMagick

ImageMagick — свободно распространяемая коллекция утилит для чтения, записи и редактирования файлов как растровых, так и векторных форматов (более 200 форматов!), от ImageMagick Studio, разработчики — *Джон Кристи* (*John Cristy*) и *Глен Рэндерс-Пёрсон* (*Glenn Randers-Pehrson*).

Первый выпуск — 1990, последняя версия — 7.0.7-24 (25 февраля 2018).

Для получения списка поддерживаемых форматов введите в терминале команду

```
convert -list format
```

ImageMagick может использоваться с языками Perl, C, C++, Python, Ruby, PHP, Node.js, Java, Pascal, Object-Pascal, в скриптах командной оболочки или самостоятельно.

Предыдущий пример:

Пример .3 (ImageMagick: GIF \mapsto EPS). `convert file.gif file.eps`

Пример .4 (Фрагмент пакетного файла для генерации файлов предпросмотра).

```
for %%f in (*.jpg) do convert -size 120x120 %%f  
    -resize 120x120 +profile "*" thumbnail/%%f
```

Полное описание всех возможностей редактора с большим количеством примеров можно найти на официальном сайте: imagemagick.org.

8. Деловая и научная графика

Деловая и научная графика присутствует во многих статистических, аналитических системах, а также средах проектирования и разработки. Например, *Deductor*, *R/RStudio*, *BPwin*, *CA ERwin Process Modeler* (ранее *ERwin*), *ER/Studio* (*Embarcadero*), *Ramus*, *BizAgi Process Modeler*, *IBM WebSphere*, *IBM Rational Data Architect*, *IBM Rational Software Architect*, ...

Существуют и отдельные графические инструменты, которые находят применение в различных областях.

- **MS Visio** — проприетарный редактор диаграмм и блок-схем для *Microsoft Windows*. В стандартный набор программ *MS Office* входит только средство для просмотра и печати диаграмм *Microsoft Visio Viewer*. Полнофункциональная версия *Microsoft Visio Professional* для создания и редактирования монограмм и диаграмм в пакеты *MS Office* не входит и распространяется отдельно. *VDX* является хорошо задокументированным *XML* «*DatadiagramML*» форматом. Начиная с версии *Visio 2013*, сохранение в формате *VDX* больше не поддерживается в пользу новых *VSDX* и *VSDM* файловых форматов. Формат *DatadiagramML* используется многими другими инструментами по управлению бизнес-процессами, такими как *Agilian*, *ARIS Express*, *Bonita Open Solution*, *ConceptDraw*, *OmniGraffle*, *IBM WebSphere*. *LibreOffice*, начиная с версии 4.0 поддерживает просмотр всего спектра *Visio* файлов (начиная с *Visio 1.0* и заканчивая *Visio 2013*, включая *VSDX*, *VSDM* и *VDX* файловые форматы).
- **Dia** — свободный кроссплатформенный редактор диаграмм, часть *GNOME Office*, но может быть установлен независимо. Он может быть использован для создания различных видов диаграмм: блок-схем алгоритмов программ, древовидных схем, статических структур *UML*, баз данных, диаграмм сущность-связь, радиоэлектронных элементов, потоковых диаграмм, сетевых диаграмм и других. *Dia* может расширяться новыми наборами объектов, которые описываются с помощью файлов в формате, основанном на *XML*, есть возможность импорта/экспорта библиотек объектов *MS Visio*. Возможности:
 - Поддержка диаграмм потоков, структурных диаграмм и т. д.
 - Экспорт в *Postscript*.
 - Загрузка и сохранение в формате *XML*.
 - Возможность описания новых объектов.
 - Установка свойств по умолчанию для добавляемых объектов.
 - Изменение цвета шрифта и заливки блоков.

Надстройки:

- *AutoDia* — автоматическое создание *UML*-схем из программного кода.
- *Dia2Code* — автоматическое преобразование *UML*-схем в программный код.

Dia позволяет экспортировать и сохранять диаграммы в различные форматы: *EPS*, *SVG*, *DXF* (Autocad's Drawing Interchange format), *CGM* (Computer Graphics Metafile), *WMF*, *PNG*, *JPEG*, *VDX*.

- **QtiPlot** — свободное программное обеспечение для анализа и визуализации научных данных. Версии для Windows и Mac OS X платные, демоверсии запрещают сохранение проекта и ограничивают время работы 10 минутами. QtiPlot по функциональности похожа на Origin и SigmaPlot, и используется для их замены в институтах, для проведения научно-исследовательских работ и подготовке их к публикации. QtiPlot может использоваться для создания 2D и 3D графиков и содержит большое количество функций для анализа данных, таких как аппроксимация кривых и т.п. При построении 3D-графиков рендеринг может производиться с использованием OpenGL (при помощи библиотеки Qwt3D). Последняя версия 0.9.8.9 (2 ноября 2011).
- **GNUplot**.

8.1. Редактор научной графики GNUplot

GNUplot — управляемая командами интерактивная программа составления графиков (в режиме командной строки).

GNUplot написан Томасом Вильямсом (*Thomas Williams*) и Колином Келли (*Colin Kelley*).

Эта программа распространяется свободно (“as is”), отличается компактностью и мобильностью. Она работает на различных платформах: UNIX/Linux, MacOS, Windows и др., а созданные в ней макрофайлы (обычно с расширением `.plt`) независимы от платформы.

GNUplot также используется в качестве системы вывода изображений в различных математических пакетах: GNU Octave, Maxima, Reduce и других.

Последняя версия 5.2.1 (октябрь 2017).

Синтаксис

Программа чувствительна к регистру, имена команд можно сокращать. В строке может быть любое количество команд, отделяемых ‘;’. Строки заключаются в двойные или одинарные кавычки.

Запуск:

`gnuplot` После этого можно писать команды.

Запуск в пакетном режиме: `gnuplot макрофайл`

Кроме того, GNUplot можно использовать в конвейере (вместе с другими командами и программами).

Выход:

`quit` или `exit`

Помощь

`help` или `?` — вывод содержания; `help команда` — вывод справки о *команде*;
`help тема` — вывод справки по указанной *теме*.

В описаниях команд необязательные аргументы указываются в квадратных скобках (`[...]`).

Редактирование командной строки

GNUplot поддерживает стиль редактирования EMACS, а в версиях для MS DOS и WINDOWS — клавиши управления курсором. Клавиша `Esc` очищает командную строку. GNUplot также поддерживает *историю команд*.

Графические устройства

GNUplot поддерживает все существующие графические (внешние) устройства. Посмотреть полный список доступных устройств можно с помощью команды `set terminal`.

`set terminal устройство [опции]` — установка в качестве выходного указанное графическое *устройство*.

`show terminal` — выводит установленное графическое устройство.

Макрофайлы

`save [functions|variables|set] 'файл'` — сохраняет в *файле* определённые пользователем функции, переменные, настройки.

Имя *файла* пишется с произвольным расширением.

Пример .5 (Сохранение файлов в GNUplot). `save 'work.gnu'`

```
save functions 'func.dat'  
save var 'var.dat'  
save set 'options.dat'
```

`load 'файл'` — считывает *файл*.

`cd 'директория'` — изменяет текущую *директорию*.

`pwd` — выводит текущую *директорию*.

Внутри файла возможно применение символов `\` — для продолжения строки (ставится в конце строки) и `#` — для комментирования строки.

Построение графиков

Двумерный график:

```
plot [диапазон] {функция |  
  'файл' [модификации]} [axes оси]  
  [title 'заголовок' | notitle]  
  [with стиль], ...
```

Трёхмерный график:

```
splot [диапазон] {функция |
      {'файл' [модификации]}} [axes оси]
      [title 'заголовок' | notitle]
      [with стиль], ...
```

Диапазон задаётся в виде интервала `[a:b]`, первый интервал относится к оси X , второй — к оси Y .

Функция записывается с использованием арифметических знаков и стандартных функций, кроме того, можно использовать операторы языка C (для возведения в степень используется оператор Фортрана `**`).

Функции могут быть параметрическими (t,u,v) .

Пример .6 (Функции в GNUplot). `plot sin(x)`

```
f(x)=sin(x)
plot f(x)
plot sin(x),cos(x)
set param
plot sin(t),cos(t+pi/2*3)
```

Данные для графиков могут быть записаны в файл. *Файл* состоит из данных, записанных в столбцы и разделённых пробелами. Данные делятся на блоки 2-мя пустыми строками. Одна пустая строка обозначает разрыв (при использовании линии). В файле данных также можно комментировать записи символом `#`.

Данные могут быть записаны в экспоненциальном формате, с использованием символов `'e'`, `'E'`, `'d'`, `'D'`, `'q'` или `'Q'`. Если записан только один столбец, эти данные принимаются за y , а соответствующие значения x считаются целыми, начиная с 0.

Имя файла может отсутствовать (`'`), тогда берётся ранее считанный файл. Если задать имя `'-'`, то данные можно вводить в командной строке, закончив ввод символом `'e'`.

В качестве модификаторов могут использоваться следующие параметры: `index`, `every`, `thru`, `using`, `smooth`.

Параметр *оси* используется, чтобы выбрать оси, для которых график должен масштабироваться; этот параметр может принимать одно из четырёх возможных значения:

x1y1 — естественный масштаб;

x2y2 — масштабирование по обоим осям;

x1y2 — масштабирование по оси Y ;

x2y1 — масштабирование по оси X .

Опция `title` задаёт *заголовок* для каждого набора данных, который записывается в легенде.

Параметр *стиль* задаёт стиль линии графика и может принимать одно из следующих значений: `lines`, `points`, `linespoints`, `impulses`, `dots`, `steps`, `fsteps`, `histeps`, `errorbars`, `xerrorbars`, `yerrorbars`, `xyerrorbars`, `boxes`, `boxerrorbars`, `boxxyerrorbars`, `financebars`, `candlesticks`, `vector`.

В качестве модификации стиля можно изменить тип, стиль, толщину линии, тип и размер точек.

Установки параметров

Для установки *параметров* используется команда

```
set параметр [опции].
```

 Опции для каждого параметра различны.

Часто используются такие параметры, как `title` — заголовок графика; `xlabel`, `ylabel`, `zlabel` — подписи по осям. Полный перечень изменяемых параметров можно посмотреть, используя справку.

Для вывода значений *параметров* используется команда `show параметр`. Для вывода значений всех переменных используется команда `show all`.

9. Полезные ссылки

[PDF to Word Convert](#) — он-лайн конвертер из PDF в формат MS Word. Для конвертации также доступны другие форматы файлов MS Office: Excel и PowerPoint.

[i.onthe.io/tools](#) (Online Image Tools) — набор графических инструментов для он-лайн редактирования изображений.

Прекращение поддержки [Picasa](#), [Picnik](#). [Google Фото](#) — приложение для обработки фотографий, с помощью которого пользователи могут редактировать загруженные на сайт фотографии в онлайн-режиме. Купив компанию Nik Software с редактором Snapseed, Google добавила мощные возможности редактирования фотографий в свой продукт. Соцсеть [Google+](#) предоставляет возможность хранить фотографии, делиться ими, создавать фотоальбомы. [Google+ Photos](#) является, пожалуй, одним из сильнейших игроков в сфере облачных фото хранилищ.

Фотохостинг — например, [Photobucket](#), [Imageshack](#). [Panoramio](#) — веб-сайт для размещения фотографий, позволяющий сохранять их географические координаты. Поддержка прекращена с 4 ноября 2016 г. Штаб-квартира [Panoramio](#) находится в Цюрихе, в офисном здании швейцарского подразделения Google. С конца 2012 года осуществляется перевод штаб-квартиры в головной офис Google в Маунтин-Вью с полной заменой европейских сотрудников сайта на американских. Серверы, хранящие фотографии пользователей, расположены в США.

[Instagram](#) — бесплатное приложение для обмена фотографиями и видеозаписями с элементами социальной сети, позволяющее снимать фотографии и видео, применять к ним фильтры, а также распространять их через свой сервис и ряд других социальных сетей. Instagram позволял делать фотографии квадратной формы — как камеры моментальной фотографии Polaroid, Kodak Instamatic и среднеформатные камеры 6×6 (большинство же мобильных фото-приложений использует соотношение сторон 3:2), но с 26 августа 2015 года Instagram ввела возможность добавлять фото и видео с ландшафтной и портретной ориентацией, без обрезания до квадратной формы. Приложение совместимо с мобильными устройствами на iOS (4.3 и выше) и на Android (2.2 и выше). Распространяется оно через App Store и Google Play соответственно. В апреле 2012 года Instagram был приобретён компанией Facebook.

[Flickr](#) — социальная сеть фотографов (более 120 000 000 участников, более 2 000 000 групп). Flickr является лучшим вариантом для профессиональных фотографов. Он сохраняет фото в нескольких разрешениях, даёт отличные настройки приватности, имеет публичный API, что позволяет ему интегрироваться с десятками сторонних сервисов. Flickr — сервис, предназначенный для хранения и дальнейшего

использования пользователем цифровых фотографий и видеороликов. Является одним из первых Web 2.0 сервисов. Один из самых популярных сайтов среди блогеров для размещения фотографий. По состоянию на 4 августа 2011 года сервис имел в своей базе более 6 млрд изображений, загруженных его пользователями. Также, по данным за август 2011 года, на сервисе было зарегистрировано 51 миллион человек, а общая посещаемость составляла 80 миллионов уникальных пользователей. Flickr был приобретён Yahoo! в марте 2005 года.

SmugMug — популярный сервис любителей и профессиональных фотографов, есть возможность создавать собственные портфолио. Фотографы могут выбрать из 24 элегантных тем и менять их нажатием одной кнопки. Помимо этого также существует большой спектр функций для точечного редактирования своего портфолио. Но, в отличие от других сервисов **SmugMug** не имеет бесплатной версии, поэтому после 14-дневного триала, придётся заплатить \$40 за стандартный план и \$300 если вы планируете продавать свои фотографии прямо на веб-сайте.

Яндекс.Фотки — сервис хранения фотографий от Яндекс. Отличительной чертой является то, что этот сервис предоставляет большое количество конкурсов для своих пользователей и даже даёт призы. Предоставляет простой, удобный интерфейс, большое количество функций и неограниченное пространство для фотографий. **Яндекс.Фотки** предоставляет приложения практически для всех существующих мобильных платформ, а также предоставляет *функцию встроенной печати*, позволяющую печатать фотографии прямо находясь на сайте.

Creative Cloud предоставляет полную коллекцию приложений Adobe для настольных ПК и мобильных устройств: от популярных базовых приложений, таких как Photoshop CC, до инструментов нового поколения, таких как Adobe XD CC. Кроме того, в пакет входят встроенные шаблоны для быстрого создания проектов и пошаговые руководства для совершенствования навыков и освоения функций. Все, что нужно для творчества, совместной работы и вдохновения. **Adobe Creative Cloud** — приложения для дизайна и работы с фотографиями, звуком и видео от Adobe:

- Photoshop CC — редактор растровой графики.
- Illustrator CC — векторная графика и иллюстрация.
- Illustrator Draw CC — векторная графика на мобильных устройствах.
- InDesign CC — издательская система.
- Adobe Stock — ресурсы для творческих проектов.
- Typekit — тысячи шрифтов от ведущих поставщиков.
- Acrobat XI Pro — создание и редактирование документов в формате PDF.
- Dreamweaver CC — разработка веб-сайтов.
- Adobe XD — создание дизайна, разработка прототипов и публикация пользовательского интерфейса.
- Adobe Muse — дизайн и публикация веб-сайтов без написания кода.

- Lightroom Classic — редактирование фотографий на настольных ПК.
- Lightroom CC — облачный фотосервис и 1 ТБ облачного хранилища.
- Audition CC — мультитрековый аудиоредактор.
- Premiere Pro CC — создание и монтаж видеоматериалов.
- After Effects CC — кинематографические визуальные эффекты и видеографика.
- Adobe Story — совместное написание сценариев и подготовка к съёмкам.
- Character Animator — анимация двухмерных персонажей в реальном времени.
- Premiere Clip — монтаж видео на мобильных устройствах.
- Spark Video — создание анимированных видео.
- Spark Page — преобразование текста и фотографий в привлекательные веб-материалы.
- Spark Post — разработка графики для социальных сетей за считанные секунды.

Список литературы

1. *Петров, М.* Компьютерная графика [Текст] / М. Петров, В. Молочков. — Второе изд. — СПб.: Питер, 2006. — 816 с.: ил.; 70 × 100/16 мм (170 × 240 мм, увеличенный). — ISBN 5-94723-758-X.
2. Алгоритмические основы растровой машинной графики [Текст] / Д. В. Иванов, А. С. Карпов, Кузьмин и др. — М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. — 286 с. (Основы информатики и математики). — ISBN 978-5-94774-654-9.
3. *Кариев, Ч. А.* Масштабируемая векторная графика (Scalable Vector Graphics) [Электронный ресурс] / Ч. А. Кариев. — М.: Интернет-университет информационных технологий — ИНТУИТ.ру, 2007. Режим доступа: <http://www.intuit.ru/department/graphics/svg/>.
4. *Царик, С. В.* Основы работы с CorelDRAW X3 [Электронный ресурс] / С. В. Царик. — М.: Интернет-университет информационных технологий — ИНТУИТ.ру, 2008. Режим доступа: <http://www.intuit.ru/department/graphics/corelx3/>.
5. *Платонова, Н. С.* Создание информационного буклета в Adobe Photoshop и Adobe Illustrator [Электронный ресурс] / Н. С. Платонова. — М.: Интернет-университет информационных технологий — ИНТУИТ.ру, 2009. Режим доступа: <http://www.intuit.ru/department/school/adobephill/>.

6. *Бондаренко, С. В.* Основы 3ds Max 2009 [Электронный ресурс] / С. В. Бондаренко, М. Ю. Бондаренко. — М.: Интернет-университет информационных технологий — ИНТУИТ.ру, 2008. Режим доступа: <http://www.intuit.ru/department/graphics/base3dmax2009/>.
7. *Платонова, Н. С.* Создание компьютерной анимации в Adobe Flash CS3 Professional [Электронный ресурс] / Н. С. Платонова. — М.: Интернет-университет информационных технологий — ИНТУИТ.ру, 2009. Режим доступа: <http://www.intuit.ru/department/school/adobeflashcs3p/>.
8. *Ватолин, Д. С.* Методы сжатия изображений [Электронный ресурс] / Д. С. Ватолин. — М.: Интернет-университет информационных технологий — ИНТУИТ.ру, 2007. Режим доступа: <http://www.intuit.ru/department/graphics/compression/>.

Глава 2

Издательские системы

1. Обзор издательских систем (ИС)

1.1. Системы визуального проектирования

Системы *визуального проектирования* WYSIWYG (What You See Is What You Get):

- QuarkXpress;
- PageMaker, FrameMaker, InDesign (Adobe);
- Corel Word Perfect, Corel Draw (Corel);
- Scribus;
- TeXmacs, LyX;
- ...

MS Word — не является ИС! ИС от Microsoft — MS Office Publisher. Однако многие издательства принимают небольшие статьи в формате *RTF* (формата *doc* не существует).

Для обмена используется *XML* или файлы формата *RTF*.

RTF (Rich Text Format, «формат обогащённого текста») — свободный межплатформенный формат хранения размеченных текстовых документов, предложенный Microsoft и др.

Первая версия стандарта *RTF* появилась в 1987г., с тех пор спецификация формата несколько раз изменялась, поэтому *имеет место несовместимость rtf-файлов разных форматов*. *RTF*-документы поддерживаются большинством современных текстовых редакторов (под Microsoft Windows это, как правило, осуществляется с помощью стандартных библиотек, входящих в состав операционной системы).

Доступность издательского ПО с функцией WYSIWYG вытеснила большинство языков разметки и логического проектирования среди обычных пользователей, хотя серьёзная издательская работа по-прежнему использует разметку для специфических не визуальных структур текста, а WYSIWYG-редакторы сейчас чаще всего сохраняют документы в форматах, основанных на языках разметки.

1.2. Системы логического проектирования

Системы *логического проектирования* WYSIWYM (What You See Is What You Mean), основанные на текстовом процессоре $\text{T}_{\text{E}}\text{X}$:

- Plain $\text{T}_{\text{E}}\text{X}$;
- \LaTeX , SLiTeX , AMS-TeX , $\text{\LaTeX 2}_{\epsilon}$, Con $\text{T}_{\text{E}}\text{X}t$, Omega;
- fp $\text{T}_{\text{E}}\text{X}$, te $\text{T}_{\text{E}}\text{X}$, TeXLive ;
- em $\text{T}_{\text{E}}\text{X}$ (MS DOS, OS/2);
- MiK $\text{T}_{\text{E}}\text{X}$ (Windows);
- oz $\text{T}_{\text{E}}\text{X}$, Mac $\text{T}_{\text{E}}\text{X}$, gw $\text{T}_{\text{E}}\text{X}$ (Apple/Mac OS X);
- pc $\text{T}_{\text{E}}\text{X}$ (Y&Y Inc.);
- Scientific Word (TCI Software Research Inc.);
- Personal $\text{T}_{\text{E}}\text{X}$;
- True $\text{T}_{\text{E}}\text{X}$;
- ...

Среди всего множества сборок $\text{T}_{\text{E}}\text{X}$ 'а, TeXLive является мультиплатформенным и наилучшим (по поддержке и обновлениям).

1.3. \LaTeX

В начале 80-х гг. *Лесли Лампортом* (*Leslie Lamport*) была разработана издательская система на базе $\text{T}_{\text{E}}\text{X}$ а, названная им \LaTeX ...

Преимущества ИС \LaTeX :

- соответствие стандарту *SGML*;
- полное разделение содержания документа с его оформлением благодаря концепции *общей разметки* (основываясь на опыте профессиональных типографских дизайнеров);
- совершенное полиграфическое качество;
- большое количество выходных форматов;

- полная совместимость для разных платформ;
- свободное распространение.

Последняя версия — $\text{\LaTeX} 2_{\epsilon}$ (предыдущая — $\text{\LaTeX} 2.09$, будущая — $\text{\LaTeX} 3$).

Во многих развитых компьютерных аналитических системах, таких, как **Maple**, **Mathematica**, **Maxima**, **Reduce**, **R/RStudio**, **Gnuplot** возможен экспорт документов в формат `.tex`. Для представления формул в Википедии также используется \TeX -нотация.

Глава 3

Введение в L^AT_EX

1. Рабочий процесс L^AT_EX

Исходный L^AT_EXовский файл является обычным текстовым файлом, содержащим, кроме текста, управляющие *команды*. Самая первая команда в исходном файле — `\documentclass{класс}` — определение *класса* документа.

Команда `\usepackage{пакет}` подключает дополнительные *пакеты*.

Сам текст документа должен быть написан между двумя командами:

```
\begin{document}
. . .
\end{document}
```

Подобные команды называются *окружениями* (environment), или *процедурами*.

В исходном файле можно комментировать строки знаком ‘%’. , закомментированные строки не компилируются. **Преамбула** — вводная часть исходного файла, предшествующая самому документу.

Пример .7 (Простая статья в L^AT_EX). `\documentclass[a4paper,10pt]{article}`
`\usepackage[cp1251]{inputenc}`
`\usepackage[russian]{babel}`

```
\title{Пример статьи} \author{Журенков~0.\,В.}
```

```
\begin{document}
\maketitle
\begin{abstract}
Очень простой пример
\end{abstract}
```

```
\section{Первый раздел} Какой-то текст.
\end{document}
```

Рассмотрим, как же работает такая издательская система. На рис. 3.1 показана логическая схема взаимодействия файлов, драйверов, программ и устройств.

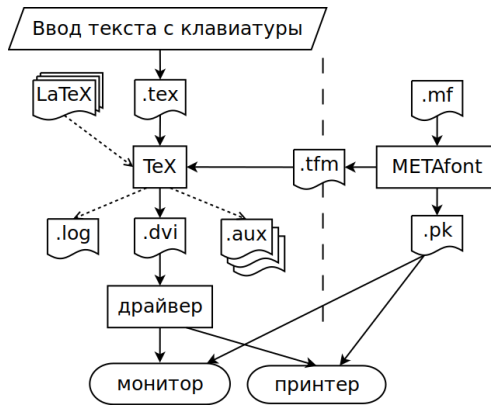


Рис. 3.1. Взаимодействие компонентов издательской системы \LaTeX

Как уже было сказано ... «Теховский» файл представляет из себя текстовый файл, включающий в себя, кроме обычного текста, набор управляющих команд, все они начинаются со знака ‘\’. Кроме этого знака существует ещё 9 служебных символов: ‘{’, ‘}’, ‘%’, ‘&’, ‘\$’, ‘#’, ‘~’, ‘^’, ‘_’.

В .log файл выводится протокол компиляции (немного подробнее, чем на экран), который полезно уметь читать. В начале выводится имя компилятора, его версия, дата. Далее читается компилируемый файл: о начале чтения любого файла свидетельствует открывающаяся круглая скобка ‘(’, когда файл полностью прочитан, скобка закрывается ‘)’.

% Символ % служит для комментирования текста до конца строки.

```

\documentclass[a4paper]{article} % Класс и опции печатного
                                % документа.
\usepackage[cp1251]{inputenc}   % Подключение
\usepackage[russian]{babel}     % дополнительных пакетов.

\title{Пример статьи}           % Заголовок документа.
\author{Журенков~0.\,В.}       % Автор документа.
%\date{}                        % Вы можете сами задать
%                               % дату.

\begin{document}                % Здесь заканчивается
                                % преамбула и начинается
                                % текст.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\maketitle                      % Делает титульный лист.

\section{Первый раздел}         % Это заголовок раздела.
  Какой-то текст. Простой текст.

```

Слова разделяются одним или несколькими пробелами,

конец строки считается тоже пробелом.

Абзацы

разделяются одной или несколькими строками. Добавление лишних пробелов или строк во входном файле не влияет на результат.

Одинарные кавычки печатаются как здесь: ‘выделенный текст’.

Двойные кавычки печатаются как здесь: ‘‘выделенный текст’’.

%

Длинный дефис (интервал) в предложениях печатается--как здесь. Длинное тире в предложениях печатается --- как здесь.

Курсив (*italic*) выглядит так: `\textit{это курсив}`. Жирный шрифт (**bold**) выглядит так: `\textbf{это жирный шрифт}`.

`\end{document}`

% Здесь заканчивается текст.

Что либо еще игнорируется.

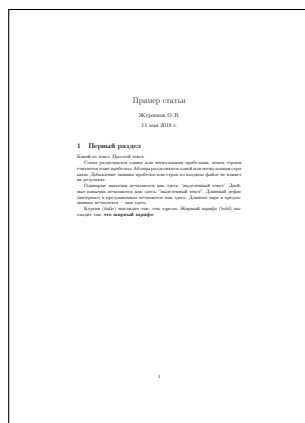


Рис. 3.2. Результат компиляции (.dvi файл) простого текста

Сообщения об ошибках и предупреждения

Сообщения об ошибках и предупреждения бывают двух видов: генерируемые \TeX ом и генерируемые \LaTeX ом.

Сообщение, об ошибке всегда начинается со знака '!'. Если ошибка обнаруживается \TeX ом, то сообщение следует сразу после знака '!'.
Пример .8 (ошибка \TeX). ! Missing } inserted.

`<inserted text>`

`}`

1.246 ... $k^{2} \tau$ $e^{-\lambda \tau} = \$$

`$`

Строка в исходном файле, на которой допущена ошибка указана ниже и начинается с буквы ‘l’. Далее выводится часть этой строки, причём, в том месте, где предполагается ошибка, строка разрывается.

Сообщения об ошибках ЛАТЭХа содержат фразу “LaTeX Error:”.

Пример .9 (ошибка ЛАТЭХа). ! LaTeX Error: \begin{document} ended by \end{equation}.

See the LaTeX manual or LaTeX Companion for explanation.

Type H <return> for immediate help.

...

1.47 \end{equation}

Your command was ignored.

Type I <command> <return> to replace it with another command, or <return> to continue without it.

При возникновении ошибки компиляция останавливается, компилятор приглашает к диалогу знаком ‘?’ . При этом можно ввести:

‘h’ — помощь;

‘e’ или ‘x’ — выход, прекращение компиляции;

‘s’ — продолжение компиляции без остановки;

‘r’ — то-же, что и ‘s’, но без вывода на экран;

‘i’ — ввод исправленной команды;

‘RETURN’ — продолжение компиляции до следующей ошибки.

Если ошибка не критическая, то возможно продолжение компиляции, в противном случае выводится знак ‘*’, тогда можно только прекратить компиляцию, нажав ‘Ctrl C’.

2. Команды и аргументы

Команды имеют один или несколько аргументов, исключение составляют только команды ввода символов. *Обязательные* аргументы пишутся в фигурных скобках после имени команды.

Замечание: Если обязательный аргумент пропущен, то в его качестве используется первый символ (не считая пробела), следующий за командой. Поэтому, при написании команд с аргументом из одного символа, скобки можно не писать.

Необязательные аргументы пишутся в квадратных скобках, перед обязательными аргументами. Если аргументов несколько, то они отделяются, чаще всего, запятыми, реже — пробелами.

Некоторые команды (в основном это команды, работающие в графической моде) имеют ещё аргументы-координаты, записываемые в круглых скобках.

Команда `\underline{текст}` служит для подчёркивания текста.

С помощью команды `\textcircled{символ}` можно обвести окружность вокруг любого символа!)

Строго говоря, команды можно разделить на *логосы*, *декларации*, собственно *команды* и *окружения*.

2.1. Логосы и декларации

Логосы, — команды, вставляющие текст (набор символов), обычно форматированные определённым образом.

Это в первую очередь имена разделов документа:

```
\partname, \chaptername, \contentsname, \listfigurename, \listtablename,  
\abstractname, \appendixname, \refname, \bibname, \indexname
```

а также некоторые «зарезервированные» слова:

```
\figurename, \tablename, \pagename, \seename, \alsoname.
```

Кроме того Л^AT_EX имеет свои логотипы:

```
\TeX, \LaTeX, \LaTeXe.
```

Многоточие — `\ldots` — тоже является логосом, как и команда текущей даты — `\today`.

Декларации ничего не печатают, а изменяют какой-либо режим (например, для изменения шрифтов, выравнивания текста, вставки отступа, ...).

Декларация `\protect` защищает следующую за ней команду, используется в подвижных аргументах (когда аргумент передаётся в другую команду, как при создании содержания, ссылки и т. п.).

2.2. Правила скобок

1. *TeX* проверяет соответствие скобок $\{$ и $\}$, которые служат для группировки; окружения (`\begin ... \end`) тоже выполняют роль скобок (называемых командными).
2. Декларации действуют в пределах фигурных и командных скобок, значит декларации можно писать в аргументах команд и в теле окружений.

3. Буквы и символы

Л^AT_EX распознаёт строчные и прописные буквы, десять цифр и 16 знаков препинания.

Спецсимволы, требующие особого ввода

Символ	Назначение	Ввод
\	Признак команды	<code>\backslash\$</code>
{	Начало группы	<code>\{</code>
}	Конец группы	<code>\}</code>
%	Комментарий	<code>\%</code>
&	Табулятор	<code>\&</code>
~	Неразрываемый пробел	<code>\~{ }</code>
\$	Начало и конец математики	<code>\\$</code>
^	Верхний индекс	<code>\^{}{ }</code>
_	Нижний индекс	<code>_{}{ }</code>
#	Признак подстановки в макросах	<code>\#</code>

3.1. Специальные символы

Существует 10 символов, которые в \LaTeX имеют особое назначение. Полный список этих символов и команд их ввода представлен в табл. 3.1.

3.2. Акценты

Как правило, для того, чтобы поставить акцент на какой-нибудь букве, перед буквой печатается соответствующее **командное слово** или **командный символ** (буквы с акцентами широко используются во многих европейских языках, в русском языке — это ‘ё’ и й).

Вот несколько примеров:

Ввод в \LaTeX	\LaTeX
a la mode	à la mode
resume	résumé
soup\c{c}on	soupċon
No\"e1	Noël
na\"i ve	naïve

Большинство акцентов печатается с помощью командных символов такой же формы. Несколько из них печатается с помощью командных слов, содержащих одну букву.

Существует командное слово `\i` (для получения *i* — строчной буквы *i* без точки над ней) и `\j` (для получения *j* — строчной буквы *j* без точки над ней). Эти команды позволяют поставить другой знак акцента над верхней частью соответствующей буквы.

Командные символы акцентирования приведены в табл. 3.2.

Командные слова для акцентирования приведены в табл. 3.3.

\LaTeX позволяет печатать нетерминальные символы из европейских алфавитов. Эти команды приведены в табл. 3.4.

Таблица 3.2.

Акцентирование с помощью командных символов

Название	Ввод	Результат
grave	<code>\'o</code>	ò
acute	<code>\'o</code>	ó
circumflex	<code>\^o</code>	ô
umlaut/dieresis/trémat	<code>\"o</code>	ö
tilde	<code>\~o</code>	õ
macron	<code>\=o</code>	ō
dot	<code>\.o</code>	ô

Таблица 3.3.

Акцентирование с помощью очень коротких командных слов

Название	Ввод	Результат
cedilla	<code>\c{o}</code>	ç
underdot	<code>\d{o}</code>	ȝ
underbar	<code>\b{o}</code>	ȕ
háček	<code>\v{o}</code>	ǎ
breve	<code>\u{o}</code>	ǔ
tie	<code>\t{oo}</code>	ôô
Hungarian umlaut	<code>\H{o}</code>	ő

Иностранные нетерминальные символы

Пример	Ввод	Результат
Ægean, æsthetics	<code>\AE, \ae</code>	Æ, æ
Œuvres, hors d'œuvre	<code>\OE, \oe</code>	Œ, œ
Ångstrom	<code>\AA, \aa</code>	Å, å
Øre, København	<code>\O, \o</code>	Ø, ø
Łodz, łódka	<code>\L, \l</code>	Ł, ł
Nuß	<code>\ss</code>	ß
¿Si?	<code>?‘</code>	¿
¡Si!	<code>!‘</code>	¡

Многие из этих символов являются лигатурами латинского алфавита.

***Лигату́ра** (лат. *ligatura* — связь), — это знак любой системы письма или фонетической транскрипции, образованный путём соединения двух и более терминальных (клавиатурных) символов.*

По не слившимся частям (иногда изменяющим свою форму) обычно можно видеть, какие буквы входят в состав лигатуры. В некоторых системах письма лигатуры многих буквенных сочетаний вошли во всеобщее употребление, являясь как бы сложными буквенными знаками для изображения определённых звуковых комплексов.

В ИС \LaTeX для ввода некоторых часто используемых нетерминальных символов используются **лигатуры**, вводимые терминальными символами. В основном, это тире и кавычки. Некоторые сочетания символов латиницы тоже образуют лигатуры (когда вместо нескольких вводимых символов получается один типографский символ).

Пример .10 (лигатуры в латинице). ff ff fi fi

`ff f{f}` `fi f{i}`

3.3. Тире, кавычки, многоточия

Используется четыре типа тире:

дефис — в сложных словах;

интервал — для обозначения числовых интервалов;

тире — как грамматический знак в предложениях;

Таблица 3.5.

Различные типы тире

Название	Ввод	Результат
hyphen	-	-
en-dash	--	—
em-dash	---	—
minus sign	\$-\$	—

Таблица 3.6.

Кавычки и многоточие

Название	Ввод	Результат
левые кавычки	`	‘
правые кавычки	'	’
левые двойные кавычки	``	“
правые двойные кавычки	''	”
левые французские кавычки ¹	<< "< < \<	«
правые французские кавычки ¹	>> "> > \>	»
многоточие	\ldots	...

минус — используемый с отрицательными числами.

Способы ввода этих символов представлен в табл. 3.5.

Тире различной длины вводятся лигатурами.

В тексте часто используются кавычки и многоточия. Способы ввода этих символов представлен в табл. 3.6. Как и тире, эти символы вводятся, в основном, **лигатурами**.

В русскоязычном (кириллическом) тексте принято использовать французские кавычки («...»).

4. Шрифты

fonts

Для выделения текста используется команда `\emph{текст}`.

4.1. Пользовательские команды выбора шрифтов

В большинстве случаев пользователю достаточно небольшого набора шрифтов, отличающихся разными характеристиками. Команды для выбора таких шрифтов

Таблица 3.7.

Пользовательские команды выбора шрифтов

Название шрифта	Команда	Декларация	Образец печати
Прямой	<code>\textrm{текст}</code>	<code>\rmfamily \rm</code>	roman
Средний	<code>\textmd{текст}</code>	<code>\mdseries</code>	medium
Жирный	<code>\textbf{текст}</code>	<code>\bfseries \bf</code>	boldface
Курсив	<code>\textit{текст}</code>	<code>\itshape \it</code>	<i>italic</i>
Наклонный	<code>\textsl{текст}</code>	<code>\slshape \sl</code>	<i>slanted</i>
Без отточий	<code>\textsf{текст}</code>	<code>\sffamily \sf</code>	sans serifed
Строчные, как прописные	<code>\textsc{текст}</code>	<code>\scshape \sc</code>	SMALL CAPS
Равноширинный	<code>\texttt{текст}</code>	<code>\ttfamily \tt</code>	typewriter

Таблица 3.8.

Пользовательские команды задания размера шрифтов

Команда	Образец
<code>\tiny</code>	Sample text.
<code>\scriptsize</code>	Sample text.
<code>\footnotesize</code>	Sample text.
<code>\small</code>	Sample text.
<code>\normalsize</code>	Sample text.
<code>\large</code>	Sample text.
<code>\Large</code>	Sample text.
<code>\LARGE</code>	Sample text.
<code>\huge</code>	Sample text.
<code>\Huge</code>	Sample text.

представлены в табл. 3.7.

4.2. Пользовательские команды изменения размера шрифтов

Кроме шрифта, иногда возникает необходимость изменить размер шрифта. L^AT_EX в своём арсенале имеет набор пользовательских команд для изменения размера шрифтов, они представлены в табл. 3.8.

5. Низкоуровневые команды задания шрифтов

После создания NFSS (New Font Selection Schem), называемой ещё «ортогональной схемой выбора шрифтов», стало возможным задание ещё большего многообразия шрифтов.

Стандартные внутренние кодировки $\LaTeX 2_{\epsilon}$

<i>enc</i>	Кодировка	Примечание
OT1	\TeX text encoding	Кодировка Д. Кнута
T1	\TeX extended text encoding	«Корковская» кодировка
U	Unknown	Неизвестная кодировка
Lxx	A Local encoding	Местная кодировка

Каждый шрифт характеризуется определённым набором параметров: **кодировка, гарнитура (семейство), насыщенность, начертание и размер**.

Теперь каждый параметр можно изменять независимо от других, чего нельзя было делать в $\LaTeX 2.09$.

5.1. Кодировка

В \TeX е введена своя кодировка, называемая *внутренней* (общепринятая кодировка или кодовая страница, в таком случае является *внешней*).

Первая кодировка (\TeX text encoding), введённая Д. Кнудом, в $\LaTeX 2_{\epsilon}$ называется **OT1** (Old Text 1) и содержит всего 128 символов.

В 1990 г., на конференции пользователей \TeX а была принята кодировка \TeX extended text encoding, называемая ещё «корковской»¹. Она содержит 256 символов, в неё включены все символы алфавитов романской группы (западно-европейских стран).

Задаётся кодировка с помощью команды `\fontencoding{enc}`. Список допустимых кодировок *enc* представлен в табл. 3.9.

Пример .11 (задание кодировки). `\fontencoding{OT1}`

Для русских шрифтов часто используется кодировка LCY или LH. Но местные (локальные) кодировки \LaTeX не знает. Lxx кодировки следует подключать через пакет `fontenc`², например: `\usepackage[LCY]{fontenc}`.

Старый способ задания русских шрифтов, основывался на кодировке OT1, дополненной 128 символами, содержащими символы русского алфавита.

В современных версиях используются официальные русские шрифты T2.

5.2. Непосредственное задание символа

Команда `\symbol{код}` вставляет в текст символ с кодом «код» внутренней кодовой таблицы. Код можно задать десятичным числом, восьмеричным (начинающимся с апострофа ') или шестнадцатеричным (начинающимся с кавычек ").

¹Название происходит от места проведения конференции, — ирландского городка Корк (Cork)

²Пакеты русификации уже содержат такие команды, так что пользователю, обычно, не надо задумываться о кодировке.

Такой способ задания символов обычно используется в определении новых команд.

5.3. Гарнитура

Гарнитура (семейство шрифтов) определяет дизайнерское решение, общее для разных шрифтов.

В группе шрифтов Computer Modern, созданных Д. Кнудом, есть три гарнитуры: Roman, Serif и Typewriter. Основные отличительные черты гарнитуры, это *контрастность, пропорциональность и засечки.*

Контрастность определяется отношением толщины вертикальных и горизонтальных линий символов.

Пример .12 (контрастность гарнитур). Гарнитура Computer Modern Roman более контрастная, чем Computer Modern Serif.

Пропорциональность задаётся отношением ширины к высоте буквы ‘М’.

Пропорциональность тесно связана с шириной букв. В пропорциональных шрифтах занимаемое буквой место пропорционально действительной ширине буквы, как например, в Computer Modern Roman.

Кроме того, шрифт может быть **моноширинный** (когда каждая буква занимает одинаковое по ширине место). Пример **моноширинного** шрифта — Computer Modern Typewriter.

*Засечки располагают на вертикальных линиях букв, такие шрифты ещё называют **романскими**.*

*Шрифты без засечек называют **рублёнными**.*

Пример .13 (засечки в шрифтах). Гарнитура Computer Modern Roman имеет засечки, что отображено в названии, а Computer Modern Serif является рублённым шрифтом.

Задаётся гарнитура с помощью команды `\fontfamily{family}`.

Основные семейства кириллических шрифтов в L^AT_EX 2_ε приведены в табл. 3.10.

5.4. Насыщенность

Насыщенность — относительная толщина линий букв.

В L^AT_EX 2_ε насыщенность задаётся с помощью команды `\fontseries{series}`. Как видно из табл. 3.11, некоторые серии задают не только насыщенность но и пропорциональность.

Таблица 3.10.

Наиболее распространённые семейства шрифтов в $\text{\LaTeX} 2_{\epsilon}$

<i>family</i>	Гарнитура	Примечание
cmr	Computer Modern Roman	Романский
cmss	Computer Modern Sans	Рублёный
cmtt	Computer Modern Typewriter	Машинописный
cmdh	Computer Modern Dunhill	Данхил
cmfib	Computer Modern Fibonacci	Фиббоначи
cmfr	Computer Modern Funny	Забавный

Таблица 3.11.

Стандартные значения параметра *series*

<i>series</i>	Насыщенность	Примечание
l	Light	Светлый шрифт
m	Medium	Нормальный шрифт
b	Bold	Жирный шрифт
bx	Bold extended	Широкий жирный шрифт
sb	Semi-bold	Полужирный шрифт
c	Condensed	Узкий шрифт

5.5. Начертание

Начертание *определяет форму шрифта*.

Задаётся начертание командой `\fontshape{shape}`. Основные начертания $\LaTeX 2_\epsilon$ приведены в табл. 3.12.

Таблица 3.12.

Стандартные значения параметра *shape*

<i>shape</i>	Начертание	Примечание
n	Normal	Прямой шрифт
it	<i>Italic</i>	<i>Курсивный шрифт</i>
sl	<i>Slanted (oblique)</i>	<i>Наклонный шрифт</i>
sc	CAPS AND SMALL CAPS	КАПИТЕЛЬ
ui	Upright italic	Прямой курсивный шрифт

5.6. Кегль

Кегль или размер шрифта измеряют в печатных пунктах (pt). В одном дюйме содержится 72,27 pt.

Задать размер шрифта можно с помощью команды `\fontsize{size}{baselineskip}`. Эта команда, кроме размера *size*, содержит ещё один параметр — *baselineskip*, который определяет расстояние между строчками.

Значения параметров могут быть любыми, в произвольных единицах измерения (например, `\fontsize{15pt}{5mm}`), однако, делать это надо осторожно.

В профессиональных издательских системах используются кегли 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 24, 30, 36 pt.

Шрифты **Computer Modern** спроектированы для размеров 5, 6, 7, 8, 9, 10, 12, 17 pt, другие размеры получаются линейным масштабированием.

В большинстве случаев можно обойтись пользовательскими командами задания размера шрифтов.

Пользовательские декларации переключения размера шрифтов задают линейку кеглей 5, 6, 7, 8, 9, 10, 10.95, 12, 14.4, 17.28, 20.74, 24.88 pt.

5.7. Переключение шрифтов

Любое изменение шрифта вступает в силу только после декларации `\selectfont`, которая должна следовать сразу же после изменения параметров шрифтов.

Существует альтернативный вариант задания шрифта, — с помощью команды `\usefont{enc}{family}{series}{shape}`, задающей четыре параметра (все, кроме размера).

Все эти команды действуют только в текстовой и строковой модах.

В *математической* моде существуют свои, математические шрифты, которые определяются своими командами.

Графические символы, используемые в *графической* моде, задаются специальными кодами.

Глава 4

Вёрстка и форматирование

1. Управление строками

Вёрстка — размещение контента, монтаж полос оригинал-макета из составных элементов (текста, заголовков, таблиц, иллюстраций и пр.).

Под **вёрсткой** понимают также результат этого процесса, то есть готовые полосы.

Строки в Л^AT_EX'e формируются автоматически, но встречаются ситуации, когда требуется «ручное вмешательство».

1.1. Горизонтальные пробелы

Для формирования красивого, более воспринимаемого текста, T_EX использует технологии **кернения** и **трекинга**.

Кернение — изменение расстояния между определённой парой символов.

Пример .14 (кернение).

Где лучше?	— без кернения
Где лучше?	— с кернением

Трекинг — изменение расстояния между символами в зависимости от кегля (размера шрифта).

Команды управления пробелами в тексте представлены в табл. 4.1.

`\hspace{длина}` — вставляет пробел заданной длины.

`\hspace*{длина}` — вставляет пробел даже на краю строки.

Единицы измерения, используемые в T_EX'e, представлены в табл. 4.2.

`\hfil` — вставляет пробел длины `fil`.







Таблица 4.1.

Пробелы в L^AT_EX 2_ε

<code>_</code>	пробел нормального размера	_
<code>\,</code>	маленький пробел	,
<code>\~</code>	неразрывный пробел	~
<code>\/</code>	корректирующий пробел	/
<code>\quad</code>	пробел, шириной с букву 'М'	___
<code>\qquad</code>	пробел, шириной как две 'М'	_____

Таблица 4.2.

Единицы измерения в T_EX_ε

<code>mm</code>	миллиметр ≈ 1/25 дюйма	
<code>cm</code>	сантиметр = 10 mm	
<code>in</code>	inch = 25,4 mm	
<code>pt</code>	пункт ≈ 1/72 дюйма ≈ 1/3 mm	
<code>em</code>	примерная ширина буквы 'М' текущего шрифта	
<code>ex</code>	примерная высота буквы 'x' текущего шрифта	

`\hfill` — вставляет пробел длины `fill`.

`\@` — увеличивает пробел в конце предложения.

`\frenchspacing` — декларирует подавление дополнительного пробела в конце предложения.

`\nofrenchspacing` — декларирует дополнительный пробел в конце предложения, действует по умолчанию.

`\fussy` — декларирует «плотные строки».

`\sloppy` — декларирует «разряжённые строки», этой декларации соответствует окружение `sloppypar`.

`\emergencystretch` — длина, которая задаёт степень разряжённости строк (изменяется в преамбуле).

Пример .15 (степень разряжённости строк). `\emergencystretch=1cm`.

1.2. Переносы

Запретить переносы можно с помощью строкового бокса `\mbox{текст}` (см. на стр. 4.1.).

Подсказать правильные переносы можно с помощью декларации `\-`, вставленной в месте предполагаемого переноса.

Пример .16 (переносы в сложных словах). `ку\ -соч\ -но-не\ -пре\ -рыв\ -ная функция.`

В преамбуле указываются правила переноса для часто используемых слов с помощью команды `\hyphenation{сло-во сло-во ...}`.

Пакет `babel` поддерживает около 200 языков.

1.3. Разрыв строки

Повлиять на разрыв строк можно с помощью следующих команд:

- `\linebreak[N]` — поощряет с «силой» N ($N = 0 \div 4$) разрыв строки, причём текущая строка выравнивается по правому краю. $N = 0$ соответствует отсутствию команды, а $N = 4$ — обязательному разрыву (действует по умолчанию).
- `\nolinebreak[N]` — препятствует с «силой» N разрыву строки. $N = 0$ соответствует отсутствию команды, а $N = 4$ — запрету разрыва (действует по умолчанию).
- `\\[высота]` — начинает новую строку, не завершая абзац. Необязательный аргумент *высота* задаёт дополнительный вертикальный пробел, вставляемый перед новой строкой. Модификация этой команды (`*[высота]`) запрещает перенос новой строки на следующую страницу.
- `\newline` — действует аналогично `\\`.

При формировании строк могут появляться предупреждения Т_EX’а:

- “`Underfull \hbox (badness ...`” — незаполненный горизонтальный бокс (в скобках указана относительная величина разряжённости);
- “`Overfull \hbox (...`” — переполненный горизонтальный бокс (в скобках указана длина переполнения в пунктах).

1.4. Абзацы

Абзацы при наборе разделяются пустыми строками. В некоторых случаях удобнее пользоваться командой `\par`.

`\indent` — вставляет абзацный отступ. Эта команда равносильна `\hspace{\parindent}`, где `\parindent` — длина абзацного отступа.

`\noindent` — подавляет абзацный отступ (ставится в начале абзаца).

2. Управление страницами

Страницы в L^AT_EX'e, как и строки, формируются автоматически, при этом также встречаются ситуации, когда требуется «ручное вмешательство».

2.1. Вертикальные пробелы

`\vspace{высота}` — вставляет между строками пробел заданной *высоты*.

`\vspace*{высота}` — вставляет пробел заданной *высоты* даже на краю страницы.

Для длин `fil` и `fill` имеются аббревиатуры: `\vfil` и `\vfill`.

`\addvspace{высота}` — добавляет отступ, необходимый для достижения вертикального пробела между абзацами высотой *высота*. Применяется только в текстовой моде и пишется обязательно на отдельной строке.

Для увеличения вертикального отступа между абзацам существуют команды:

- `\smallskip` — малый отступ (высотой `\smallskipamount`).
- `\medskip` — отступ высотой `\medskipamount` (средний, в 2 раза больше малого).
- `\bigskip` — отступ высотой `\bigskipamount` (большой, в 2 раза больше среднего).

`\baselinestretch` — определяет расстояние между строками (как в команде `\fontsize`), пишется в преамбуле.

Пример .17 (расстояние между строками во всём документе). Например, для задания интервала 1,5 надо написать `\renewcommand{\baselinestretch}{1.5}`.

2.2. Страницы

`\flushbottom` — включает режим, когда все страницы выровнены по нижнему краю (действует по умолчанию при включении опции класса `twoside`).

`\raggedbottom` — включает режим, когда страницы не выровнены.

При этом могут появляться предупреждения компилятора:

- “Underfull \vbox (badness ...” — незаполненный вертикальный бокс (в скобках указана относительная величина разряжённости);
- “Overfull \vbox (...” — переполненный вертикальный бокс (в скобках указана высота переполнения в пунктах).

Повлиять на разбиение страниц можно с помощью следующих команд:

- `\nopagebreak[N]` — препятствует с «силой» N разрыву страницы ($N = 0 \div 4$). $N = 0$ соответствует отсутствию команды, а $N = 4$ — запрещает разрыв (действует по умолчанию).
- `\samepage` — запрещает разрыв страницы, так же как и `\nopagebreak`.
- `\pagebreak[N]` — поощряет с «силой» N разрыв страницы. $N = 0$ соответствует отсутствию команды, а $N = 4$ — обязательному разрыву (действует по умолчанию).
- `\newpage` — начинает новую страницу, так же как и `\pagebreak` (при двухколоночной печати эти команды начинают новую колонку). При этом текущая страница получается укороченной, даже если включена декларация `\flushbottom`.
- `\clearpage` — выводит на печать оставшиеся плавающие объекты и начинает новую страницу.
- `\cleardoublepage` — действует как и команда `\clearpage`, но при двусторонней печати, при необходимости, добавляет пустую левую страницу.
- `\enlargethispage{высота}` — увеличивает высоту страницы на величину *высота*. Модификация команды (`\enlargethispage*{высота}`) при этом сжимает все сжимаемые пробелы.

3. Форматирование текста

Форматирование (текста) — процесс придания тексту определённого вида.

В этом деле L^AT_EX использует логическую разметку. Физическую разметку, через задание параметров шрифтов, мы рассмотрели ранее, см. 3.7 раздел.

3.1. Выравнивание

По умолчанию, текст выравнивается T_EX’ом по ширине. Другие варианты выравнивания представлены в табл. 4.3.

Команды выравнивания используются для коротких строк, в них текст не переносится.

Команды выравнивания текста

	Окружение	Декларация	Команда
по центру	<code>center</code>	<code>\centering</code>	<code>\centerline{текст}</code>
влево	<code>flushleft</code>	<code>\raggedright</code>	<code>\leftline{текст}</code>
вправо	<code>flushright</code>	<code>\raggedleft</code>	<code>\rightline{текст}</code>

3.2. Цитаты и стихи

Для выделения абзацев равными отступами слева и справа существует два окружения:

- `quote` — используется для небольших цитат (фрагментов текста);
- `quotation` — используется для больших фрагментов текста, состоящих из нескольких абзацев.

Для форматирования стихов используется окружение `verse`. Каждая строка должна заканчиваться командой `\\`, а строфы отделяться пустыми строками.

3.3. Списки

Л^AT_EX имеет в своём арсенале три вида списков, которые задаются соответствующими окружениями:

1. `enumerate` — нумерованный список;
2. `itemize` — маркированный (нечисловой) список;
3. `description` — список-описание.

Каждый элемент списка начинается с команды `\item[метка]`, необязательный аргумент *метка* можно использовать для задания номера или маркера (для первых двух списков). В третьем списке этот аргумент обязателен, в нём задаётся описываемое слово.

3.4. Неформатированный текст

Для изображения текста без форматирования используется окружение `verbatim`. Модификация `verbatim*` отображает пробелы символом `_`.

Для отображения небольшой части неформатированного текста определена команда `\verb<c>{текст}<c>`, в качестве `<c>` можно использовать любой, отсутствующий в тексте, символ.

Команда `\verb*<c>{текст}<c>` так-же отображает пробелы.

Пример .18 (неформатированный текст). `\verb|\LaTeX <-- logos| \verb*|\LaTeX <-- lo`
`\LaTeX <-- logos` `\LaTeX_ _ _ _ <--_ logos`

3.5. Сноски и заметки

Текст, который пишется внизу страницы, называется *сноской*¹.

Сноски формируются командой `\footnote[N]{текст}`, она ставится сразу же после слова, без пробела. Сноски нумеруются автоматически. С помощью N можно изменить нумерацию.

`\footnotemark[N]` — ставит только маркер, увеличивая значение своего счётчика.

`\footnotetext[N]{текст}` — пишет *текст* подстрочного примечания.

Кроме сносок в тексте можно использовать **заметки**, которые пишутся на полях, напротив нужного места в тексте. Для **заметок** используется команда `\marginpar[текст слева]{текст}`.

Необязательный аргумент *текст слева* используется при двусторонней печати, если *текст* заметки должен быть разным при появлении заметки на левой и правой странице.

Декларация `\reversmarginpar` меняет текст левых и правых заметок, а

`\normalmarginpar` восстанавливает нормальный режим.

4. Боксы

Бокс — это прямоугольная область, с которой $\text{T}_\text{E}_\text{X}$ оперирует как с единым, неделимым объектом.

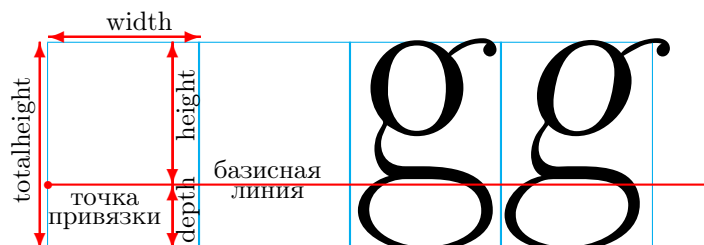


Рис. 4.1. Бокс и его параметры

Параметры **бокса** задаются размерными величинами *width*, *depth*, *height*, *totalheight*.

4.1. Строковые боксы

Внутри строкового бокса $\text{T}_\text{E}_\text{X}$ работает в *строковой моде*, т. е. *текст не может быть разорван*.

¹Синоним — подстрочные примечания.

`\mbox{текст}` — помещает аргумент *текст* в бокс, размеры бокса чуть больше фактических размеров аргумента.

`\fbox{текст}` — создаёт такой-же бокс как и команда `\mbox`, но ещё и окружает бокс рамкой.

`\makebox[ширина][позиц.]{текст}` — помещает аргумент *текст* в бокс, размеры которого можно задать с помощью аргумента *ширина*. Если задан аргумент *ширина*, то можно задействовать аргумент *позиц.*, задающий позиционирование содержимого бокса.

Аргумента *позиц.* может принимать следующие значения:

- l — текст сдвинут к левому краю бокса;
- c — текст расположен по центру бокса;
- r — текст сдвинут к правому краю бокса;
- s — текст растянут по всему боксу.

По умолчанию действует опция *c*.

`\framebox[ширина][позиц.]{текст}` — создаёт такой-же бокс как и команда `\makebox`, но ещё и окружает бокс рамкой.

`\raisebox{высота}[height][depth]{текст}` — изменяет положение текста относительно базисной линии, — поднимает на высоту *высота* (или опускает, если *высота* отрицательная). Необязательные аргументы *height* и *depth* изменяют естественные значения этих параметров.

4.2. Текстовые боксы

Внутри текстового бокса Т_ЕX работает в *текстовой моде*.

`\parbox[внеш.позиц.][высота][позиц.]{ширина}{текст}` создаёт бокс шириной *ширина*, форматированный *текст* в виде одного абзаца. Аргумент *внеш.позиц.* задаёт положение бокса относительно текущей базисной линии:

- t — по текущей базисной линии выравнивается базисная линия *верхней строки*;
- c — по текущей базисной линии выравнивается *центр бокса*;
- b — по текущей базисной линии выравнивается базисная линия *нижней строки*.

По умолчанию действует опция *c*.

Аргумент *высота* изменяет высоту бокса.

Вертикальное позиционирование текста внутри бокса задаётся аргументом *позиц.*:

- t — текст сдвинут к верхнему краю бокса;
- c — текст расположен по центру бокса;
- b — текст сдвинут к нижнему краю бокса;

s — текст растянут по всему боксу за счёт растяжимых вертикальных пробелов (если таковые имеются).

По умолчанию действует опция, указанная в аргументе *внеш.позиц.*

Окружение `minipage[внеш.позиц.][высота][позиц.]{ширина}` создаёт текстовый бокс шириной *ширина* со всеми возможностями обычного текста.

Аргумент *внеш.позиц.* так-же задаёт положение бокса относительно текущей базисной линии.

С помощью аргумента *высота* можно изменить высоту бокса, а с помощью аргумента *позиц.* так-же задать и вертикальное позиционирование текста внутри бокса.

4.3. Линейные боксы

С помощью линейных боксов можно создавать декоративные или невидимые линейки.

`\rule[смещение]{ширина}{высота}` — создаёт линейный бокс с размерами *ширина* и *высота*. С помощью аргумента *смещение* задаётся смещение по вертикали относительно базисной линии.

`\hrulefill` — создаёт тонкую горизонтальную линию длиной fill_____.

`\dotfill` — создаёт пробел fill, заполненный точками

`\strut` — создаёт бокс нулевой ширины («распорку»), высота и глубина которого максимальны для букв текущего шрифта.

Пример .19 (применение распорки). `\underline{без} \underline{распорки}`
`\quad`
`\underline{c\strut} \underline{распоркой\strut}`
без распорки с распоркой

В математической моде имеется аналогичная команда `\mathstrut`.

4.4. Сохранение бокса

Кроме рассмотренных команд в ЛАТЭХ'е существует ряд команд для измерения размеров боксов, для определения, сохранения и использования новых боксов.

Повторяющийся в документе сложный текст (рисунки, таблицы, трудоёмкое форматирование) лучше отформатировать один раз, «запомнить» его в виде бокса и использовать в уже «готовом» виде. Именно таким образом задаются логосы.

Для работы с такими боксами имеются следующие команды: `\newsavebox{имя}` — задаёт *имя* бокса.

`\sbox{имя}{текст}` — форматирует *текст* в строковой моде (действует как `\mbox`, но не выводит на печать содержимое бокса).

Окружение `lrbox{имя}` действует аналогично команде `\sbox`, с тем отличием, что все пробелы в начале и конце текста будут проигнорированы. Кроме того, в

тексте могут быть неформатированные участки (созданные командой `\verb` или окружением `verbatim`).

`\savebox[ширина][позиц.]{\имя}{текст}` — форматирует *текст* в строковой моде. Можно задать *ширину* бокса и *позиционирование текста* (действует как `\makebox`, но не выводит на печать содержимое бокса). Опция *позиц.*, может принимать те же значения:

- l — текст сдвинут к левому краю бокса;
- c — текст расположен по центру бокса;
- r — текст сдвинут к правому краю бокса;
- s — текст растянут по всему боксу.

По умолчанию действует опция *c*.

`\usebox{\имя}` — печатает бокс *имя*.

Пример .20 (переиспользование бокса). `\newsavebox{\Help}`

`\savebox{\Help}[2em]{\fbox{\texttt{F1}}}`

Клавиша `\usebox{\Help}` --- помощь.

Клавиша `F1` — помощь.

5. Команды в L^AT_EX 2_ε

5.1. Командные длины

Командные длины — команды, выражающие определённые длины.

Для работы с произвольными командными длинами существуют следующие команды:

- `\newlength{\имя}` — задание новой командной длины.
- `\setlength{\имя}{длина plus длина minus длина}` — установка длины команды *имя*.
- `\addtolength{\имя}{длина plus длина minus длина}` — увеличение длины команды *имя*.

Команды для измерения строковых боксов:

- `\settowidth{\длина}{текст}` — измеряет ширину строкового бокса, образованного аргументом *текст*. Измеренная длина присваивается аргументу `\длина`.
- `\settoheight{\длина}{текст}` — измеряет высоту строкового бокса, образованного аргументом *текст*.

Основные счётчики ЛАТЭХ'а

Имя	Что нумерует
<code>page</code>	страницы
<code>part</code>	части
<code>chapter</code>	главы
<code>section</code>	разделы
<code>subsection</code>	подразделы
<code>subsubsection</code>	подподразделы
<code>paragraph</code>	параграфы
<code>subparagraph</code>	подпараграфы
<code>enumi</code>	элементы списка 1-го уровня
<code>enumii</code>	элементы списка 2-го уровня
<code>enumiii</code>	элементы списка 3-го уровня
<code>enumiv</code>	элементы списка 4-го уровня
<code>footnote</code>	подстрочные примечания
<code>mpfootnote</code>	подстрочные примечания в министраницах
<code>equation</code>	уравнения
<code>table</code>	таблицы
<code>figure</code>	рисунки

- `\settodepth{длина}{текст}` — измеряет глубину строкового бокса, образованного аргументом *текст*.

Пример .21 (использование командных длин). `\newlength{\bx}`
`\settoheight{\bx}{\fbox{F8}}`
`\addtolength{\bx}{2\fboxsep plus 2pt}`

Клавиша `\framebox[\bx]{F1}`

--- помощь.

Клавиша `F1` — помощь.

5.2. Счётчики

Основные счётчики ЛАТЭХ'а представлены в табл. 4.4.

Внутренние счётчики подчинены (подчинённые счётчики) своим внешним счётчикам и автоматически сбрасываются при изменении внешних счётчиков.

Пример .22 (подчинённость счётчиков). Счётчик `subsection` является внешним для счётчика `subsubsection` и внутренним для `section`.

Большинство счётчиков устанавливаются в нуль в начале компиляции. Команды и окружения, работающие со счётчиками, сначала увеличивают значение счётчика, а потом уже используют его.

Для работы непосредственно со счётчиками существуют команды:

`\thesчётчик` — печатает значение счётчика в формате, определённом для этого счётчика.

Всего имеется 6 форматов: *arabic*, *roman*, *Roman*, *alph*, *Alph*, *fnsymbol*. Им соответствуют следующие команды:

`\arabic{счётчик}` — печатает значение *счётчика* арабскими цифрами;

`\roman{счётчик}` — печатает значение *счётчика* строчными римскими цифрами;

`\Roman{счётчик}` — печатает значение *счётчика* прописными римскими цифрами;

`\alph{счётчик}` — печатает значение *счётчика* строчными буквами английского алфавита (значение счётчика ≤ 26);

`\Alph{счётчик}` — печатает значение *счётчика* прописными буквами английского алфавита (значение счётчика ≤ 26);

`\fnsymbol{счётчик}` — печатает значение *счётчика* специальными символами, часто используемыми для подстрочных примечаний в `minipage` (значение счётчика ≤ 9).

`\setcounter{счётчик}{N}` — задаёт значение *счётчика* равное N (N — целое, возможно отрицательное).

`\addtocounter{счётчик}{N}` — увеличивает значение *счётчика* на N .

`\value{счётчик}` — возвращает «чистое» значение счётчика. Может использоваться в любых командах, где L^AT_EX ожидает число, в основном — в аргументе N команд `\setcounter` и `\addtocounter`. Эта команда устойчивая, её никогда не следует защищать командой `\protect`.

`\stepcounter{счётчик}` — наращивает (увеличивает на 1) значение *счётчика*, при этом все внутренние счётчики сбрасываются (устанавливаются в начальное значение).

`\refstepcounter{счётчик}` — выполняет тоже, что и предыдущая команда, кроме того, ещё декларирует, что текст, генерируемый командой `\thesчётчик`, будет использован в месте употребления команды перекрёстного цитирования `\ref`.

`\newcounter{имя}[счётчик]` задаёт счётчик с именем *имя*. Начальное значение счётчика равно 0, а формат `\theимя` — *arabic*. Необязательный аргумент задаёт внешний *счётчик*. При этом, при наращивании значения этого внешнего *счётчика* (командами `stepcounter` или `refstepcounter`) все подчинённые счётчики (в т. ч. и новый счётчик *имя*) сбрасываются в нуль.

Замечание: эта команда не может присутствовать в файлах, включаемых командой `\include`!

Пример .23. `\thepage \Roman{page}`
79 LXXIX

```
\thesubsection
\setcounter{subsection}{\value{page}}
\thesubsection
\stepcounter{section} \thesubsection
```

5.2. 5.79. 6.0.

```
\fnsymbol{mpfootnote} \arabic{mpfootnote}
\stepcounter{mpfootnote} \fnsymbol{mpfootnote}
\setcounter{mpfootnote}{9} \fnsymbol{mpfootnote}
```

0 * ‡‡

6.1. Определение новых команд

`\newcommand{\имя}[N пар.][знач.]{определение}` — определение новой команды `\имя`. Здесь N *пар.* — число параметров (аргументов), *знач.* — значение по умолчанию. В *определении* в месте появления параметра пишется знак `#` и порядковый номер параметра.

`\renewcommand{\имя}[N пар.][знач.]{определ.}` — переопределение существующей команды.

`\providecommand{\имя}[N пар.][знач.]{определ.}` — определение новой команды; если такая команда существует, то сохраняется её предыдущее значение.

Пример .24. `\providecommand{\No}{\textnumero}`
`\renewcommand{\No}{\textnumero}`

Правило `\No1`: Всякая команда заканчивается
небуквенным символом.

Правило №1: Всякая команда заканчивается небуквенным символом.

6.2. Определение новых окружений

`\newenvironment{имя}[N пар.][знач.]{нач.}{кон.}` — определение нового окружения. *Нач.* и *кон.* — команды, выполняемые в начале и в конце окружения.

`\renewenvironment{имя}[N пар.][знач.]{нач.}{кон.}` — переопределение существующего окружения.

Пример .25. `\newenvironment{Squote}[1][\checkmark]{%`
`\raggedright\rightmargin=5em\begin{itemize}`
`\item[#1]\rmfamily\fontshape{ui}\selectfont`
`{\end{itemize}}`

```
\begin{Squote}
Издательская система логической вёрстки
\end{Squote}
\begin{Squote}[\copyright]
\LaTeXe
\end{Squote}
```

✓ Издательская система логической вёрстки

© L^AT_EX 2_ε

6.3. Определение команд типа «теорема»

`\newtheorem{имя}{Заголовок}[счётчик]` — определяет новую теорему *имя*.
Необязательный аргумент задаёт внешний *счётчик*.

`\newtheorem{имя}[теорема]{Заголовок}` — определение новой теоремы по прототипу (определяемая теорема будет нумероваться так же как и *теорема*-прототип).

Теоремы применяются также, как окружения, при этом возможен один необязательный аргумент.

Пример .26 (использование теорем). `\section[Искусство научных исследований]`
`{Искусство научных исследований}\footnote{Из`
`книги А.~Блоха <<Закон Мерфи>>}}`

```
\newtheorem{rules}{Правило}
\newtheorem{low}{Закон}[section]
```

`\begin{low}[Майерса]Если факты не подтверждают теорию, от них надо избавиться.\end{low}`

`\begin{rules}[точности] Работая над решением задачи, всегда полезно знать ответ.\end{rules}`

`\begin{low}[лаборатории Фетта] Никогда не пытайтесь повторить удачный эксперимент.\end{low}`

Искусство научных исследований¹

Закон 6.1 (Майерса). *Если факты не подтверждают теорию, от них надо избавиться.*

Правило 1 (точности). *Работая над решением задачи, всегда полезно знать ответ.*

Закон 6.2 (лаборатории Фетта). *Никогда не пытайтесь повторить удачный эксперимент.*

¹Из книги А. Блоха «Закон Мерфи»

Глава 5

Печатный документ

1. Структура документа

1.1. Преамбула

Преамбула — вводная часть исходного файла, предшествующая самому документу.

`\documentclass[опции]{класс}[дата]` — определяет *класс* документа, *опции* — дополнительные параметры, *дата* указывает дату выпуска наиболее старой версии класса, пригодного для компиляции (записывается в формате гггг/мм/дд).

`\usepackage[опции]{пакет}[дата]` — подключает дополнительные *пакеты*. Эта команда действует аналогично предыдущей.

1.2. Титульный лист

Для автоматической генерации титульного листа используется декларация `\maketitle`. Информация для титульного листа берётся из команд:

- `\title{Заголовок}` — обязательна;
- `\author{Автор}` — обязательна, можно между «авторами» вставлять декларацию `\and`;
- `\date{дата}` — если не указать, то ставит текущую дату.

В аргументах всех этих команд можно использовать команду `\thanks{примечания}`.

Для составления титульного листа вручную используется окружение `titlepage`, которое создаёт новую страницу, устанавливает счётчик страниц на 1, но не печатает номер страницы.

1.3. Аннотация (abstract)

Для аннотации используется окружение `abstract`, оно начинает новую страницу, если включена опция документа `titlepage`. Текст внутри окружения выделяется дополнительными отступами слева и справа.

1.4. Секционирование (рубрикация)

Таблица 5.1.

Стандартные команды рубрикации

Команда	Рубрика
<code>\part[альт.загол.]{загол.}</code>	Часть
<code>\chapter[альт.загол.]{загол.}</code>	Глава
<code>\section[альт.загол.]{загол.}</code>	Раздел
<code>\subsection[альт.загол.]{загол.}</code>	Подраздел
<code>\subsubsection[альт.загол.]{загол.}</code>	Подподраздел
<code>\paragraph[альт.загол.]{загол.}</code>	Параграф
<code>\subparagraph[альт.загол.]{загол.}</code>	Подпараграф

Обязательный аргумент *загол.* — заголовок рубрики, необязательный *альт.загол.* — альтернативный заголовок. Если задан альтернативный заголовок, то он будет фигурировать в содержании и в колонтитулах.

Все эти рубрики имеют разновидность со звёздочкой. Команда со звёздочкой (*) создаёт рубрику без номера и не имеет дополнительного аргумента.

Аргумент, который передаётся в содержание (колонтитул), является *подвижным*, поэтому *неустойчивые команды*, используемые в таких аргументах, должны быть защищены с помощью декларации `\protect`.

Нумерация рубрик устроена иерархически (за исключением `\part`, эта рубрика необязательна и никак не влияет на нумерацию глав). В классах **article** и **proc** не определена рубрика `\chapter`, для них самой старшей и обязательной рубрикой является `\section`. Класс **letter** вообще не имеет рубрик.

Счётчик `secnumdepth` определяет уровень самого младшего раздела, использующего нумерацию.

Декларация `\appendix` обозначает приложения. Она изменяет способ нумерации, а при использовании рубрики `\chapter` пишет слово «Приложение» вместо «Глава».

Ключевые слова рубрик задаются следующими командами-логосами: `\abstractname`, `\partname`, `\chaptername`, `\appendixname`, которые должны переопределяться в файле русификации.

1.5. Оглавление (содержание)

Оглавление, списки рисунков и таблиц делают соответствующие декларации:

- `\tableofcontents` — оглавление;
- `\listoffigures` — список рисунков;
- `\listoftables` — список таблиц.

При этом создаются служебные файлы `.toc`, `.lof` и `.lot`, соответственно.

Служебные файлы создаются/обновляются ЛАТЭХом по мере необходимости, если только, в преамбуле не стоит декларация `\nofiles`. Обычно эта декларация используется на самом последнем этапе подготовки документа, для внесения последних поправок (если в этом есть необходимость).

Для «ручной» коррекции содержаний используются следующие команды:

- `\addcontentsline{файл}{формат}{запись}` — добавление *записи* в соответствующий *файл* в указанном *формате*. Форматы могут быть следующими:
 - для файлов `.toc` — *part*, *chapter*, *section*, *subsection*, *subsubsection*, *paragraph*, *subparagraph*;
 - для файлов `.lof` — *figure*;
 - для файлов `.lot` — *table*.
- `\addtocontents{файл}{запись}` — добавление *записи* в соответствующий *файл*. Здесь в качестве *записи* можно использовать любой текст в т. ч. и форматизирующие команды.

Для внесения записи с произвольным номером (например, номером раздела) можно использовать команду `\numberline{номер}{запись}` с предварительной командой `\protect`.

Для составления оглавления полезно использовать счётчик `tocdepth`, который определяет уровень самого младшего раздела, входящего в оглавление.

Замечание: *Запись* в приведённых командах является подвижным аргументом!

1.6. Библиография (список литературы)

Список литературы формируется с помощью окружения `thebibliography`:

```
\begin{thebibliography}{текст}
  \bibitem[метка]{имя} данные
  \bibitem[метка]{имя} данные
  .
  .
  .
  \bibitem[метка]{имя} данные
\end{thebibliography}
```

Обязательный аргумент *текст* служит для задания левого отступа, равного ширине этого *текста*.

Если *метка* в аргументе `\bibitem` не задана, то записи списка нумеруются порядковыми номерами.

Для задания *имени* литературного источника можно использовать любые символы, кроме ‘,’.

При записи данных литературного источника можно использовать декларацию `\newblock`, которую следует вставлять между логическими блоками. При использовании стандартных классов в этом месте будет добавлен дополнительный пробел. При наличии опции класса `openbib` каждый блок будет напечатан с новой строки, а строки внутри блока наделены дополнительным левым отступом на величину `\bibindent`.

Цитирование литературы в тексте производится командой `\cite[текст]{имя}`, при этом, если указан *текст*, то он вставляется, через запятую, после индекса источника. Можно в одной команде указывать несколько источников, перечисляя их через запятую.

Для цитирования литературы используется механизм *перекрёстного цитирования* (файл `.aux`).

Для более удобной работы с библиографией существует утилита ВивТ_ЕX.

2. Механизм перекрёстного цитирования

Для того, чтобы сослаться на ту или иную часть текста, необходимо поставить *метку* в этой части с помощью команды `\label{метка}`.

Для ссылок существуют две стандартные команды:

- `\ref{метка}` — вставляет значение счётчика изменённого перед соответствующей командой `\label`.
- `\pageref{метка}` — вставляет номер страницы, на которой стоит соответствующая команда `\label`.

Метку надо ставить *сразу же после помечаемого объекта!*

```
Пример .27 (содержимое .aux файла). \@writefile{lot}{\contentsline {table}{
  \numberline {1}{\ignorespaces Стандартные
  команды рубрикации}}{2}{table.0.1}}
\newlabel{sect}{{1}{2}{Секционирование (рубрикация)
  \relax }{table.0.1}}
\bibcite{latexe}{\hyper@link[Cite]{}{\hyper@hash
  latexe}{1}}
\bibcite{latex}{\hyper@link[Cite]{}{\hyper@hash
  latex}{2}}
\newlabel{ex4}{{}{1}{\relax }{task.0..4}}
\newlabel{step}{{{\bf 8.}}{2}{Структура документа
  \relax }{enumi.0.8}}
```

3. Большие документы

3.1. Условная компиляция

Часть документа, для которой выполняется *условная компиляция* надо окружить командными скобками `if`: `\iftrue` ... `\fi` или `\iffalse` ... `\fi`.

Первые командные скобки разрешают компиляцию, вторые — запрещают.

Можно внутри скобок `if` использовать «оператор» `\else`, который имеет тот-же смысл (инверсия), что и в программировании.

3.2. Включение файлов

Включать файлы в основной документ можно двумя способами.

1. Можно использовать команду `\input{файл}`.

Имя *файла* можно указать как локальное или как полное, при этом, для описания пути используется слэш `'/'`. Если файл имеет расширение `.tex`, то его можно не писать.

Действие этой команды аналогично тому, как если-бы вместо команды было написано содержимое файла.

2. Другой способ заключается в том, что документ разбивается на логически законченные части, которые записываются в отдельных файлах.

Создаётся *главный документ*, который включает в себя преамбулу, окружение **document**, при необходимости — команды генерации содержаний и команды `\include{файл}` для каждого используемого файла.

В преамбулу помещается команда `\includeonly{список файлов}`, в которой перечисляются те файлы, которые должны быть откомпилированы.

Содержимое каждого откомпилированного файла будет начинаться с новой страницы, при этом, сохранится вся необходимая информация для перекрёстного цитирования.

Пример .28 (использование метода “include”). Пример использования метода “include”:

```
\documentclass[a4paper,openbib]{report}
\usepackage[cp1251]{inputenc}
\usepackage[russian]{babel}
  \includeonly{%
%tex/format,
%tex/ex3,
tex/doc,
%tex/ex4,
  }
\begin{document}
```

```

\tableofcontents
\include{tex/format}
\include{tex/ex3}
\include{tex/doc}
\include{tex/ex4}
\end{document}

```

4. Стиль документа

4.1. Стиль страницы

Страница состоит из *верхнего колонтитула*, *тела страницы* и *нижнего колонтитула*. Для задания стиля используются декларации:

- `\pagestyle{стиль}` — распространяется на весь текст, начиная с текущей страницы;
- `\thispagestyle{стиль}` — распространяется только на текущую страницу.

Стиль может иметь одно из 4 значений:

empty — пустые колонтитулы (по умолчанию использует класс **letter**);

plain — верхний колонтитул пустой, в нижнем колонтитуле — номер страницы (по умолчанию использует все классы, кроме **book** и **letter**);

headings — нижний колонтитул пустой, а в верхнем колонтитуле — номер страницы и другая информация, в зависимости от класса документа, например, название текущего раздела (по умолчанию использует класс **book**);

myheadings — нижний колонтитул пустой, а в верхний колонтитул записывается информация, определяемая командами `\markboth{лев.}{прав.}` и `\markright{прав.}`.

Стиль номера страницы задаётся командой `\pagenumbering{стиль}`, здесь *стиль* может принимать, уже знакомые по счётчикам, значения: *arabic*, *roman*, *Roman*, *alph* и *Alph*.

Формат страницы

Формат страницы определяется многими параметрами — **командными длинами**. Для удобства, можно подключить пакет `layout`, который по команде `\layout` выводит макет страницы с указанием всех длин и их значений.

Далее приводится перечень этих параметров.

`\paperheight` — высота листа бумаги.

`\paperwidth` — ширина листа бумаги.

`\topmargin` — расстояние между верхним краем листа и верхним колонтитулом минус 1 дюйм.

`\oddsidemargin` — расстояние между левым краем листа и левым краем текста на правой странице минус 1 дюйм.

`\evensidemargin` — то же самое для левой страницы.

`\textheight` — высота текста, т. е. вертикальный размер тела страницы (без колонтитулов).

`\textwidth` — ширина текста, т. е. горизонтальный размер колонтитулов и тела страницы. Переопределяется внутри процедур, изменяющих правую и/или левую границы текста.

`\linewidth` — ширина строки; равна значению `\textwidth` за исключением строк внутри процедур форматирования абзацев, таких как `quote` или `itemize`. Значение `\linewidth` не должно изменяться командами, изменяющими длину.

`\topskip` — минимальное расстояние между верхним краем тела страницы и базисной линией первой строки текста. Соответствует `\baselineskip` для первой строки страницы.

`\headheight` — высота верхнего колонтитула.

`\headsep` — вертикальное расстояние между верхним колонтитулом и телом текстовой страницы.

`\footheight` — высота нижнего колонтитула.

`\footskip` — вертикальное расстояние между нижним краем текста и нижним краем нижнего колонтитула; измеряется между базисными линиями последних строк текста и нижнего колонтитула.

`\parindent` — ширина отступа в начале абзаца. Внутри `\parbox` устанавливается равной нулю. Может быть изменена в любом месте.

`\parskip` — дополнительный вертикальный пробел, вставляемый перед абзацем.

`\marginparwidth` — ширина заметок на полях.

`\marginparsep` — горизонтальное расстояние между внешним краем текста и заметкой на полях.

Эффективные значения `\baselinestretch` для различных размеров шрифта

<i>интервал</i>	10 pt	11 pt	12 pt
полтора	1,25	1,213	1,241
два	1,667	1,618	1,655

`\baselineskip` — минимальное расстояние между базисными линиями последовательных строк текста. Расстояние между некоторыми строками может быть больше, если они содержат высокие объекты. Значение `\baselineskip` устанавливается декларациями, изменяющими размер шрифта.

`\baselinestretch` — действительное число, равное величине межстрочного интервала; по умолчанию равно 1. Значение `\baselinestretch` изменяется командой `\renewcommand`.

Пример .29 (задание `\baselinestretch`). Например, поставленная в преамбулу декларация `\renewcommand{\baselinestretch}{1.5}` приведёт к тому, что весь печатный документ будет напечатан через 1,5 интервала + `\baselineskip` текущего шрифта.

Пакет `doubleSPACE` *Стефана Пейджса* позволяет изменять интерлиньяж локально в k -раз посредством окружения `spacing{k}`.

Пакет `setspace` *Горффри Тобина* основан на пакете `doubleSPACE`, он автоматически рассчитывает `\baselinestretch` для текущего шрифта.

В пакете определены команды `\singleSpacing`, `\onehalfSpacing` и `\doubleSpacing` — для глобального изменения интерлиньяжа, а также, соответствующие им окружения: `singleSpace`, `onehalfSpace`, и `doubleSpace` — для локального изменения интерлиньяжа.

Текст можно разместить в две колонки. Переход в двухколоночный режим осуществляется по команде `\twoColumn[текст]`. При этом начинается новая страница, если имеется необязательный аргумент *текст*, то он будет написан сверху страницы в одноколоночном режиме.

Переход в одноколоночный режим выполняется по команде `\oneColumn`.

Если основная часть документа должна быть свёрстана в две колонки, то применяется опция документа `twoColumn`. В классе `proc` определён только двухколоночный режим.

4.2. Стандартные классы

На сегодняшний день в ЛАТЭХе существует 6 стандартных классов:

report — *отчёт*, предназначен для написания небольших документов.

article — *статья*, предназначен для написания небольших документов, разделы начинаются с `\section`, хотя раздел `\part` определён.

proc — *доклад*, предназначен для написания небольших документов (статья в журнал, в сборник трудов), разделы начинаются с `\section`, в отличии от **article**, печатается только в две колонки.

book — *книга*, предназначен для написания больших документов (книга, журнал, сборник трудов). Отличается от **report** оформлением страниц и тремя дополнительными командами: `\frontmatter` — для вводной части, `\mainmatter` — для основной части и `\backmatter` — для заключительной части.

slides — *слайд*, предназначен для подготовки презентаций, слайд-шоу. Отличается увеличенными шрифтами, имеет свою дополнительную опцию и набор команд и окружений для подготовки и печати слайдов, оверлеев и заметок.

letter — *письмо*, предназначен для написания разного рода деловых писем. Имеет большой набор своих команд а многие стандартные команды отсутствуют.

Опции стандартных классов

10pt | **11pt** | **12pt** — устанавливают размер основного шрифта. По умолчанию используется 10pt.

*Эти опции отсутствуют в классе **slides**.*

letterpaper | **legalpaper** | **executivpaper**

a4paper | **a5paper** | **b5paper** — устанавливают размер листа бумаги:

Letter	8,5 д	× 11 д
Legal	8,5 д	× 14 д
Executive	7,25 д	× 10,5 д
A4	210 мм	× 297 мм
A5	148 мм	× 210 мм
B5	176 мм	× 250 мм

По умолчанию установлена опция **letterpaper**.

*Опции **a5paper** и **b5paper** отсутствуют в классе **proc**.*

landscape — устанавливает альбомное расположение листа (ландшафт), при этом ширина и высота листа меняются местами.

final | **draft** — устанавливают режим печати (чистой или черновой). В черновом режиме строки, выходящие за правый край помечаются чёрными маркерами. По умолчанию используется опция **final**.

oneside | **twoside** — устанавливают формат документа для односторонней / двусторонней печати. При двусторонней печати форматы чётной и нечётной (левой и правой) страниц отличаются. При односторонней печати все страницы считаются правыми. По умолчанию используется опция **oneside** во всех классах, кроме **book**.

*Эти опции отсутствуют в классе **slides**.*

openright | **openany** — устанавливают режим печати глав, соответственно, на правой странице (используется по умолчанию в классе **book**) или на любой странице (используется по умолчанию в классе **report**).

В других классах эти опции отсутствуют.

onecolumn | **twocolumn** — устанавливают одноколоночный/двухколоночный формат документа. По умолчанию **onecolumn** используется для всех классов, кроме **proc**.

*Эти опции отсутствуют в классах **slides** и **letter**, а в классе **proc** не поддерживается опция **onecolumn**.*

titlepage | **notitlepage** — устанавливают режим печати титульной страницы (сгенерированной автоматически, декларацией `\maketitle`) и аннотации: на отдельных страницах (**titlepage**) или перед текстом, на той-же странице. По умолчанию **titlepage** используется для всех классов, кроме **article**.

*Эти опции отсутствуют в классе **letter**, а в классе **proc** не поддерживается опция **titlepage**.*

openbib — устанавливает режим форматирования списка литературы в *открытом стиле*, когда элементы записи, написанные в виде отдельных блоков печатаются с новой строки.

*Эта опция отсутствует в классах **slides** и **letter**.*

leqno — устанавливает формат печати номеров формул слева, а не справа.

fleqn — устанавливает выравнивание формул не по центру, а по левому краю.

Глава 6

Математика в L^AT_EX'e

1. Основные понятия

Математические формулы записываются в исходном файле в виде логических структур, а L^AT_EX уже строит формулу в соответствии со всеми требованиями и стандартами.

1.1. Математические моды (режимы)

Математический режим можно включить несколькими способами, при этом, возможны два варианта расположения математических формул: Одни и те же математические формулы выглядят по-разному в этих двух режимах.

- *Внутри строки (in-line)*, когда математический текст является частью текстовой строки. Этот режим задаётся тремя способами: с помощью окружения `math` (`\begin{math} . . . \end{math}`), или `\(. . . \)`, или `$. . . $` (последняя команда происходит из T_EXa). В этом режиме формулы переносятся автоматически, как текст.
- В явном виде (**display**), когда математический текст печатается на отдельной строке (**вынесенная формула**). Этот режим тоже можно задать разными способами: с помощью окружения `displaymath` (`\begin{displaymath} . . . \end{displaymath}`), или `\[. . . \]`, или `$$. . . $$` (эта команда тоже происходит из T_EXa).

Кроме того, для вынесенных формул имеется специальное окружение. Окружение `equation` формирует вынесенную формулу и автоматически её нумерует, вариант `equation*` отменяет нумерацию. *В вынесенном режиме формулы не переносятся автоматически.*

При наборе сложных формул рекомендуется пользоваться $\mathcal{A}\mathcal{M}\mathcal{S}$ -L^AT_EX'ом (`amsmath`) (см. раздел 4.2.). Тогда, кроме обычного **equation** доступны окружения:

- **multline** — для длинных выражений;

- **align (flalign, alignat)** — для выравнивания выражений;
- **gather** — для центрирования выражения;
- **split** — для произвольного выравнивания (используется внутри другого окружения, обычно **equation**).

Кроме того, этот пакет вводит ряд новых, более удобных команд, а также решает некоторые проблемы с расположением отдельных элементов в математической моде.

Следует различать обычный (t) и математический (t) текст.

В свою очередь, в математических выражениях единицы измерения, химические элементы, в русскоязычном тексте ещё и запятая (как разделитель дробной части), пишутся в текстовой моде.

Для того, чтобы простой текст внутри математической формулы выглядел правильно, можно использовать команду `\mbox{текст}`, а лучше `\text{текст}` из пакета **amsmath**.

Пример .30 (математический и текстовый режимы).

$$S_5 = 1,09 \cdot 10^{-18} \left(\frac{10^5 \text{ ГэВ}}{E} \right)^{3,02}$$

```


$$S_5 = 1,09 \cdot 10^{-18} \left( \frac{10^5 \text{ ГэВ}}{E} \right)^{3,02}$$


```

$t = 1,8$ км в. э.	<code>\$t=1\$,8~км~в.~э.</code>
$B_{\text{кон}} = 100$ нТл	<code>\$B_{\text{кон}}=100\text{ нТл}\$</code>
$n_0 = 100$ см ⁻³	<code>\$n_0=100\$ см⁻³\$</code>

1.2. Пробелы в математических формулах

В математических текстах, как в тех, которые располагаются в текстовой строке, так и в тех, которые печатаются в красную строку, межсимвольные пробелы устанавливаются только ЛАТЭХ'ом. Добавление лишних пробелов во входном файле никак не влияет на расположение символов в математических формулах.

Для ручного управления пробелами существуют специальные команды, перечисленные в табл. 6.1.

Первые три пробела и малый пробел можно использовать и в обычном тексте, остальные — только в математическом.

Для точной настройки горизонтальных пробелов в математике имеется команда `\mspace{длина}`, где *длина* измеряется *только* в математических единицах ($1\text{mu} = 1/18\text{em}$).

Пробелы внутри математических текстов

Название пробела	Команда	←Размер→
Двойной математический	<code>\qquad</code>	$\quad\quad$
Математический	<code>\quad</code>	\quad
Межсимвольный	<code>\quad</code>	\quad
Большой	<code>\; </code>	$\; $
Средний	<code>\: </code>	$\: $
Малый	<code>\, </code>	$\, $
Отрицательный малый	<code>\! </code>	$\! $
Отрицательный средний	<code>\negmedspace</code>	\negmedspace
Отрицательный большой	<code>\negthickspace</code>	\negthickspace

2. Алфавит математики

Литеры, используемые в формулах можно разделить на два класса: *алфавитно-цифровые символы* и *математические*.

Терминальные (клавиатурные) символы `+ - / * = ' | < > ()` в математических формулах выглядят следующим образом: `+ - /* = ' | < > ()`.

2.1. Математические акценты

Акценты, используемые в текстовой и строковой моде не работают в математической моде. Их, конечно, можно использовать в аргументах команд `\mbox{}` и `\text{}`). Для математики существуют свои акценты, которые задаются специальными командами, действующими только в математической моде.

Математические акценты

Команда	Пример	
<code>\hat</code>	<code>\hat a</code>	\hat{a}
<code>\check</code>	<code>\check a</code>	\check{a}
<code>\breve</code>	<code>\breve{a}</code>	\breve{a}
<code>\acute</code>	<code>\acute{a}</code>	\acute{a}
<code>\grave</code>	<code>\grave{a}</code>	\grave{a}
<code>\tilde</code>	<code>\tilde{a}</code>	\tilde{a}
<code>\bar</code>	<code>\bar{a}</code>	\bar{a}
<code>\vec</code>	<code>\vec{a}</code>	\vec{a}
<code>\dot</code>	<code>\dot{a}</code>	\dot{a}
<code>\ddot</code>	<code>\ddot{a}</code>	\ddot{a}
<code>\ddd</code>	<code>\ddd{a}</code>	\ddd{a}
<code>\dddd</code>	<code>\dddd{a}</code>	\dddd{a}

Последние две команды доступны в пакете **amsmath**.

Имеется два растяжимых акцента, — `\widehat{}` и `\widetilde{}`, их ширина подбирается автоматически, от одного до трёх символов.

Пример .31. $\widehat{ABC} = \widetilde{AB} + \widetilde{BC}$ `\widehat{ABC}=\widetilde{AB}+\widetilde{BC}`.

Пакет **amxtra** вводит команды, задающие акценты как верхние индексы имена таких команд образованы добавлением приставки “sp”:

Таблица 6.3.

Математические акценты

Команда	Пример	
<code>\sphat</code>	<code>(AmSxtra)\sphat</code>	$(AmSxtra)^{\hat{}}$
<code>\spcheck</code>	<code>(AmSxtra)\spcheck</code>	$(AmSxtra)^{\check{}}$
<code>\spbrev</code>	<code>(AmSxtra)\spbrev</code>	$(AmSxtra)^{\breve{}}$
<code>\sptilde</code>	<code>(AmSxtra)\sptilde</code>	$(AmSxtra)^{\tilde{}}$
<code>\spdot</code>	<code>(AmSxtra)\spdot</code>	$(AmSxtra)^{\cdot}$
<code>\spddot</code>	<code>(AmSxtra)\spddot</code>	$(AmSxtra)^{\ddot{}}$
<code>\spddd</code>	<code>(AmSxtra)\spddd</code>	$(AmSxtra)^{\cdots}$

2.2. Греческие буквы

Имена команд для букв греческого алфавита составлены из их английских транскрипций.

В этом наборе отсутствует буква «омикрон», т. к. она совпадает с латинской буквой ‘o’. По той-же причине отсутствуют специальные команды для некоторых прописных букв.

По умолчанию, прописные буквы пишутся прямым шрифтом, а строчные — курсивом.

Некоторые строчные буквы имеют два варианта написания.

2.3. Бинарные операторы

Л^AT_EX окружает эти символы дополнительными пробелами (за исключением индексов). Команда `\not` перечёркивает символ (любой).

2.4. Символы сравнения

2.5. Большие операторы и символы переменного размера

Символы, начинающиеся с **big** имеют меньшие аналоги среди знаков бинарных операций, они обычно пишутся с индексами.

Такие символы, как знаки суммы, произведения, интегралы меняют свой размер в зависимости от типа математической моды, при этом пределы (индексы) автоматически размещаются справа от символа или сверху и снизу.

Для ручного управления размещением пределов имеются две команды:

Таблица 6.4.

Команды для букв греческого алфавита

α	<code>\alpha</code>	β	<code>\beta</code>	γ	<code>\gamma</code>
δ	<code>\delta</code>	ϵ	<code>\epsilon</code>	ε	<code>\varepsilon</code>
ζ	<code>\zeta</code>	η	<code>\eta</code>	θ	<code>\theta</code>
ϑ	<code>\vartheta</code>	ι	<code>\iota</code>	κ	<code>\kappa</code>
\varkappa	<code>\varkappa</code> ¹	λ	<code>\lambda</code>	μ	<code>\mu</code>
ν	<code>\nu</code>	ξ	<code>\xi</code>	o	<code>o</code>
π	<code>\pi</code>	ϖ	<code>\varpi</code>	ρ	<code>\rho</code>
ϱ	<code>\varrho</code>	σ	<code>\sigma</code>	ς	<code>\varsigma</code>
τ	<code>\tau</code>	υ	<code>\upsilon</code>	ϕ	<code>\phi</code>
φ	<code>\varphi</code>	χ	<code>\chi</code>	ψ	<code>\psi</code>
ω	<code>\omega</code>	Γ	<code>\Gamma</code>	Δ	<code>\Delta</code>
Θ	<code>\Theta</code>	Λ	<code>\Lambda</code>	Ξ	<code>\Xi</code>
Π	<code>\Pi</code>	Σ	<code>\Sigma</code>	Υ	<code>\Upsilon</code>
Φ	<code>\Phi</code>	Ψ	<code>\Psi</code>	Ω	<code>\Omega</code>

Таблица 6.5.

Команды для бинарных операторов

\pm	<code>\pm</code>	\cap	<code>\cap</code>	\vee	<code>\vee</code>
\mp	<code>\mp</code>	\cup	<code>\cup</code>	\wedge	<code>\wedge</code>
\setminus	<code>\setminus</code>	\uplus	<code>\uplus</code>	\oplus	<code>\oplus</code>
\cdot	<code>\cdot</code>	\sqcap	<code>\sqcap</code>	\ominus	<code>\ominus</code>
\times	<code>\times</code>	\sqcup	<code>\sqcup</code>	\otimes	<code>\otimes</code>
$*$	<code>\ast</code>	\triangleleft	<code>\triangleleft</code>	\oslash	<code>\oslash</code>
\star	<code>\star</code>	\triangleright	<code>\triangleright</code>	\odot	<code>\odot</code>
\diamond	<code>\diamond</code>	\wr	<code>\wr</code>	\dagger	<code>\dagger</code>
\circ	<code>\circ</code>	\bigcirc	<code>\bigcirc</code>	\ddagger	<code>\ddagger</code>
\bullet	<code>\bullet</code>	\triangleup	<code>\triangleup</code>	\amalg	<code>\amalg</code>
\div	<code>\div</code>	\triangledown	<code>\triangledown</code>		

Таблица 6.6.

Команды для символов сравнения (отношений)

\leq	<code>\leq</code>	\geq	<code>\geq</code>	\equiv	<code>\equiv</code>
\prec	<code>\prec</code>	\succ	<code>\succ</code>	\sim	<code>\sim</code>
\preceq	<code>\preceq</code>	\succeq	<code>\succeq</code>	\simeq	<code>\simeq</code>
\ll	<code>\ll</code>	\gg	<code>\gg</code>	\asymp	<code>\asymp</code>
\subset	<code>\subset</code>	\supset	<code>\supset</code>	\approx	<code>\approx</code>
\subseteq	<code>\subseteq</code>	\supseteq	<code>\supseteq</code>	\cong	<code>\cong</code>
\sqsubset	<code>\sqsubset</code>	\sqsupseteq	<code>\sqsupseteq</code>	\bowtie	<code>\bowtie</code>
\in	<code>\in</code>	\ni	<code>\ni</code>	\propto	<code>\propto</code>
\vdash	<code>\vdash</code>	\dashv	<code>\dashv</code>	\models	<code>\models</code>
\smile	<code>\smile</code>	\mid	<code>\mid</code>	\doteq	<code>\doteq</code>
\frown	<code>\frown</code>	\parallel	<code>\parallel</code>	\perp	<code>\perp</code>

Таблица 6.7.

Команды для больших операторов

\sum	<code>\sum</code>	\bigcap	<code>\bigcap</code>	\bigodot	<code>\bigodot</code>
\prod	<code>\prod</code>	\bigcup	<code>\bigcup</code>	\bigotimes	<code>\bigotimes</code>
\coprod	<code>\coprod</code>	\bigsqcup	<code>\bigsqcup</code>	\bigoplus	<code>\bigoplus</code>
\int	<code>\int</code>	\bigvee	<code>\bigvee</code>	\biguplus	<code>\biguplus</code>
\oint	<code>\oint</code>	\bigwedge	<code>\bigwedge</code>	\iint	<code>\iint¹</code>
\iiint	<code>\iiint^a</code>	\iiint	<code>\iiint^a</code>	$\int \cdots \int$	<code>\int \cdots \int^a</code>

Команды для символов-разделителей

(([[{	\{
))]]	}	\}
<	\langle	/	/		
>	\rangle	\	\backslash		\
⌊	\lfloor	⌈	\lceil		
⌋	\rfloor	⌉	\rceil		
↑	\uparrow	↓	\downarrow	↕	\updownarrow
⇑	\Uparrow	⇓	\Downarrow	⇕	\Updownarrow

- `\limits` — расположение пределов сверху и снизу от символа;
- `\nolimits` — расположение пределов справа от символа.

Для глобального управления размещением пределов, в пакете **amsmath** определены опции:

sumlimits | **nosumlimits** — для управления пределами в суммах,

intlimits | **nointlimits** — для управления пределами в интегралах.

2.6. Разделители

В качестве разделителей в математике используются различные скобки, уголки, вертикальные стрелки.

Для автоматического управления высотой разделителей следует использовать команды `\left` (для левой скобки) и `\right` (для правой) перед соответствующим разделителем. Для непарных разделителей можно использовать одну из этих команд, однако, надо иметь в виду, что в каждой строке количество команд `\left` и `\right` должно совпадать.

Для этого можно использовать вместо разделителя точку: команды `\left.` и `\right.` ничего не пишут.

Пример .32 (разделители в сложных формулах).

$$H_c = \frac{n_1!n_2!}{n_1+n_2} \sum_i \left[\binom{n_1}{i} \binom{n_2}{n_1+i} + \binom{n_1-1}{i} \binom{n_2-1}{n_1+i} \right].$$

```
\begin{multline*}
H_c=\frac{n_1!\,n_2!}{n_1+n_2}\sum_i \left[
\binom{n_1}{i}\binom{n_2}{n_1+i}+\right.\right.\end{pre}
```

Команды, увеличивающие разделители

Слева	В центре	Справа	Символ
<code>\bigl</code>	<code>\bigm</code>	<code>\bigr</code>	<code>\big</code>
<code>\Bigl</code>	<code>\Bigm</code>	<code>\Bigr</code>	<code>\Big</code>
<code>\biggl</code>	<code>\biggm</code>	<code>\biggr</code>	<code>\bigg</code>
<code>\Biggl</code>	<code>\Biggm</code>	<code>\Biggr</code>	<code>\Bigg</code>

```
+\left.\binom{n_1-1}{i}\binom{n_2-1}{n_1+i}
\right].
\end{multline*}
```

Иногда автоматический выбор высоты не удовлетворяет авторским замыслам. Для ручного управления высотой разделителей используют команды из табл. 6.9.

Здесь записаны команды в порядке возрастания. Команды из второй колонки относятся к непарным разделителям, а команды из последней колонки рекомендуется применять к символам (при этом окружающие пробелы будут несколько меньше, чем для непарного разделителя).

Пример .33 (большие разделители в формулах).

$$\left\{ \left[\frac{A}{B} \right] = C \right\}; \quad A \Big/ B = C; \quad A \Bigg/ B = C;$$

```
$$ \Biggl\{ \Bigl[ \frac{A}{B} \Bigr] = C \Biggr\};
\quad A \Bigg/ B = C; \quad \quad A \Bigg/ B = C; $$
```

2.7. Стрелки

Вертикальные стрелки автоматически изменяют свою высоту, когда используются в качестве разделителей.

2.8. Функции

Функции (их ещё называют логарифмоподобными) принято писать в математике прямым шрифтом.

Некоторые функции могут иметь индексы.

Если требуется ввести новую функцию, то лучше это сделать с помощью команд пакета `amsopn` `\DeclareMathOperator{\имя}{написание}` и `\DeclareMathOperator*{\имя}{написание}`. Здесь *имя* и *написание* имеют тот-же смысл, что и при определении новых команд. Определять новые функции следует в преамбуле.

Таблица 6.10.

Команды для символов-стрелок

\leftarrow	<code>\leftarrow</code>	\Leftarrow	<code>\Leftarrow</code>
\rightarrow	<code>\rightarrow</code>	\Rightarrow	<code>\Rightarrow</code>
\longleftarrow	<code>\longleftarrow</code>	\Longleftarrow	<code>\Longleftarrow</code>
\longrightarrow	<code>\longrightarrow</code>	\Longrightarrow	<code>\Longrightarrow</code>
\uparrow	<code>\uparrow</code>	\Uparrow	<code>\Uparrow</code>
\downarrow	<code>\downarrow</code>	\Downarrow	<code>\Downarrow</code>
\leftrightarrow	<code>\leftrightarrow</code>	\Leftrightarrow	<code>\Leftrightarrow</code>
\longleftrightarrow	<code>\longleftrightarrow</code>	\Longleftrightarrow	<code>\Longleftrightarrow</code>
\updownarrow	<code>\updownarrow</code>	\Updownarrow	<code>\Updownarrow</code>
\nearrow	<code>\nearrow</code>	\searrow	<code>\searrow</code>
\swarrow	<code>\swarrow</code>	\nwarrow	<code>\nwarrow</code>
\mapsto	<code>\mapsto</code>	\longmapsto	<code>\longmapsto</code>
\hookrightarrow	<code>\hookrightarrow</code>	\hookrightarrow	<code>\hookrightarrow</code>
\leftharpoonup	<code>\leftharpoonup</code>	\rightharpoonup	<code>\rightharpoonup</code>
\rightharpoondown	<code>\rightharpoondown</code>	\leftharpoondown	<code>\leftharpoondown</code>
\rightleftharpoons	<code>\rightleftharpoons</code>		

Таблица 6.11.

Команды для функций

sin	<code>\sin</code>	arcsin	<code>\arcsin</code>	sec	<code>\sec</code>
cos	<code>\cos</code>	arccos	<code>\arccos</code>	csc	<code>\csc</code>
tan	<code>\tan</code>	arctan	<code>\arctan</code>	cot	<code>\cot</code>
sinh	<code>\sinh</code>	cosh	<code>\cosh</code>	max	<code>\max</code>
tanh	<code>\tanh</code>	coth	<code>\coth</code>	min	<code>\min</code>
log	<code>\log</code>	ln	<code>\ln</code>	lg	<code>\lg</code>
exp	<code>\exp</code>	sup	<code>\sup</code>	inf	<code>\inf</code>
lim	<code>\lim</code>	lim sup	<code>\limsup</code>	lim inf	<code>\liminf</code>
arg	<code>\arg</code>	ker	<code>\ker</code>	hom	<code>\hom</code>
det	<code>\det</code>	dim	<code>\dim</code>	gcd	<code>\gcd</code>
deg	<code>\deg</code>	Pr	<code>\Pr</code>		

Команды для разных математических символов

\aleph	<code>\aleph</code>	\prime	<code>\prime</code>	\forall	<code>\forall</code>
\hbar	<code>\hbar</code>	\emptyset	<code>\emptyset</code>	\exists	<code>\exists</code>
\imath	<code>\imath</code>	∇	<code>\nabla</code>	\neg	<code>\neg</code>
\jmath	<code>\jmath</code>	$\sqrt{\quad}$	<code>\surd</code>	\flat	<code>\flat</code>
ℓ	<code>\ell</code>	\top	<code>\top</code>	\natural	<code>\natural</code>
\wp	<code>\wp</code>	\perp	<code>\bot</code>	\sharp	<code>\sharp</code>
\Re	<code>\Re</code>	\parallel	<code>\ </code>	\clubsuit	<code>\clubsuit</code>
\Im	<code>\Im</code>	\angle	<code>\angle</code>	\diamondsuit	<code>\diamondsuit</code>
∂	<code>\partial</code>	\triangle	<code>\triangle</code>	\heartsuit	<code>\heartsuit</code>
∞	<code>\infty</code>	\backslash	<code>\backslash</code>	\spadesuit	<code>\spadesuit</code>

Пример .34 (определение пользовательских функций). `\DeclareMathOperator{\tg}{tg}`
`\DeclareMathOperator*\{Lim}{Lim}`
`$$\Lim_{x\rightarrow k\pi}\tg^2x=0,`
`\forall k=0,1,\dotsc$$`

$$\lim_{x \rightarrow k\pi} \operatorname{tg}^2 x = 0, \forall k = 0, 1, \dots$$

2.9. Прочие символы

В этой группе собраны все остальные символы. Если этих символов оказалось недостаточно для записи выражения, можно воспользоваться символами $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ 'а.

3. Основные структуры

3.1. Верхние и нижние индексы

Символы ‘ $_$ ’ и ‘ $\^$ ’ обозначают, что следующий за ними символ является, соответственно, верхним или нижним индексом. Если в индексе должно быть несколько символов, то их нужно взять в фигурные скобки (сгруппировать).

Примеры .35 (использование индексов).

<code>\$x^{2y}\$</code>	x^{2y}	<code>\${x_y}_1\$</code>	x_{y_1}
<code>\$x_{2y}\$</code>	x_{2y}	<code>\$x^y^1\$</code>	ошибка
<code>\$x^y_1\$</code>	x_1^y	<code>\$x^{y^2}\$</code>	x^{y^2}
<code>\$x_y^1\$</code>	x_y^1	<code>\${x^y}^2\$</code>	x^{y^2}
<code>\$x_y_1\$</code>	ошибка	<code>cm\$^2\$</code>	cm^2
<code>\$x_{y_1}\$</code>	x_{y_1}	<code>\$x_{\text{рад}}\$</code>	x
		<code>\$_i^j A_{\ \ }^{2y}\$</code>	${}_i^j A_{\ \ }^{2y}$
		<code>\$A^k_{\ }_{ij}^1\$</code>	$A^k_{ij}{}^1$

Пакет **amsmath** вводит команду `\substack{выражения}` для записи многострочных индексов. Для разбиения *выражения* на строки используется команда `\\`.

Пример .36 (использование многострочных индексов).

$$\sum_{\substack{0 \leq i \leq m \\ 0 < j < n}} P(i, j)$$

```
\begin{equation*}
\sum_{\substack{0 \leq i \leq m \\ 0 < j < n}} P(i, j)
\end{equation*}
```

3.2. Корни

`\sqrt[n]{arg.}` — корень из *arg.* степени *n*.

Для корректировки положения указателя в пакете **amsmath** имеются команды `\leftroot{n}` для сдвига влево (вправо, если *n* отрицательно) на *n* математических длин и `\uproot{n}` для сдвига вверх (вниз, если *n* отрицательно).

Примеры .37 (задание корней).

$$\sqrt{x} \quad \sqrt[3]{x} \quad \sqrt[\beta]{k} \quad \sqrt[\beta]{k}$$

```
$$ \sqrt{x} \quad \quad \sqrt[3]{x} \quad \quad
\sqrt{[\beta]k} \quad \quad
\sqrt{[\leftroot{-2}\uproot{2}\beta]k} $$
```

3.3. Дроби

Небольшие дроби можно писать, используя знак `‘/’`. `\frac{числ.}{знам.}` — дробь.

В $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ ’е имеются дополнительные команды: `\dfrac{числ.}{знам.}` и `\tfrac{числ.}{знам.}`, которые форматируют дроби как в вынесенной формуле и в текстовой строке, соответственно.

Примеры .38 (задание дробей). $\frac{a}{b} \quad \frac{a}{b} \quad \frac{a}{b}$ `\frac{a}{b}\;` `\dfrac{a}{b}\;` `\tfrac{a}{b}`

Непрерывную дробь можно получить с помощью команды `\cfrac[сдвиг]{числ.}{знам.}`. В необязательном параметре можно указать *сдвиг* числителя: **l** — влево, **r** — вправо.

Пример .39 (непрерывная дробь).

$$\frac{1}{\sqrt{2} + \frac{1}{\sqrt{2} + \frac{1}{\sqrt{2} + \frac{1}{\sqrt{2} + \frac{1}{\sqrt{2} + \dots}}}}}$$

```
\cfrac{1}{\sqrt{2}+
\cfrac{1}{\sqrt{2}+
\cfrac{1}{\sqrt{2}+
\cfrac[r]{1}{\sqrt{2}+
\cfrac{1}{\sqrt{2}+\dotsb
}}}}}
```

3.4. Биномиальные коэффициенты

Биномиальные коэффициенты можно получить с помощью команды `\choose`.

Пример .40 (биномиальные коэффициенты в L^AT_EX'e). $\binom{n}{n-k}$ `{n\choose n-k}`

В $\mathcal{A}\mathcal{M}\mathcal{S}$ -L^AT_EX'e имеются более удобные команды

- `\binom{числ.}{знам.}`,
- `\dbinom{числ.}{знам.}`,
- `\tbinom{числ.}{знам.}`.

Последние две команды имеют тот-же смысл, что и аналогичные команды для дробей.

Пример .41 (биномиальные коэффициенты в $\mathcal{A}\mathcal{M}\mathcal{S}$ -L^AT_EX'e). $\binom{n}{n-k}$ `\binom{n}{n-k}`

3.5. Размещение объектов друг над другом

В математических выражениях часто используют символы, расположенные друг над другом, как, например, акценты или в бинарных операциях, операциях сравнения, в выражениях со стрелками. Для размещения объектов друг над другом имеется команда `\stackrel{верхний}{нижний}`.

Пример .42 (размещения объектов друг над другом в L^AT_EX'e).

$$\sin \alpha \xrightarrow{\alpha \rightarrow 0} 0$$


```

 $\sin\alpha$ 
 $\stackrel{\alpha}{\rightarrow} 0$ 

```

Пакет **amsmath** предоставляет дополнительные команды:

- `\overset{верхний}{нижний}` — для расположения объекта над символом;
- `\underset{верхний}{нижний}` — для расположения объекта под символом;
- `\sideset{верхний}{нижний}` — для расположения объекта слева и справа от символа, используется, в основном для индексов.

Все эти команды можно использовать в сочетании друг с другом и другими командами.

`\overline{выражение}` и `\underline{выражение}` используют для надчёркивания и подчёркивания *выражения*, соответственно. Команда подчёркивания может применяться и в текстовом режиме.

Команды `\overbrace{выражение}` и `\underbrace{выражение}` вставляют, соответственно, над и под *выражением* горизонтальную фигурную скобку.

Пример .43 (размещения объектов в $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ 'е).

$$\underbrace{A \xrightarrow{\forall} B}_{\forall x \in \mathbb{R}}$$

```

 $\overset{\looparrowright}{\underset{\forall x \in \mathbb{R}}{\underbrace{A \rightharpoonup B}}}$ 

```

Команды `\overleftarrow{выражение}` и `\overrightarrow{выражение}` рисуют левую и правую стрелку над *выражением*, соответственно.

Пакет **amsmath** дополняет их следующими командами:

- `\underleftarrow{выражение}` и `\underrightarrow{выражение}` — для стрелок под *выражением*;
- `\overleftrightharpoonup{выражение}` и `\underleftrightharpoonup{выражение}` — для двунаправленных стрелок над и под *выражением*;
- `\xleftarrow[ниж.]{верх.}` и `\xrightarrow[ниж.]{верх.}` — для растяжимых стрелок, обязательный аргумент *верх.* — верхний индекс, необязательный — нижний.

Пример .44 (размещения стрелок в $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ 'е).

$$\overleftrightarrow{A \xrightarrow[\underset{x \not\subseteq B}{\forall x \in \mathbb{R}}]{B}}$$

```

 $\overleftrightharpoonup{A \xrightarrow[x \not\subseteq B]{\forall x \in \mathbb{R}} B}$ 

```

3.6. Знаки пунктуации и многоточия

Л^AT_EX вставляет небольшие пробелы после знаков пунктуации, но в *математическом режиме* точка, двоеточие и восклицательный знак не являются знаками пунктуации, для этих знаков есть специальные команды:

,	,	;	;	:	<code>\colon</code>	.	<code>\ldotp</code>	.	<code>\cdotp</code>
---	---	---	---	---	---------------------	---	---------------------	---	---------------------

Для многоточий в математической моде имеются свои команды:

<code>\ldots</code>	<code>\ldots</code>	<code>\cdots</code>	<code>\cdots</code>	<code>\vdots</code>	<code>\vdots</code>	<code>\ddots</code>	<code>\ddots</code>
---------------------	---------------------	---------------------	---------------------	---------------------	---------------------	---------------------	---------------------

Горизонтальные многоточия, в большинстве случаев, А_MS-Л^AT_EX правильно поставит при использовании команды `\dots`, используемой и в текстовом режиме. В тех случаях, когда многоточие заканчивает формулу, следует явно указать тип многоточия: `\dotsc`, `\dotscb`, `\dotsci`, `\dotscm`, — соответственно после *заятой*, *бинарного оператора*, *знака интеграла* и *точки умножения*.

3.7. Формулы в рамке

Формулы в рамке создаёт команда из пакета `amsmath` `\boxed{формула}`.

3.8. Условный выбор

Условный выбор в математике записываются с помощью левой фигурной скобки, после которой следуют возможные варианты, в зависимости от условий. Для написания таких конструкций в пакете `amsmath` имеется окружение `cases`:

Пример .45 (выражения с условным выбором).

$$P_{r-j} = \begin{cases} 0, & \text{если } r-j \text{ не чётно;} \\ r!(-1)^{(r-j)/2}, & \text{если } r-j \text{ чётно.} \end{cases}$$

```
\begin{equation*}
\label{eq:C} P_{r-j}=
\begin{cases}
0,& \text{если } r-j \text{ не чётно}; \\
r!(-1)^{(r-j)/2},& \text{если } r-j \text{ чётно}.
\end{cases}
\end{equation*}
```

3.9. Матрицы

Л^AT_EX имеет гибкое окружение `array` для написания многострочных выражений типа *матриц*.

```

\begin{array}{колонки}
... & ... & \\
... & ... & \\
... & ... & \\
\end{array}

```

В параметре *колонки* задаётся формат колонок (l, c, r), литерал формата задаёт выравнивание соответствующей колонки. Внутри колонки разделяются табулятором ‘&’, а строка заканчивается знаком \\.

Пример .46 (построение матрицы с помощью *array*).

$$\mathbf{X} = \left\langle \begin{array}{ccc} x_{11} & x_{12} & \dots \\ x_{21} & x_{22} & \dots \\ \vdots & \vdots & \ddots \end{array} \right\rangle$$

```

\mathbf{X} = \left\langle \begin{array}{lcr}
x_{11} & x_{12} & \dots \\
x_{21} & x_{22} & \dots \\
\vdots & \vdots & \ddots
\end{array} \right\rangle

```

Для написания матриц в пакете **amsmath** имеется следующий набор окружений:

- `pmatrix` — матрица в круглых скобках ();
- `bmatrix` — матрица в квадратных скобках [];
- `vmatrix` — матрица в вертикальных разделителях | |;
- `Vmatrix` — матрица в двойных вертикальных разделителях || ||;
- `matrix` — матрица без скобок.
- `smallmatrix` — маленькая матрица в текстовой строке, скобки, при необходимости, нужно вставлять явно.

Внутри матриц можно использовать команду `\hdotsfor[расст.]{n}`, вставляющую строку точек, занимающую *n* колонок. Необязательный аргумент *расст.* служит для изменения расстояний между точками.

Пример .47 (построение матрицы в $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ ’е). В этом примере команда `\hdotsfor[2]{4}` рисует точки шириной в четыре колонки с увеличенным расстоянием между точками в два раза.

$$\begin{pmatrix} D_1 t & -a_{12} t_2 & \dots & -a_{1n} t_n \\ -a_{21} t_1 & D_2 t & \dots & -a_{2n} t_n \\ \dots & \dots & \dots & \dots \\ -a_{n1} t_1 & -a_{n2} t_2 & \dots & D_n t \end{pmatrix},$$

```

\begin{pmatrix}
D_1t & & -a_{12}t_2 & \dots & -a_{1n}t_n \\
-a_{21}t_1 & D_2t & & \dots & -a_{2n}t_n \\
\hdotsfor[2]{4} \\
-a_{n1}t_1 & -a_{n2}t_2 & \dots & & D_nt
\end{pmatrix}

```

4. Большие формулы

4.1. Уравнения

Большие формулы пишутся в вынесённом режиме. Для этого используется окружение `equation`, которое автоматически вставляет номер уравнения, `equation*` — без номера.

Пример .48 (простое уравнение).

$$f(x) = \cos x$$

$$f(x) = \cos x \tag{1}$$

```

\begin{equation*}\label{eq:simple}
f(x) = \cos x
\end{equation*}
\begin{equation}\label{eq:simple}
f(x) = \cos x
\end{equation}

```

Для многострочных выражений имеются окружения `eqnarray` и `eqnarray*`, которые, как и `equation`, включают математическую моду. Они работают, как таблица **array**, состоящая из трёх столбцов формата `{rcl}`. Средний столбец, обычно, используется для знака отношения. Команда `\` разрывает строки.

Пример .49 (многострочное уравнение).

$$\begin{array}{rcl}
 f(x) & = & \cos x \\
 f'(x) & = & -\sin x \\
 \int_0^x f(y)dy & = & \sin x
 \end{array}$$

```

\begin{eqnarray*}
f(x) & = & \cos x \\
f'(x) & = & -\sin x
\end{eqnarray*}

```

```
\int_{0}^{x} f(y)dy & = & \sin x
\end{eqnarray*}
```

Здесь `\\` используется также, как и в текстовой моде.

4.2. Сложные формулы

Рассмотрим окружения для вынесенных формул пакета **amsmath**.

Пример .50 (окружение *split*).

```
\begin{equation}\label{split}
\begin{split}
a& =b+c-d+\\
& \quad +e-f\\
& =g+h\\
& =i
\end{split}
\end{equation}
```

$$\left. \begin{array}{l} a = b + c - \\ \quad + e - f \\ = g + h \\ = i \end{array} \right|$$

Пример .51 (окружение *multline*).

```
\label{multline}
\begin{multline}
a+b+c+d+\\
+i+j+k+l
\end{multline}
```

$$\left. \begin{array}{l} a + b + c + d + \\ + i + j + k + l \end{array} \right|$$

Пример .52 (окружение *gather*).

```
\begin{gather}\label{gather}
a_1=b_1+c_1\\
a_2=b_2+c_2-d_2+e_2
\end{gather}
```

$$\left. \begin{array}{l} a_1 = b_1 + c_1 \\ a_2 = b_2 + c_2 - d_2 + e_2 \end{array} \right|$$

Пример .53 (окружение *align*).

```
\begin{align}\label{align}
a_1& =b_1+c_1\\
a_2& =b_2+c_2-d_2+e_2
\end{align}
```

$$\left. \begin{array}{l} a_1 = b_1 + c_1 \\ a_2 = b_2 + c_2 - d_2 + e_2 \end{array} \right|$$

```
\begin{align}\label{align2}
a_{11}& =b_{11}&
a_{12}& =b_{12}\\
a_{21}& =b_{21}&
a_{22}& =b_{22}+c_{22}
\end{align}
```

$$\left. \begin{array}{l} a_{11} = b_{11} \quad a_{12} = b_{12} \quad (8) \\ a_{21} = b_{21} \quad a_{22} = b_{22} + c_{22} \quad (9) \end{array} \right|$$

Пример .54 (окружение *alignat*).

```
\begin{alignat}{2}\label{alignat}
a_{11}& =b_{11}&
a_{12}& =b_{12}\\
a_{21}& =b_{21}&
a_{22}& =b_{22}+c_{22}
\end{alignat}
```

$$\left. \begin{array}{l} a_{11} = b_{11} a_{12} = b_{12} \\ a_{21} = b_{21} a_{22} = b_{22} + c_{22} \end{array} \right|$$

```

\begin{flalign*}
a_{11}&=b_{11}& \\
a_{12}&=b_{12}& \\
a_{21}&=b_{21}& \\
a_{22}&=b_{22}+c_{22} &
\end{flalign*}

```

Пример .55 (окружение *flalign*).
$$\left. \begin{array}{l} a_{11} = b_{11} \\ a_{21} = b_{21} \end{array} \right\}$$

Для вставки текста в многострочную формулу используется команда `\intertext{текст}`.

В сложных формулах, определённых в $\mathcal{A}\mathcal{M}\mathcal{S}$ - \LaTeX 'е, автоматический разрыв страницы не производится. Для управления разрывом определены 2 способа. Команда `\displaybreak[N]` (ставится перед `\`) действует аналогично `\pagebreak`. Декларация `\allowdisplaybreaks[N]` может располагаться в любом месте и действует внутри скобок.

4.3. Позиционирование

Команды, регулирующие размеры бокса, часто бывают полезны, особенно в математических формулах:

- `\mathstrut` — бокс, имеющий высоту круглой скобки и нулевую ширину.
- `` — бокс, имеющий ширину *формулы* и нулевую высоту.
- `\vphantom{формула}` — бокс, имеющий высоту *формулы* и нулевую ширину.
- `\smash[t|b]{формула}` — бокс с *формулой*, имеющий нулевую высоту. Возможны опции: *t* — зануляется только верхняя часть бокса (выше базисной линии); *b* — зануляется нижняя часть бокса (глубина).

Все эти команды могут использоваться и в текстовой моде.

4.4. Нумерация и ссылки

Большие формулы часто нумеруются, причём нумерация происходит автоматически. В многострочных выражениях бывают ситуации, когда номера нужны лишь для некоторых строк.

Для ручного управления имеются команды:

- `\tag{метка}` — ставит *метку* (номер);
- `\tag*{метка}` — ставит *метку* без окружающих скобок;
- `\notag` (или \LaTeX 'овская команда `\nonumber`) — убирает номер.

Для подчинённой нумерации (формулы внутри группы) используется окружение `subequations`, как в примерах (12a), (12b) и (12c).

Пример .56 (окружение *subequations*).

$$f(x) = \cos x \tag{12a}$$

$$f'(x) = -\sin x \tag{12b}$$

$$\int_0^x f(y)dy = \sin x \tag{12c}$$

```
\begin{subequations}
\begin{align}
\label{f} f(x) &= \cos x \\
\label{f'} f'(x) &= -\sin x \\
\label{F} \int_0^x f(y)dy &= \sin x
\end{align}
\end{subequations}
```

Ссылки на формулы пишутся, как и их номера, в круглых скобках. В пакете **amsmath** для ссылок имеется команда `\eqref{имя}`.

Пример .57 (ссылки на формулы). Нумерация формул показана в примерах (1)–(10), а подчинённая — в примерах (12a), (12b) и (12c).

Нумерация формул показана в примерах `\eqref{eq:simple}`–`\eqref{alignat}`, а подчинённая --- в примерах `\eqref{f}`, `\eqref{f'}` и `\eqref{F}`.

5. Шрифты

Математические шрифты, как и текстовые, различаются гарнитурой, насыщенностью, начертанием и размером. Однако, в математических шрифтах нельзя изменять отдельные атрибуты.

Шрифты определяются математической версией (командой `\mathversion{версия}`). Изначально существует две версии: **bold** и **normal**, которые в L^AT_EX 2_ε задаются, соответственно, декларациями `\boldmath` и `\unboldmath`. Переключать версию можно только вне математической моды.

Внутри математической моды можно изменять математические алфавиты, они определяют начертание букв, цифр, прописных греческих букв и акцентов.

Пакет `amsbsy` (загружаемый пакетом **amsmath**) вводит команду `\boldsymbol{выражение}`, которая выводит *выражение* в **bold** версии математических шрифтов.

Может оказаться, что для какого-либо символа отсутствует **bold**-вариант. Для этого случая имеется команда `\pmb{символ}`, рисует символ несколько раз, с небольшими смещениями.

Глава 7

Таблицы в L^AT_EX'e

1. Табулятор

Табулятор создаёт окружение `tabbing`.

Внутри используются следующие управляющие команды:

`\=` — устанавливает табулятор;

`\>` — переход к следующему табулятору справа;

`\\` — начинает новую строку;

`\kill` — переход на новую строку без печати текста текущей строки с сохранением табуляторов, обычно используется для задания шаблона;

`\+` — передвигает левое поле последующих строк на один табулятор вправо, как бы вставляя в начало строк контрольный символ `\>`, несколько таких команд дают суммарный эффект;

`\-` — передвигает левое поле последующих строк на один табулятор влево, отменяя тем самым одну, выданную ранее, команду `\+`;

`\<` — переход к предыдущему табулятору слева, применимо лишь после употребления одной или нескольких команд `\+`;

`\pushtabs` — запоминает текущее положение табуляторов, чтобы восстановить их по команде `\poptabs`;

`\poptabs` — восстанавливает положение табуляторов, сохранённое по команде `\pushtabs`;

`\'` — сдвигает в колонке текст, предшествующий команде, вправо, последующий текст выровнен как обычно — влево;

`\'` — сдвигает весь последующий текст к правой границе *страницы*, действует только на *последнюю* колонку;

`\a=`, `\a'`, `\a'` — заменяют, переопределённые внутри **tabbing** — команды акцентирования `\=`, `\'`, `\'`.

Пример .59 (табулированный текст). Кафедра ПИЭГМУ\quad \= 29--65--47
\quad\= с 10^{00} до ∞ \kill
\textbf{Подразделение} \> \textbf{Телефон}
\> \textbf{Часы приёма} \\
Деканат МИЭМИС \> 29--65--47
\> с 8^{00} до 17^{00} \\
Кафедра ПИЭГМУ \> 29--65--25
\> с 9^{00} до ∞ \\

Подразделение	Телефон	Часы приёма
Деканат МИЭМИС	29-65-47	с 8^{00} до 17^{00}
Кафедра ПИЭГМУ	29-65-25	с 9^{00} до ∞

2. Таблица

Таблица в L^AT_EX'e — это бокс из последовательных рядов, элементы которых выровнены в столбцах.

Текст внутри ячеек обрабатывается в текстовой моде, т. е., при необходимости, разбивается на строки.

Таблица может быть разливована. В L^AT_EX'e имеется три разновидности таблиц.

`array[положение]{формат}` — используется только в математической моде.

`tabular[положение]{формат}` — используется в любой моде.

`tabular*{ширина}[положение]{формат}` — таблица заданной *ширины*.

Аргумент *ширина* используется только в **tabular*** и задаёт ширину таблицы. Необязательный аргумент задаёт *положение* таблицы относительно текущей базисной линии:

t — по текущей базисной линии выравнивается базисная линия *верхней строки*;

c — по текущей базисной линии выравнивается *центр таблицы* (действует по умолчанию);

b — по текущей базисной линии выравнивается базисная линия *нижней строки*.

Аргумент *формат* задаёт количество и формат колонок. Можно использовать следующие управляющие символы:

l — текст сдвинут к левому краю колонки.

c — текст расположен по центру колонки.

r — текст сдвинут к правому краю колонки.

p{*ширина*} — текст выровнен по ширине в колонке шириной *ширина* (как в `\parbox`).

| — вертикальная линия на полную высоту и глубину бокса (два символа подряд создают двойную линию).

@{*текст*} — вставка *текста* во все строки, *текст* является подвижным аргументом, в окружении **array** он обрабатывается в математической моде. Такая конструкция называется ещё @-выражением. @-выражение подавляет горизонтальный пробел, вставляемый между колонками.

Команда `\extracolsep{ширина}` в @-выражении вставляет дополнительный пробел величиной *ширина* слева всех последующих колонок (кроме первой).

*{*n*}{*формат*} — мультиформат, задаёт *n* повторений формата колонок *формат*.

Внутри окружений записываются строки таблиц, где можно использовать следующие команды и управляющие символы:

& — табулятор (разделитель колонок), количество ячеек в строке должно совпадать с количеством столбцов, заданном в формате таблицы;

`\\[высота]` — начинает новую строку, может иметь необязательный параметр *высота*, увеличивающий расстояние между этой и следующей строкой на заданную величину;

`\tabularnewline[высота]` — аналогична предыдущей команде, используется для отличия в исходном файле конца строки таблицы от конца строки в других командах внутри таблицы (`\parbox`, окружениях **minipage**, **tabular**, **array**, при использовании деклараций `\centering`, `\raggedright`, `\raggedleft`, которые можно использовать в ячейках таблицы);

`\vline` — вставляет вертикальную линию на полную высоту и глубину строки (может использоваться в @-выражениях);

`\hline` — вставляет горизонтальную линию на всю ширину бокса (две подряд команды `\hline` рисуют двойную линию с зазором, непересекаемую вертикальными линиями);

`\cline{i-j}` — проводит горизонтальную линию через колонки с *i*-й по *j*-ю;

`\multicolumn{n}{формат}{текст}` — создаёт одну ячейку шириной *n* колонок с форматом *формат*. При употреблении крайних вертикалей в аргументе *формат* надо следить за положением вертикальных линий, обычно для всех, кроме крайних ячеек не следует задавать левую вертикаль, если она задана в формате таблицы.

2.1. Параметры настройки

Следующие команды представляют из себя командные длины, которые можно изменять как за пределами таблиц так и внутри ячейки. В последнем случае обязательно следует ограничивать область действия этих команд.

`\arraycolsep` — половина ширины горизонтального пробела между колонками в окружении **array**;

`\tabcolsep` — половина ширины горизонтального пробела между колонками в окружениях **tabular** и **tabular***;

`\arrayrulewidth` — толщина линий;

`\doublerulesep` — ширина зазора между двойными линиями.

`\arraystretch` — коэффициент, определяющий интервал между строками. Действительный интервал получается умножением этого коэффициента на стандартное расстояние между строками, определённое для используемых в строке шрифтов. Значение коэффициента изменяется с помощью команды `\renewcommand` (как и коэффициент `\baselinestretch`).

3. Размещение таблиц

При использовании окружения **tabbing** таблица обрабатывается в текстовой моде, в том месте, где она написана и обрабатывается по строкам, т. е., при необходимости переносится на следующую страницу.

При использовании окружения **tabular** создаётся неразрывный бокс в месте написания таблицы а содержимое таблицы обрабатывается не последовательно, как текст, а целиком, после прочтения всей таблицы. Во первых, это приводит к увеличению времени компиляции (по сравнению с **tabbing**), а во вторых, может привести к переполнению страницы (когда таблица не помещается по высоте страницы). Поэтому, чтобы не замедлять работу компилятора, следует, по мере возможностей использовать окружение **tabbing**.

Для правильного расположения сложных таблиц надо их создавать, как *плавающий объект*.

Плавающим называется объект, расположением которого (в выходном файле) занимается сам компилятор.

В $\text{\LaTeX} 2_{\epsilon}$ имеется три таких объекта: *таблица*, *рисунок* и *заметка на полях*.

Плавающая таблица задаётся окружениями: `table[положение]` и `table*[положение]`.

В одноколоночной печати эти окружения не различаются, а в двухколоночной — первое окружение всегда размещает таблицу в одной колонке, а второе — в двух (на страницу, как в одноколоночной печати).

Аргумент *положение* задаёт расположение бокса на странице:

- t** — (top) расположение *вверху страницы*;
- b** — (bottom) расположение *внизу страницы*, в случае неудачи, L^AT_EX будет пытаться разместить объект вверху, затем внизу последующих страниц;
- p** — (page) расположение *на отдельной странице*, содержащей только плавающие объекты;
- h** — (here) расположение, *по возможности на месте*, сразу же после заполнения текущей строки, в случае неудачи, **h** заменяется на **t** (не допускается в **table*** при печати в две колонки);
- !** — расположение *на указанном месте*, игнорируя большинство запретов, используется перед дескрипторами **h**, **t** и **b** (хотя бы с одним).

Опции действуют независимо от порядка перечисления. По умолчанию действует опция [tbp].

`\supressfloats[t|b]` — запрещает появление плавающих объектов на текущей странице. Запрет действует с того места, где написана декларация, до конца страницы. Дополнительные опции запрещают появление плавающих объектов только вверху или внизу страницы, соответственно, однако знак ‘!’ в аргументе таблицы *положение* подавляет эту декларацию.

В теле окружения **table** обычно помещается окружение **tabular**. Таблицы часто снабжают подписями.

Для подписи используется команда `\caption[альт.подпись]{подпись}`, которая должна предшествовать содержимому таблицы. Тогда аргумент *подпись* будет размещён правильно, — сверху таблицы, после номера, проставляемого автоматически (см. табл. 7.1).

Текст подписи заносится в список таблиц (файл .lot). Если подпись слишком большая или сложная, то следует воспользоваться необязательным аргументом *альт.подпись*, который будет помещён в список таблиц.

Обычно, при использовании `\caption` ставят метку (`\label`), ставить её нужно обязательно после `\caption` (см. табл. 7.1).

Пример .60 (таблица). (см. `\табл.~\ref{pi}`)

Таблица 7.1.

Пример таблицы

Выражение с π	Значение
π	3.1416
π^π	36.46
$(\pi^\pi)^\pi$	80662.7

Пример .61 (исходник таблицы). `\begin{table}[htb]`
`\centering`
`\caption{Пример таблицы}\label{pi}`

```

\renewcommand{\arraystretch}{1.2}
\begin{tabular}{||c|r@{.}l||}
\hline
Выражение с  $\pi$  & & \\
\multicolumn{2}{c||}{Значение} & \\
\hline\hline
 $\pi$  & 3.1416 & \\
 $\pi^{\pi}$  & 36.46 & [1mm]
 $(\pi^{\pi})^{\pi}$  & 80662.7 & \\
\hline
\end{tabular}
\end{table}

```

Глава 8

Графика в L^AT_EX 2_ε

1. Основные понятия

Способы воспроизведения графики в L^AT_EX'e:

1. Набор шрифтов с элементарными линиями (окружение **picture** в L^AT_EX'e, пакет X_y-pic).
2. Макропакеты для рисования, дополняющие и расширяющие окружение **picture** (**epic**, **eepic**, **feynman**, X_MT_EX, **bg**, **bidding**, **crossword**, **cwpuzzle**).
3. Шрифты с картинками, созданные и встраиваемые METAFONT'ом или METAPOST'ом и основанные на них пакеты (**mfpic**, FeynMF, **timing**, **circ**, MusiX_TE_X, **chess**, **bdfchess**, **cchess**, **go**).
4. ASCII-рисование — рисование кривых точками (P_IC_TE_X, ppch_TE_X).
5. Встраиваемый PostScript (PSTrics, **axodraw**).
6. Включение графических файлов разных форматов (в зависимости от драйверов и набора графических фильтров), созданных графическими редакторами (пакеты **epsf**, **epsfig**, **graphics**, **graphicx**).
7. Создание изображений в графических редакторах с последующей конвертацией в какой-либо внутренний формат L^AT_EX'a (m4, GNUPlot, MAPLE, MatLab, Mathematica, ...).

2. Импортирование графики

Вставка (импорт) графических файлов в T_EX-документ осуществляется с помощью внешних *программ-драйверов* (см. табл. 8.1), выполняющих вывод на внешние устройства (монитор, принтер).

Для непосредственного обращения к драйверам существует команда `\special`, имеющая множество аргументов, специфичных для каждого драйвера. Чтобы упро-

Наиболее распространённые драйверы

Драйвер	Автор	ОС
dvips	Tom Rokicki	все ОС
dview	Василий К. Малышев	Windows
dviwin	Hippocrates Sendoukas	Windows
emtex	Eberhard Mattes	MSDOS, OS/2
xdvi	Paul Vojsa	UNIX
windvi	Fabrice Popineau	Windows

стить работу пользователя, создано много пакетов для работы с внешними устройствами, в том числе и вставки графических файлов.

На сегодня широко известны такие пакеты, как **epsf** Тома Рокички (*Tom Rokicki*), **epsfig** Себастьяна Раца (*Sebastian Rahtz*), **graphics** и **graphicx** Дэвида Карлисли (*David Carlisle*) и Себастьяна Раца.

Последние два пакета являются наиболее современными и развитыми, они способны вставлять графические файлы любых форматов, поддерживаемых драйверами, а не только .eps файлы (как первые два пакета). Кроме того, коллекция **graphics** содержит ещё некоторые пакеты для работы с внешними устройствами.

Пакеты **graphics** и **graphicx** осуществляют одинаковый доступ к драйверам и вводят одинаковый набор команд, они различаются лишь способом задания необязательных аргументов.

Наиболее совершенным и удобным пакетом является **graphicx** из коллекции **graphics**

При подключении пакета можно выбрать *драйвер* из следующего списка:

dvips (используется по умолчанию), xdvi, dvipdf, pdftex,
 dvipsone, dviwindo, emtex, dviwin, pctexps, pctexwin, pctexhp,
 pctex32, truetex, tcidvi, oztex, textures.

Импортирование графических файлов осуществляется с помощью команды `\includegraphics[параметры]{файл}` или `\includegraphics*[параметры]{файл}`. Модифицированная команда обрезает часть рисунка, выходящую за пределы заданного бокса.

Чтобы включить в печатный документ рисунок, записанный в графическом файле, достаточно в нужное место в исходном файле вставить команду `\includegraphics{файл}`. В качестве разделителя директорий используется не ‘\’ а ‘/’, как в UNIX.



Пример .62 (вставка файла zip).

```
\includegraphics{zip}
```

Параметры отделяются друг от друга запятой. Параметры записываются в виде равенства, в левой части которого стоит наименование параметра, а в правой — его

значение. Многие параметры имеют размерные величины. По умолчанию используется единицы измерения **bp** (большие пункты, 1 in=72 bp).

bb — параметр, используемый для задания всех координат углов ограничивающего бокса (**BoundingBox**). Значение параметра должно содержать четыре числа, разделённых пробелами.

Например, `bb=0 0 1in 2in` задаёт ограничивающий бокс с координатами нижнего левого угла (0,0) и с координатами верхнего правого угла — (72,144).

bbllx, **bblyl** — x- и y-координаты нижнего левого угла ограничивающего бокса.

bburx, **bbury** — x- и y-координаты верхнего правого угла ограничивающего бокса.

Набор ключей `bbllx=a`, `bblyl=b`, `bburx=c`, `bbury=d` эквивалентен `bb= a b c d`.

natwidth, **natheight** — естественная ширина и высота рисунка. Эти параметры удобны для установки координат верхнего правого угла в случае, когда обе координаты нижнего левого угла равны нулю.

Набор ключей `natwidth=w`, `natheight=h` эквивалентны `bb= 0 0 w h`.

hiresbb — указывает, что L^AT_EX должен считывать размеры ограничивающего бокса из строки, начинающейся с `%%HiresBoundingBox`, а не из строки `%%BoundingBox`.

`hiresbb=true` эквивалентен простому заданию параметра `hiresbb` без значения.

В случае, когда по умолчанию считываются размеры изображения с высоким разрешением, ключ `hiresbb=false` позволяет импортировать первый рисунок.

clip — может принимать значения `true` или `false`. Если в параметрах присутствует запись `clip=true` (или просто `clip`), то часть рисунка, выходящая за границы видимой области отсекается.

viewport — задаёт видимую часть рисунка. Подобно параметру `bb`, видимая часть рисунка определяется двумя парами координат нижнего левого угла и верхнего правого угла видимой области.

Координаты углов видимой области отсчитываются от левого нижнего угла ограничивающего бокса.

trim — альтернативный способ задания видимой части рисунка. Положительные числа соответствуют смещению границы видимой области внутрь ограничивающего бокса, отрицательные числа расширяют видимую область. Например, `trim= 1mm 2mm 3mm 4mm` отрезают от рисунка 1 мм слева, 2 мм снизу, 3 мм справа и 4 мм сверху.

draft — устанавливает черновой режим импортирования, когда вместо рисунка выводится рамка с размерами ограничивающего бокса, а внутри неё написано название импортированного файла. Параметр `draft` эквивалентен заданию `draft=true`.

Если включена опция `draft` пакета `graphicx` или класса документа, то параметр задание `draft=false` позволяет реально импортировать отдельный рисунок.

scale — изменение размера рисунка по заданному масштабу. Ключ `scale=2` увеличивает рисунок в два раза относительно его естественного размера. Пропорции рисунка сохраняются.

width — требуемая ширина рисунка.

hieght|totalheight — требуемая *высота* / *полная высота* рисунка, соответственно. Задание одного из параметров `width`, `height`, `totalheight` приводит к изменению масштаба рисунка с сохранением пропорций.

keepaspectratio — означает `keepaspectratio=true`, задаёт режим масштабирования рисунка с сохранением пропорций. Рисунок увеличивается настолько, насколько это возможно при сохранении его аспектного отношения без выхода изображения за заданные размеры видимой области.

angle — угол поворота рисунка (в градусах), положительное значение задаёт вращение против часовой стрелки.

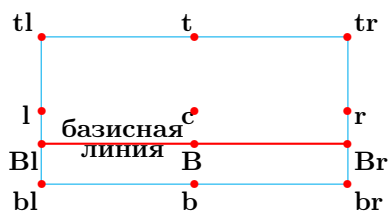
origin — определяет положение центра вращения рисунка, по умолчанию рисунок вращается относительно точки привязки.

Значение ключа может состоять из одной или двух букв, фиксирующих положение центра вращения. Горизонтальная координата центра вращения задаётся одним из трёх спецификаторов: `l` — слева, `c` — центр, `r` — справа, в то время как вертикальная координата определяется одним из четырёх спецификаторов: `t` — вверх, `c` — центр, `b` — низ, `B` — базисная линия.

Например, `origin=rb` задаёт в качестве точки вращения правый нижний угол бокса, `origin=lt` задаёт левый верхний угол. Комбинация `cB` задаёт середину отрезка базисной линии, проходящей через вращаемый бокс, а `lc` задаёт среднюю по высоте точку с левой стороны бокса. Точке привязки соответствует `lb`. Всего таким способом можно задать 12 точек. Порядок следования спецификаторов не имеет значения: `br` эквивалентно `rb`. Если только задан один спецификатор, то вторым предполагается `c`.

type — задаёт тип рисунка. Все файлы одного типа обрабатываются при помощи одних и тех же внутренних команд `\special` (конкретная форма которых зависит от выбранного драйвера).

Например, файлы с расширениями `.ps` и `.eps` рассматриваются драйвером `dvips` как файлы одного типа векторной графики *EPS*, а файлы с расширениями `.bmp` и `.psx` считаются принадлежащими к типу растровой графики *BMP*.



Поэтому следует задать ключ `type=eps`, если рисунок *PostScript* почему-то записан в файл с расширением, не совпадающим с `.eps` и `.ps`.

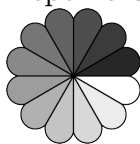
ext — определяет расширение имени графического файла. Например, если задан ключ `ext=eps`, то драйвер выходного устройства попытается загрузить файл с именем *файл.eps*.

read — определяет расширение имени файла, в котором записаны размеры ограничивающего бокса в виде строки `%%BoundingBox: llx lly urx ury`, который считывается ЛАТЭХ'ом при компиляции исходного текста печатного документа. Для графического файла с расширением `.eps` размеры считываются из этого же файла, поэтому применение ключа `read=.eps` повторяет правило, используемое по умолчанию.

command — задаёт команду, которую программа-драйвер должна применить к графическому файлу перед его импортированием. С eps-файлом не требуется производить каких-либо предварительных действий, поэтому этот ключ задавать не нужно. Обычно его используют при импортировании сжатых файлов.

ЗАПОМНИТЕ: Если параметры пропущены, то рисунок будет вставлен с естественными размерами, т. е. размерами ограничивающего бокса (в .eps файле размеры записываются в параметре `BoundingBox`).

Пример .63. В этом примере вставлен рисунок, не превысив заданных размеров и



не изменив пропорции.

```
\includegraphics[keepaspectratio,  
width=0.7in, height=4in]{rosette}
```

Высота рисунка не достигла заказанной величины в 4 дюйма, т. к. ширина рисунка превысила бы заказанное значение в 0,7 дюйма или были бы искажены пропорции рисунка.

В случаях, когда в печатный документ (используя драйвер `dvips`) нужно вставить *растровый* рисунок, необходимо указать его размеры в аргументе команды `\includegraphics` или в специальном текстовом файле. С помощью этих размеров осуществляется и масштабирование.

3. Размещение рисунков

Рисунки, вставляемые в текст из внешних файлов и рисунки, созданные средствами самого ЛАТЭХ'а образуют бокс заданного размера и обрабатываются как все боксы.

Часто требуется рисунок отделить от текста, снабдить его подписью и разместить нужным образом на странице. Для автоматизации этого процесса имеется специальное окружение, помещающее рисунок (вообще — любое содержимое) в **плавающий бокс**.

Плавающий рисунок задаётся, подобно плавающим таблицам, окружениями `figure[положение]` и `figure*[положение]`. В двухколоночной печати, первое окружение всегда размещает рисунок в одной колонке, а второе — в двух (на всю ширину страницы, как в одноколоночной печати).

Необязательный аргумент *положение* задаёт расположение бокса на странице, он может состоять из следующих символов:

- h** — (here) расположение на месте, сразу же после заполнения текущей строки (не допускается в *-форме при печати в две колонки);
- t** — (top) расположение вверху страницы;
- b** — (bottom) расположение внизу страницы;
- p** — (page) расположение на отдельной странице, содержащей только плавающие объекты.
- !** — расположение *на указанном месте*, игнорируя большинство запретов, используется перед дескрипторами **h**, **t** и **b** (хотя бы с одним);

Подпись создаётся командой `\caption{подпись}`, которая ставится внутри окружения **figure** после самого рисунка, при этом L^AT_EX сформирует соответствующую подпись с номером рисунка, как, например рис. 8.1.

Пример .64 (плавающий рисунок). `\begin{figure}[hbtpt]`
`\centering`
`\includegraphics[width=1.\textwidth,`
`bb=59 366 507 519,clip]{pic1}`
`\caption{Результаты классификации}`
`\label{SamplePic}`
`\end{figure}`

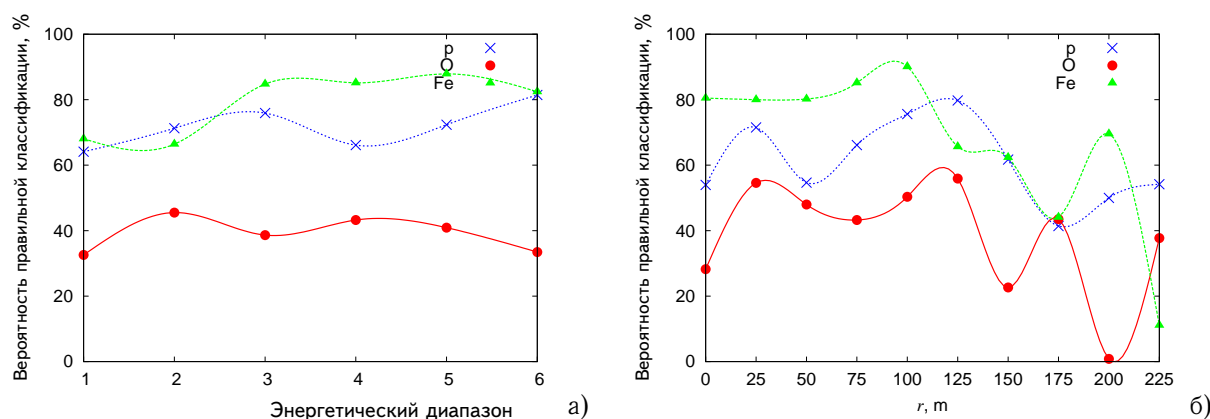
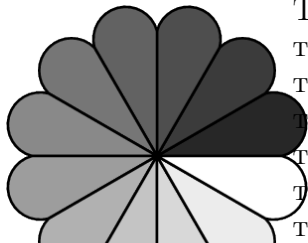


Рис. 8.1. Результаты классификации

Подробнее о плавающих объектах написано в разделе, описывающем таблицы. Для вставки рисунка, обтекаемого текстом (справа или слева), существуют специальные пакеты:

- `floatflt` — для плавающих рисунков;
- `wrapfig` — для неплавающих рисунков.



Текст текст текст текст текст текст текст текст текст текст
 текст текст текст текст текст текст текст текст текст текст
 текст текст текст текст текст текст текст текст текст текст
 текст текст текст текст текст текст текст текст текст текст
 текст текст текст текст текст текст текст текст текст текст
 текст текст текст текст.

Пример 65 (обтекаемый плавающий рисунок). `\begin{floatingfigure}{.25\textwidth}`
`\includegraphics[width=.25\textwidth,height=.25\textheight,`
`keepaspectratio]{rosette}`
`\end{floatingfigure}`

Текст текст текст текст текст текст текст текст текст текст текст текст текст текст текст

4. Псевдографика

Л^AT_EX имеет несколько шрифтов псевдографики (элементарных линий разных размеров), которые используются исключительно в **графической моде**.

Для перехода в графическую моду служит окружение `picture(ширина, высота)(x_0, y_0)`. *Ширина* и *высота* задают размеры графического бокса и являются обязательными, а x_0 и y_0 задают координаты левого нижнего угла и являются необязательными, по умолчанию $x_0 = 0$, $y_0 = 0$.

Аргументы окружения **picture** (как обязательные, так и необязательные) задаются в круглых скобках, чтобы подчеркнуть, что они являются парами длин (по осям X и Y).

Размерность не указывается а используется текущая единица измерения, задаваемая длиной `\unitlength` (по умолчанию она равна 1 pt), которая переопределяется до окружения **picture**. Такие же правила задания длин и координат действуют для всех команд в графическом режиме.

Графические элементы могут изображаться линиями разной толщины:

- `\thinlines` — тонкие линии ↗ (используется по умолчанию);
- `\thicklines` — толстые линии ↗.

Команда `\linethickness{толщина}` служит для произвольного задания *толщины* горизонтальных и вертикальных линий.

4.1. Позиционирование

Любой объект в графике должен быть помещён в определённую точку, задаваемую парой координат (x, y) (иначе объект будет помещён в точку $(0,0)$). Команда

`\put(x,y){объект}`, размещает *объект* так, чтобы его точка привязки находилась в точке с координатами (x, y) .

Для размещения многократно повторяющегося объекта существует команда `\multiput(x,y)(dx,dy){n}{объект}`, — создаёт n копий *объекта*, размещая их так, чтобы точка привязки i -й копии находилась в точке с координатами $(x + [i - 1]dx, y + [i - 1]dy)$.

4.2. Линии

Для задания *прямых линий* используется команда `\line(l_x, l_y){длина_x}`, где пара (l_x, l_y) задают относительные проекции линии на соответствующие координатные оси, тем самым определяя наклон линии. l_x и l_y могут принимать только целые значения из диапазона $[-6; 6]$ и не могут иметь общего делителя. Длина линии определяется проекцией на ось X , задаваемом в параметре $длина_x$.

Кроме линий существует возможность рисовать *векторы* (линии со стрелками на конце). Векторы задаются командой `\vector(l_x, l_y){длина_x}`, её аргументы имеют такой же смысл как и в команде `\line`, с единственным отличием: l_x и l_y могут принимать значения из диапазона $[-4; 4]$.

В \LaTeX е имеется возможность построения кривых второго порядка. Команда `\qbezier[N](x_A, y_A)(x_B, y_B)(x_C, y_C)` рисует квадратичную кривую Безье. Для задания кривой Безье второго порядка необходимо задать начальную (А) и конечную точки (С) а также контрольную точку (В), в которой пересекаются касательные, проведённые к параболе в граничных точках.

Необязательный аргумент N задаёт количество точек, используемых для аппроксимации кривой. По умолчанию это число определяется параметром `\qbeziermax`, его можно изменить с помощью команды `\renewcommand`.

4.3. Круги

Команда `\circle{диаметр}` рисует окружность \bigcirc .

Команда `\circle*{диаметр}` рисует круг \bullet .

Точкой привязки этих объектов служит центр окружности. *Диаметр* может даваться любым положительным числом, однако реальный диаметр окружности будет выбран \LaTeX 'ом из набора имеющихся шрифтов, наиболее близкий к заданному. Максимальный диаметр для окружности обычно составляет 40 pt а для круга — в два раза меньше.

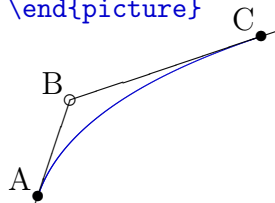
Пример .66 (линии и круги). Здесь кривая Безье задана командой `\qbezier(5,5)(15,35)(75,55)`.

```
\begin{picture}(100,40)(10,10)
{\color{MyBlue}\qbezier(5,5)(15,35)(75,55)}
\put(5,5){\circle*{3}}
\put(3,7){\makebox(0,0)[br]{A}}
\put(15,35){\circle{3}}
```

```

\put(13,37){\makebox(0,0)[br]{B}}
\put(75,55){\circle*{3}}
\put(73,57){\makebox(0,0)[br]{C}}
\put(15,35){\line(-1,-3){11}}
\put(15,35){\line(3,1){65}}
\end{picture}

```



4.4. Овалы

Овал — это прямоугольник с максимально скруглёнными углами .

Команда `\oval(ширина, высота) [часть]` строит овал с заданной *шириной* и *высотой*. Точка привязки овала находится в его центре.

Необязательный аргумент задаёт *часть* овала, он может состоять из одного или двух спецификаторов: **l** — левый; **r** — правый; **t** — верхний; **b** — нижний.

Если задан один спецификатор, то рисуется соответствующая половина овала, если заданы два совместимых спецификатора, то — соответствующая четверть овала.

Чтобы нарисовать скругление меньшего радиуса, используются четверти овалов соответственно меньшего размера.

4.5. Боксы

В графической моде текстовые боксы имеют другой синтаксис и в этих боксах не добавляются окружающие пробелы вокруг текста. Команды выглядят следующим образом: `\makebox(ширина, высота) [позиц.] {текст}`;

`\framebox(ширина, высота) [позиц.] {текст}`.

Позиционирование может состоять из одного или двух спецификаторов:

- l** — текст сдвинут к левому краю бокса;
- c** — текст расположен по центру бокса;
- r** — текст сдвинут к правому краю бокса;
- s** — текст растянут на всю ширину боксу;
- t** — текст сдвинут к верхнему краю бокса;
- b** — текст сдвинут к нижнему краю бокса.

По умолчанию текст центрируется.

Кроме перечисленных боксов, в графической моде добавляются ещё несколько команд: `\dashbox{штрих}(ширина, высота)[позиц.] {текст}` — для боксов, обведённых пунктирным контуром, длина пунктира задаётся параметром *штрих*, измеряемым в единицах `\unitlength`.

`\frame{текст}` — упрощённый вариант команды `framebox`, аналог команды `\fbox` в текстовой моде.

`\shortstack[позиционирование] {текст}` — формирует стек. Текст внутри бокса располагается в одну колонку, строки разделяются командой `\\`. Опция позиционирование позволяет выравнивать текст по горизонтали, допустимые значения: *r*, *l*, *c* (используется по умолчанию).

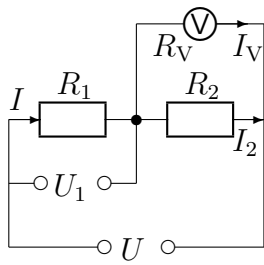
Эта команда может использоваться не только в графическом, но и в любом другом режиме.

Для сохранения бокса также существует «графический вариант»: `\savebox{имя}(ширина, высота)`. Здесь *имя* также задаёт имя сохраняемого бокса. Рекомендуется использовать эту команду для сложных рисунков из повторяющихся графических элементов, т. к. с этой командой компиляция происходит быстрее, чем с использованием вложенных команд `\multiput`.

Пример .67 (линии и круги). `\begin{picture}(80,100)(0,-50)`

```
\put(10,-20){\line(1,0){8}}
\put(50,-20){\line(-1,0){8}}
\put(10,-40){\line(1,0){28}}
\put(90,-40){\line(-1,0){28}}
\put(20,-20){\circle{4}}
\put(10,3){$I$}
\put(40,-20){\circle{4}}
\put(40,-40){\circle{4}}
\put(45,-44){$U$}
\put(60,-40){\circle{4}}
\put(10,0){\line(1,0){10}}
\put(10,0){\vector(1,0){10}}
\put(40,0){\line(1,0){20}}
\put(24,-24){$U_1$}
\put(80,0){\line(1,0){10}}
\put(80,0){\vector(1,0){10}}
\put(80,-10){$I_2$}
\put(50,0){\circle*{3}}
\put(50,-20){\line(0,1){50}}
\put(90,30){\line(0,-1){70}}
\put(10,0){\line(0,-1){40}}

\put(50,30){\line(1,0){15}}
\put(75,30){\line(1,0){15}}
\put(75,30){\vector(1,0){10}}
\put(80,20){$I_{\text{V}}$}
\thicklines
\put(20,-5){\framebox(20,10){}}
\put(25,8){$R_1$}
\put(60,-5){\framebox(20,10){}}
\put(65,8){$R_2$}
\put(70,30){\circle{10}}
\put(70,30){\makebox(0,0){\small\sf V}}
\put(55,20){$R_{\text{V}}$}
\end{picture}
```

5. Цвет

Для вывода в цвете Т_ЕX'у также нужно указать драйвер устройства. Чтобы за- действовать цвет надо подключить пакет `color` из коллекции `graphics`. Представле- ние цвета зависит не от Л^AT_ЕX'а (пакета `color`), а от используемого драйвера.

Обычно все драйверы поддерживает 4 цветовые модели:

`gray` — задаётся числом от 0 до 1.

`rgb` — задаётся тройкой чисел от 0 до 1.

`cmuk` — задаётся четвёркой чисел от 0 до 1.

`named` — задаётся именем. Драйвер `dvips` содержит определения 68 имён цветов.

5.1. Определение цвета

Любой драйвер поддерживает цвета `black`, `white`, `red`, `green`, `blue`, `cyan`, `magenta` и `yellow`,

Для определения произвольного цвета существует команда `\definecolor{имя}{модель}{описание}`

Пример .68 (определение произвольного цвета). `\definecolor{MyBlue}{rgb}{.6,.8,1}`

Цвет `RoyalBlue` (`rgb(65,105,225)`) определён только в драйвере `dvips`.

```
\definecolor{RBlue}{rgb}{.255,.412,.88}
```

5.2. Использование цветов

Для задания цвета текста имеется декларация `\color{цвет}` или `\color[модель]{описание}` и команда `\textcolor{цвет}{текст}` или `\textcolor[модель]{описание}{текст}`.

Цвет страницы изменяет декларация `\pagecolor{цвет}`, где вместо имени *цвета* также можно задать цвет используя его *описание* в к.-л. цветовой *модели*.

Цветные текстовые боксы задаются командами `\colorbox{цвет фона}{текст}` и `\fcolorbox{цвет рамки}{цвет фона}{текст}`, которые действуют и регулируются как команда `\fbox`.

Цвета фона и *рамки* также можно указывать явно, в описаниях цветовой *модели*.

Пример .69 (определение произвольного цвета). colorPage.ps

```
{\pagecolor[named]{Salmon} \color{cyan}Текст  
цвета \colorbox[named]{OliveGreen}{\texttt{cyan}}  
с вкраплением \textcolor{green}{\tt{зёленого}}  
на фоне цвета \fcolorbox{green}{cyan}%  
\color[named]{Salmon}Salmon}. }
```

Для более сложной работы с цветом можно использовать пакет для цветных таблиц (`colortbl`) и пакеты для подготовки слайдов (`seminar`), а лучше — класс `beamer`).