



Л. Сиденко

# Компьютерная графика и геометрическое моделирование



Москва · Санкт-Петербург · Нижний Новгород · Воронеж  
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск  
Киев · Харьков · Минск  
**2009**

ББК 32.973.2-044я7

УДК 004.92(075)

С34

**Сиденко Л. А.**

С34      Компьютерная графика и геометрическое моделирование: Учебное пособие. — СПб.: Питер, 2009. — 224 с.: ил. — (Серия «Учебное пособие»).

ISBN 978-5-388-00339-3

В учебном пособии представлены принципы построения систем компьютерной графики, рассмотрены алгоритмы создания векторных и растровых изображений. Большое внимание уделено геометрическому моделированию, получению трехмерных моделей, способам работы с ними. Рассмотрены современные подходы к визуализации графической информации и получению реалистичных изображений сложных трехмерных сцен.

Книга предназначена для студентов, изучающих компьютерную графику (курс «Геометрическое моделирование» в технических вузах), для аспирантов и преподавателей, а также для специалистов, работающих в области программирования, дизайна, САПР.

ББК 32.973.2-044я7

УДК 004.92(075)

Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

# **Краткое содержание**

---

Введение .....	10
От издательства .....	11
<b>Часть I. Компьютерная графика .....</b>	<b>12</b>
<b>Глава 1.</b> Общие сведения о компьютерной графике.....	13
<b>Глава 2.</b> Геометрические преобразования .....	27
<b>Глава 3.</b> Алгоритмы растровой графики .....	42
<b>Глава 4.</b> Отсечение линий .....	62
<b>Глава 5.</b> Проектирование графического диалога .....	73
<b>Часть II. Геометрическое моделирование.....</b>	<b>80</b>
<b>Глава 6.</b> Общие сведения о геометрическом моделировании.....	81
<b>Глава 7.</b> Двумерное моделирование .....	88
<b>Глава 8.</b> Трехмерное моделирование.....	92
<b>Глава 9.</b> Описание и характеристика поверхностей .....	105
<b>Глава 10.</b> Получение реалистичных изображений .....	122
<b>Глава 11.</b> Проецирование .....	125
<b>Глава 12.</b> Алгоритмы удаления скрытых линий и поверхностей.....	139
<b>Глава 13.</b> Свет в компьютерной графике .....	154
<b>Глава 14.</b> Трассировка лучей.....	188
<b>Глава 15.</b> Цвет в компьютерной графике.....	200
<b>Глава 16.</b> Сжатие графических изображений .....	208
Список литературы.....	219
Адреса сайтов.....	220

# **Оглавление**

---

Введение .....	10
От издательства .....	11
<b>Часть I. Компьютерная графика .....</b>	<b>12</b>
<b>Глава 1. Общие сведения о компьютерной графике .....</b>	<b>13</b>
1.1. История развития компьютерной графики.....	13
История развития методов вывода.....	13
Развитие методов ввода .....	16
Развитие программного обеспечения .....	16
1.2. Основные сведения о графических системах .....	16
1.3. Функции графических систем .....	17
1.4. Блок-схема графической системы .....	18
1.5. Получение изображения на векторном дисплее .....	19
Генератор векторов.....	19
Относительные координаты .....	20
Генератор символов.....	21
1.6. Получение изображения на растровом дисплее .....	23
<b>Глава 2. Геометрические преобразования .....</b>	<b>27</b>
2.1. Двумерные преобразования .....	27
Перенос .....	27
Масштабирование.....	28
Поворот.....	29
2.2. Однородные координаты и двумерные преобразования .....	30
Перенос .....	30
Масштабирование.....	31
Поворот.....	32
2.3. Композиции двумерных преобразований .....	33
2.4. Матричное представление трехмерных преобразований .....	34
Перенос .....	34
Масштабирование.....	35
Поворот.....	35
2.5. Композиции трехмерных преобразований .....	36
2.6. Преобразования как изменение систем координат.....	39

<b>Глава 3. Алгоритмы растровой графики.....</b>	42
3.1. Преобразование отрезков из векторной формы в растровую .....	42
Пошаговый алгоритм.....	42
Алгоритм Брезенхема .....	43
Ускорение алгоритма Брезенхема .....	45
3.2. Растровая развертка литер .....	46
3.3. Растровая развертка окружностей.....	46
Четырехсторонняя симметрия.....	47
Восьмисторонняя симметрия .....	47
Алгоритм Брезенхема для окружностей.....	47
3.4. Растровая развертка эллипсов .....	50
Простой метод.....	50
Инкрементивный метод.....	51
3.5. Методы устранения искажений в растровых изображениях .....	51
Лестничный эффект.....	52
Мелкие и движущиеся объекты.....	53
3.6. Сглаживание линий. Алгоритм Ву .....	54
3.7. Заполнение области .....	56
Алгоритм построчного сканирования.....	56
Метод заполнения с затравкой.....	57
Заполнение линиями .....	58
3.8. Разложение в растр сплошных многоугольников .....	58
Когерентность сканирующих строк .....	58
Когерентность ребер .....	60
<b>Глава 4. Отсечение линий.....</b>	62
4.1. Алгоритм Коэна—Сазерленда .....	62
4.2. Алгоритм разбиения средней точкой.....	64
4.3. Трехмерное отсечение отрезков.....	66
4.4. Отсечение многоугольников.....	68
4.5. Отсечение литер .....	71
<b>Глава 5. Проектирование графического диалога .....</b>	73
5.1. Языковая аналогия .....	73
5.2. Языковая модель .....	74
5.3. Обеспечение обратной связи .....	76
5.4. Помощь пользователю .....	77
5.5. Возможность исправления ошибок.....	78

<b>Часть II. Геометрическое моделирование</b>	80
<b>Глава 6. Общие сведения о геометрическом моделировании</b>	81
6.1. Геометрическая модель	81
6.2. Основные виды геометрических моделей	82
6.3. Требования, предъявляемые к геометрическим моделям	86
<b>Глава 7. Двумерное моделирование</b>	88
7.1. Типы данных	88
7.2. Построение базовых элементов	88
Непосредственное задание с использованием выбранного синтаксиса представления	88
С помощью уравнений	88
С помощью ограничений	89
С использованием геометрических преобразований	90
7.3. Примеры моделей	90
Автоматизация черчения	90
Параметризация	90
Цепное кодирование	90
<b>Глава 8. Трехмерное моделирование</b>	92
8.1. Типы данных	92
Представление с помощью границ	92
Представление с помощью дерева	93
8.2. Методы описания трехмерных объектов	94
Описание геометрии объекта с использованием алфавитно-цифрового входного языка	94
Описание объекта в режиме графического диалога	95
Получение модели объекта путем ввода эскизов и восстановления модели по имеющимся проекциям	95
8.3. Методы построения трехмерных моделей	96
Построение кривых и поверхностей	97
Задание гранями (кусочно-аналитическое описание)	97
Кинематический принцип	98
Булевы операции	99
Полигональные сетки	101
Октаэдрические деревья	103
<b>Глава 9. Описание и характеристика поверхностей</b>	105
9.1. Описание поверхностей	105
Параметрическое описание	105

Описание неявными функциями .....	107
Поточечное описание.....	107
9.2. Характеристики поверхностей.....	108
Поверхности первого порядка.....	108
Поверхности второго порядка .....	109
Фрактальные поверхности .....	110
9.3. Моделирование деформации трехмерных полигональных поверхностей в режиме реального времени .....	113
Метод деформации на основе использования неявного задания поверхности объекта .....	113
Метод деформации плоских протяженных объектов .....	113
Метод деформации тела, заданного полигональной сеткой .....	114
9.4. Триангуляция поверхностей.....	115
Области Вороного и триангуляция Делоне.....	117
Алгоритм Рапперта.....	118
Триангуляция монотонных полигонов .....	119
Уровень детализации (LOD) .....	120
<b>Глава 10. Получение реалистичных изображений.....</b>	<b>122</b>
10.1. Методы создания реалистичных изображений .....	122
Перспективные проекции .....	122
Передача глубины яркостью .....	123
Отсечение по глубине.....	123
Динамические проекции .....	123
Удаление скрытых линий и поверхностей.....	123
Стереоскопия.....	123
10.2. Перспективные изображения.....	124
<b>Глава 11. Проектирование .....</b>	<b>125</b>
11.1. Основные виды проекций .....	125
Параллельные проекции .....	126
Центральные проекции.....	128
11.2. Математическое описание прямоугольных проекций .....	129
11.3. Математическое описание косоугольных проекций.....	132
11.4. Математическое описание перспективной проекции.....	134
11.5. Задание произвольных проекций. Видовое преобразование.....	136
<b>Глава 12. Алгоритмы удаления скрытых линий и поверхностей.....</b>	<b>139</b>
12.1. Общие сведения об удалении скрытых линий и поверхностей .....	139
12.2. Алгоритм сортировки по глубине.....	140

12.3. Алгоритм, использующий Z-буфер.....	141
12.4. Алгоритм построчного сканирования .....	142
12.5. Алгоритм разбиения области .....	144
12.6. Алгоритм плавающего горизонта.....	146
12.7. Алгоритм Робертса.....	148
12.8. Алгоритм трассировки лучей .....	149
12.9. Иерархический Z-буфер.....	151
<b>Глава 13. Свет в компьютерной графике .....</b>	<b>154</b>
13.1. Общие сведения о свете.....	154
13.2. Модель освещения .....	157
Свойства объектов .....	157
Диффузное отражение .....	158
Зеркальное отражение.....	160
Пропускание света (прозрачность) .....	161
Специальные модели .....	163
13.3. Закраска полигональных сеток .....	164
Однотонная закраска .....	164
Интерполяция интенсивностей (метод Гуро) .....	164
Интерполяция векторов нормали (метод Фонга).....	166
13.4. Тени .....	166
Источник света в бесконечности.....	167
Локальный источник .....	168
13.5. Фактура. Нанесение узора .....	169
Нанесение узора на поверхность. Регулярная текстура.....	169
Нанесение узора на поверхность. Стохастическая текстура .....	171
13.6. Создание неровностей на поверхности.....	174
Рельефное текстурирование .....	175
Метод возмущения нормали .....	176
Использование фрактальных поверхностей .....	177
Использование карт смещения .....	178
13.7. Фильтрация текстур.....	179
13.8. Полутоновые изображения .....	185
<b>Глава 14. Трассировка лучей .....</b>	<b>188</b>
14.1. Метод прямой трассировки .....	189
14.2. Метод обратной трассировки .....	190
14.3. Дискретная трассировка лучей в октантных деревьях.....	196

<b>Глава 15. Цвет в компьютерной графике .....</b>	200
15.1. Ахроматический и хроматический цвета .....	200
15.2. Цветовые модели .....	202
Системы смешивания основных цветов .....	203
Цветовая модель HSV .....	204
Цветовая модель HSL.....	205
Цилиндрическая цветовая модель .....	206
15.3. Цветовая гармония.....	206
<b>Глава 16. Сжатие графических изображений.....</b>	208
16.1. Графические форматы .....	208
BMP .....	208
TIFF .....	209
GIF.....	209
PSD.....	210
PDF .....	210
JPEG .....	210
16.2. Основные сведения о сжатии изображений.....	211
16.3. Алгоритмы сжатия файлов без потерь.....	211
Алгоритм Хаффмана.....	211
Алгоритм Лемпеля–Зива (LZW) .....	212
Алгоритм RLE (Run Length Encoding) .....	212
CCITT Group 3, CCITT Group 4.....	213
Обрезка «хвостов» .....	213
16.4. Сжатие с потерями цветных и полутоночных файлов.....	213
Сжатие изображения по стандарту JPEG .....	214
Новый стандарт JPEG 2000.....	216
Фрактальное сжатие изображений.....	217
<b>Список литературы.....</b>	219
<b>Адреса сайтов.....</b>	220

## **Введение**

---

«Потенциальные возможности компьютерной графики грандиозны, ограничения же зависят только от нашей фантазии. И чем она богаче, тем полнее раскрываются возможности компьютерной графики». Этими словами великого американского ученого Ликлайдера, сказанными много лет назад, хотелось бы начать путешествие в мир компьютерной графики.

Еще до недавнего времени компьютерная графика представляла собой весьма редкое занятие, требующее дорогостоящей аппаратуры, значительных ресурсов памяти и своеобразного программного обеспечения, что могла себе позволить далеко не каждая фирма.

В настоящее время ситуация сильно изменилась. Сформировалась новая отрасль информатики — компьютерная графика. Ее можно определить как науку о математическом и геометрическом моделировании форм и размеров объектов, а также методах их визуализации.

Интерес к синтезу изображений объясняется высокой информативностью. Информация, содержащаяся в изображении, представлена в наиболее наглядной форме, не требующей слов, и эта информация, как правило, более доступна для восприятия.

Стремление визуализировать информацию наблюдается практически во всех сферах деятельности человека. С 1960-х годов, которые считаются началом зарождения компьютерной графики, до наших дней пройден большой путь. Сегодня с экрана дисплея можно наблюдать не только графики, диаграммы, чертежи, но и полноценные динамически изменяющиеся трехмерные объекты, соответствующие реальным объектам как по форме и расположению, так и по цвету, фактуре, освещению. Это вызывает все больший интерес к компьютерной графике, которая применяется для решения большого класса задач, а также в области развлечений. Большинство специалистов, использующих компьютерную графику, уделяют особое внимание проблемам программирования, задачам конкретного проектирования, создания различных технических средств.

Целью данного учебного пособия является изложение математических методов и технических аспектов, лежащих в основе компьютерной графики. В книге используется единая система обозначений. Показаны связи между отдельными этапами на пути к получению высокореалистичных трехмерных сцен. Подробно изложены принципы построения графических систем, способы преобразования объектов, методы описания кривых и поверхностей. Особое внимание уделено геометрическому моделированию, так необходимому при проектировании технических объектов. Кроме того, представлена новейшая информация по визуализации объектов, получению реалистичных изображений, использованию освещения, теней, наложению рисунка и фактуры.

## **От издательства**

---

Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты [dgurski@minsk.piter.com](mailto:dgurski@minsk.piter.com) (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На сайте издательства <http://www.piter.com> вы найдете подробную информацию о наших книгах.

# **Часть I**

## **КОМПЬЮТЕРНАЯ ГРАФИКА**

---

Первая часть посвящена вопросам получения графического изображения с помощью компьютера. Рассмотрены векторные и растровые дисплеи, способы формирования изображения на них, приведена сравнительная характеристика. Особое внимание уделено алгоритмам растровой графики — разложению примитивов в растровую форму и устранению искажений в растровых изображениях.

Кроме того, рассмотрены геометрические преобразования, лежащие в основе всех операций по изменению объектов, и определены однородные координаты. Также уделено внимание отсечению примитивов по прямоугольной области и приведены принципы организации графического интерфейса.

# **Глава 1**

## **Общие сведения о компьютерной графике**

---

Данная глава знакомит с общей информацией о компьютерной графике: историей развития, способами получения изображений на экране компьютера, особенностями векторной и растровой графики. Также рассматриваются функции графических систем, с помощью которых получаются графические изображения.

### **1.1. История развития компьютерной графики**

При всем великолепии изображений, полученных с помощью компьютера, и впечатляющих эффектов, которые мы можем сегодня наблюдать, все начиналось совсем не так легко. Компьютерная графика пробивалась через терни технических ограничений и зарождающегося нового мышления. Прошло немногим более полувека ее существования, а добилась она за это время высоких вершин. И продолжает покорять их с большой скоростью, неведомой многим другим ее собратьям. За время своей жизни компьютерная графика находит все больше и больше ценителей и друзей.

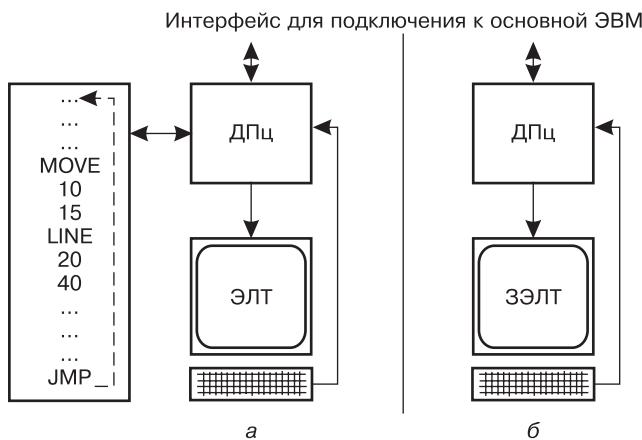
Рождение компьютерной графики можно отнести к середине XX века, когда появилась минимальная возможность получать незатейливые рисунки с помощью компьютера. Чаще всего это были графики и простые схемы. Простейшие графические программы были привязаны к конкретному компьютеру, что не позволяло передавать графические изображения и тем самым мешало развитию. Процесс получения компьютерных изображений был весьма дорогим, так как требовал достаточно объема памяти для хранения графических данных и регенерации для получения динамических изображений. Поэтому лишь немногие и особо богатые фирмы могли позволить себе эту роскошь. Первыми, кто использовал компьютерную графику, были состоятельные организации и, конечно, военная промышленность.

### **История развития методов вывода**

Первые дисплейные устройства, которые были разработаны в 1960-х годах, назывались *векторными*.

Векторные дисплеи состоят из трех основных частей (рис. 1.1):

- дисплейный процессор (ДПц);
- буферная память;
- электронно-лучевая трубка (ЭЛТ).



**Рис. 1.1.** Устройство дисплея: а — векторного; б — запоминающего<sup>1</sup>

Буфер служит для запоминания подготовленного дисплейного списка (или дисплейной программы). Дисплейная программа включает команды вывода точек, отрезков, символов. Эти команды интерпретируются дисплейным процессором, который преобразует цифровые значения в аналоговые напряжения, управляющие электронным лучом. Луч вычерчивает линии на люминофорном покрытии электронно-лучевой трубы. Полученное таким образом изображение не может храниться долго, так как светоотдача люминофора падает до нуля за несколько микросекунд. Поэтому изображение нужно обновлять. Данный процесс называется *регенерацией*. Частота регенерации должна быть не меньше 25 раз в секунду, чтобы глаз человека не наблюдал мерцание. В связи с этим буфер, в котором хранится дисплейная программа, называют буфером регенерации (см. рис. 1.1, а).

Команда перехода (JMP) обеспечивает возврат к началу дисплейной программы с целью обеспечения циклической регенерации.

В 1960-х годах буферная память большого объема и быстрые дисплейные процессоры с частотой регенерации не меньше 30 Гц были очень дорогими. В связи с этим для обеспечения доступности компьютерной графики в конце 1960-х годов были созданы запоминающие электронно-лучевые трубы (ЗЭЛТ), которые позволили отказаться от буфера и регенерации (рис. 1.1, б).

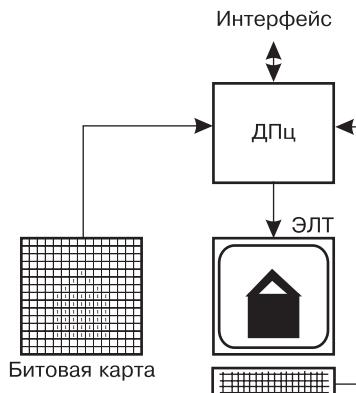
В таких электронно-лучевых трубках изображение запоминается путем его однократной записи на запоминающую сетку с люминофором медленно движущимся электронным лучом. Запоминающие трубы применяются в тех случаях, когда нужно вывести большое количество отрезков и литер и когда нет необходимости в динамических операциях с изображением.

В середине 1970-х годов при интенсивном внедрении телевизионной техники была изобретена *растровая графика*, которая по своей стоимости была намного ниже графики, полученной на векторном дисплее. В растровых дисплеях примити-

<sup>1</sup> Александров В. В., Шнейдеров В. С. Рисунок, картина, чертеж на ЭВМ. — Ленинград: Машиностроение, 1988.

вы хранятся в памяти для регенерации в виде совокупности образующих их точек, называемых *пикселями*. Значения пикселов хранятся в битовой карте, которая и является в данном случае дисплейной программой. Изображение, формируемое на растре, представляет собой совокупность горизонтальных растровых строк, каждая из которых состоит из отдельных пикселов.

Таким образом, *растр* — это матрица пикселов, покрывающая всю площадь экрана. Все изображение последовательно сканируется около 30 раз в секунду по отдельным строкам растра в направлении сверху вниз (рис. 1.2).



**Рис. 1.2.** Устройство растрового дисплея<sup>1</sup>

Растровые дисплеи обладают следующими достоинствами:

- растровая графика по сравнению с векторной легче закрашивает изображения. В векторных дисплеях необходимость закраски области сильно увеличивает размер дисплейной программы, так как закраска происходит векторами. В растровом же дисплее эта процедура не влияет на размер дисплейной программы, так как меняются лишь значения внутренних пикселов;
- процесс регенерации не зависит от сложности рисунка. Векторные же дисплеи часто начинают мерцать, когда количество примитивов в буфере становится таким большим, что его нельзя считать и обработать за время, отведенное на считывание одного кадра, в результате чего требуется больше времени, а это может понизить частоту регенерации изображения;
- дешевизна.

При указанных выше достоинствах растровые дисплеи имеют и недостатки:

- необходимо больше памяти, так как полное изображение, состоящее из пикселов, должно храниться как битовая карта. В векторном дисплее вектор задается двумя точками, в растровом необходимо хранение и всех промежуточных;
- разрешающая способность растровых графических систем пока еще ниже, чем векторных ( $1280 \times 1024$  против  $4096 \times 4096$  пикселов);

<sup>1</sup> Климов В. Е. Графические системы САПР. — М.: Высшая школа, 1990.

- для отображения примитивов необходимо больше времени, так как в векторных дисплеях задаются две конечные точки (для отрезка), а в растровом нужно рассчитать еще и все промежуточные точки отрезка.

### **Развитие методов ввода**

Параллельно с совершенствованием методов вывода улучшались и методы ввода. Громоздкое и хрупкое световое перо вытесняется тонкой указкой, которую перемещают по электронному планшету, или же смонтированной на экране прозрачной сенсорной панелью, реагирующей на прикосновение. Кроме того, большие надежды возлагаются на речевую связь, которая помогает вводить информацию без помощи рук и выводить ее в естественном виде.

### **Развитие программного обеспечения**

Многие трудности в развитии графического программного обеспечения были связаны с его примитивностью. Процесс совершенствования программного обеспечения был длительным и медленным. Был пройден путь от аппаратно-зависимых пакетов низкого уровня, поставляемых изготовителями вместе с конкретными дисплеями, к аппаратно-независимым пакетам высокого уровня. Такие пакеты могут быть использованы для управления самыми разнообразными графическими устройствами. Основная цель аппаратно-независимого пакета — обеспечение мобильности прикладной программы при переходе от одного компьютера к другому.

## **1.2. Основные сведения о графических системах**

Процессы проектирования в различных областях техники связаны с созданием и модификацией моделей объектов проектирования. Значительную часть этих моделей составляют данные о геометрических или графических характеристиках объектов.

Таким образом, *компьютерная графика* — это область деятельности, в которой компьютеры используются для создания, хранения и обработки моделей объектов и их изображений.

Графические системы служат для создания, поиска, хранения и модификации графических данных.

Графические системы могут быть пассивными и интерактивными.

- *Пассивные* — обеспечивают только вывод графических изображений, но человек при этом не имеет возможности прямого воздействия на графические преобразования.
- *Интерактивные* — дают возможность пользователю динамически управлять содержимым изображения. В таких системах используются интерактивные дисплеи, позволяющие работать в диалоге с графическим изображением.

Области применения графических систем представлены в табл. 1.1.

**Таблица 1.1. Области применения графических систем**

<b>Область применения</b>	<b>Синтез изображения</b>	<b>Анализ изображения</b>	<b>Обработка изображения</b>
Вход	Формальное описание	Визуальное представление	Визуальное представление
Выход	Визуальное представление	Формальное описание	Визуальное представление
Объект	Линии, пиксели, объекты, текст	Сгенерированное или сканируемое изображение	Сканируемое изображение
Задачи	Генерация, преобразование изображения	Распознавание образов, структурный анализ, анализ сцен	Повышение качества изображения

Синтез изображения позволяет получать на базе описания объекта пользователем его геометрическую модель с последующим ее отображением. Анализ изображения выполняет обратную задачу — на базе уже имеющегося графического изображения, сгенерированного ранее с помощью графической системы или отсканированного, дает формальное описание. Для этого необходимы специальные алгоритмы, выполняющие или распознавание образов, или структурный анализ, или анализ сцен в трехмерной графике.

### **1.3. Функции графических систем**

Интерактивные графические системы выполняют следующие функции:

- ввод данных;
- вывод графических изображений;
- обработка запросов пользователя;
- поиск и хранение данных;
- реализация преобразований графической информации.

Функции ввода реализуются с помощью графических устройств ввода: клавиатуры, планшета, мыши, светового пера и т. д.

Функции вывода — с помощью графических устройств вывода: графопостроителя, дисплея, станка с ЧПУ.

Функции обработки запросов пользователя на входных и командных языках реализуются программой, называемой лингвистическим (диалоговым) процессором. Процессор преобразует описания геометрии объектов, заданные на входных языках, в формы, принятые в системе. В настоящее время наиболее эффективный метод работы пользователя с графической системой — диалог с использованием меню. Данные, получаемые системой через диалоговый процессор, делятся на два класса: параметры объекта и коды для управления графической системой. Первые поступают из входных языков, вторые — из командных. Параметры объекта направляются через СУБД в базу данных. Коды для управления графической системой поступают в монитор. Он управляет работой системы.

Организация базы данных графической системы определяется классами моделей объектов. Если объекты проектирования имеют графическое представление (схемы, планы, чертежи), в базе данных хранятся модели графических изображений этих объектов. Ориентация системы на объект определяет наличие в базе данных геометрических моделей объектов в трехмерном пространстве.

Формирование моделей и их модификаций, а также преобразование этих моделей выполняет геометрический процессор. В зависимости от сложности модели объекта в системе может исполняться несколько геометрических процессоров.

Геометрический процессор может выполнять и следующие функции:

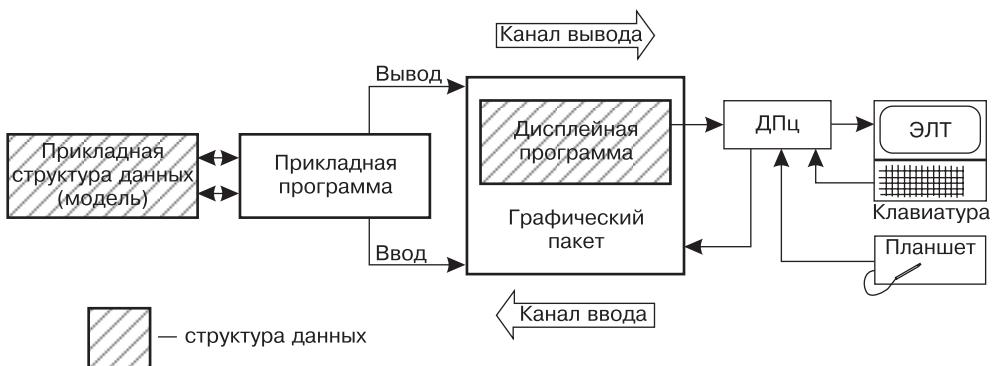
- построение сечений и разрезов;
- проверка корректности геометрической компоновки узла конструкции;
- моделирование работы робота.

Для систем, работающих с двумерными геометрическими объектами, функции формирования, модификации и преобразования геометрической модели выполняет графический процессор.

## 1.4. Блок-схема графической системы

Блок-схема графической системы приведена на рис. 1.3. В памяти размещаются два важных информационных модуля:

- прикладная структура данных, содержащая описание объектов, изображения которых должны показываться на экране. Она же является моделью объектов;
- дисплейная программа, которая формируется графическим пакетом и читается дисплейным процессором во время регенерации изображения на экране.



**Рис. 1.3.** Блок-схема графической системы<sup>1</sup>

**Канал вывода** (от описания объекта к его изображению).

Прикладная программа извлекает информацию из прикладной структуры данных и записывает ее, а затем направляет графические команды, которые обраба-

<sup>1</sup> Александров В. В., Шнейдеров В. С. Рисунок, картина, чертеж на ЭВМ. — Ленинград: Машиностроение, 1988.

тываются графическим пакетом. Последний формирует дисплейную программу, используемую дисплейным процессором для получения изображения. Таким образом, канал вывода последовательно преобразует описание объекта в структуру представления, принятую в дисплейной программе.

**Канал ввода** (от устройств ввода к структуре данных и дисплейной программе).

Дисплейный процессор регистрирует факт использования устройства ввода и либо прерывает, либо передает данные по запросу. Ввод данных от дисплейного процессора осуществляется специальная программа ввода, которая передает их прикладной программе. Эти данные меняют состояние прикладной программы. Они могут также побудить прикладную программу модифицировать структуру данных, изменить параметры.

## 1.5. Получение изображения на векторном дисплее

В векторных дисплеях электронный луч, управляемый дисплейным процессором, создает изображение, двигаясь от точки к точке по отрезкам прямых, которые называются *векторами*. Для получения изображения необходимы хранение координат точек и генератор векторов. С помощью генератора векторов можно рисовать прямолинейные участки символов и осуществлять кусочно-линейную аппроксимацию кривых. При рисовании кривой необходимо учитывать следующую особенность: в областях, где кривая имеет малый радиус кривизны, отрезки прямых должны быть короче, чем в областях, где радиус кривизны относительно велик.

Максимальная скорость вычерчивания векторов — около 2 см/мкс. В лучших системах за один цикл регенерации выводится 6–10 тыс. векторов длиной 2–3 см.

### Генератор векторов

Генератор векторов должен выполнять следующие действия.

1. Перемещать электронный луч из позиции  $(x_{\text{пред}}, y_{\text{пред}})$  в позицию  $(x, y)$  по прямой линии или, по крайней мере, по линии, которая кажется наблюдателю прямой.
2. Рисовать все линии с одинаковой яркостью. Чем медленнее движется луч, тем ярче светится получаемая линия. Постоянная яркость может быть достигнута либо перемещением луча с постоянной скоростью для всех векторов, либо изменением интенсивности луча для разных векторов.
3. Включать луч точно в тот момент, когда он начинает перемещаться из начальной точки, и выключать в момент прихода в конечную точку.
4. Осуществлять все операции как можно быстрее.

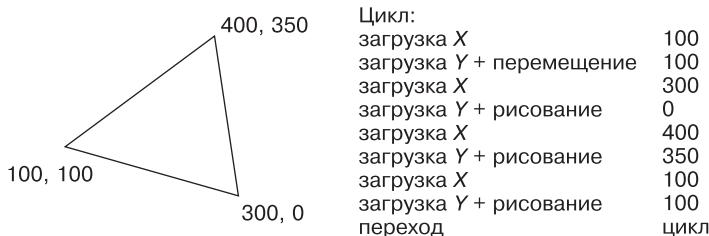
Генератор векторов должен управлять тремя параметрами:

- отклонением по оси X;
- отклонением по оси Y;
- интенсивностью.

Если векторы рисуются с одной и той же скоростью, сигнал управления интенсивностью превращается просто в сигнал его включения и выключения. Для изображения прямой линии, проходящей от точки  $(x_1, y_1)$  к точке  $(x_2, y_2)$ , необходимо дополнить массивы  $X$  и  $Y$  координатами точек, лежащих приблизительно на этой прямой. Приблизительно, потому что для некоторого целого  $x$  соответствующее ему  $y$  может оказаться нецелым. Поэтому возникает необходимость в округлении или отсечении.

### Относительные координаты

Рассмотрим дисплейную программу, изображающую треугольник (рис. 1.4).



**Рис. 1.4.** Пример дисплейной программы, изображающей треугольник<sup>1</sup>

Все идет хорошо до тех пор, пока в прикладной программе не возникнет необходимость переместить треугольник в другую точку экрана, например на  $\Delta x = -50$ ,  $\Delta y = 100$ . Существуют два способа решения этой задачи.

Первый — прикладная программа находит описание треугольника в прикладной структуре данных, изменяет значения координат вершин и вызывает графический пакет для создания новой дисплейной программы.

Или второй, более эффективный. Прикладная программа выдает графическому пакету запрос на изменение сегмента дисплейной программы, который генерирует треугольник. Изменение заключается в вычитании 50 из всех  $x$ -координат и прибавлении 100 ко всем  $y$ -координатам. Для треугольника эта процедура будет выполнена быстро, но для сложных объектов она будет более трудоемкой.

Еще проще реализуется перемещение, когда координаты объекта можно задавать абсолютными и относительными величинами. Дисплейная программа имеет вид, представленный на рис. 1.5.

Цикл:	
загрузка X абс.	100
загрузка Y абс. + перемещение	100
загрузка X отн.	200
загрузка Y отн. + рисование	-100
загрузка X отн.	100
загрузка Y отн. + рисование	350
загрузка X отн.	-300
загрузка Y отн. + рисование	-250
переход	цикл

**Рис. 1.5.** Пример дисплейной программы (координаты объекта заданы абсолютными и относительными величинами)

<sup>1</sup> Александров В. В., Шнейдеров В. С. Рисунок, картина, чертеж на ЭВМ. — Ленинград: Машиностроение, 1988.

Таким образом, для перемещения треугольника достаточно модифицировать значения начальных координат  $x$  и  $y$ .

### Генератор символов

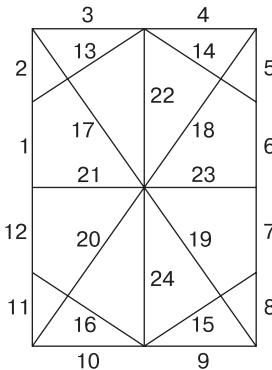
Существуют четыре способа генерации символов:

- метод маски;
- метод Лиссажу;
- штриховой метод;
- метод точечной матрицы.

### Метод маски

Луч ЭЛТ описывает фигуру, как на рис. 1.6 (24 штриха). Символ генерируется путем высвечивания одних штрихов и пропуска других. Символ кодируется 24-разрядным словом, в котором каждый разряд представляет один штрих.

Если в разряде 0, то штрих пропускается, если 1, то штрих высвечивается.



**Рис. 1.6.** Метод маски

Недостатки метода:

- низкое качество символов;
- низкая скорость, так как луч должен пойти по всей маске для любого символа;
- необходима перекодировка из ASCII в специальный 24-разрядный код.

### Метод Лиссажу

Для формирования символов используются фигуры Лиссажу. Этот способ получения символов является аналоговым. Различные возможности генерирования примитивов (отрезков и дуг), из которых формируются символы, показаны на рис. 1.7.

Достиныства метода:

- хорошее качество, плавность линий, что особенно важно для строчных букв;
- любые формы символов.

Недостатки метода:

- низкая скорость;
- высокая стоимость аппаратной части.

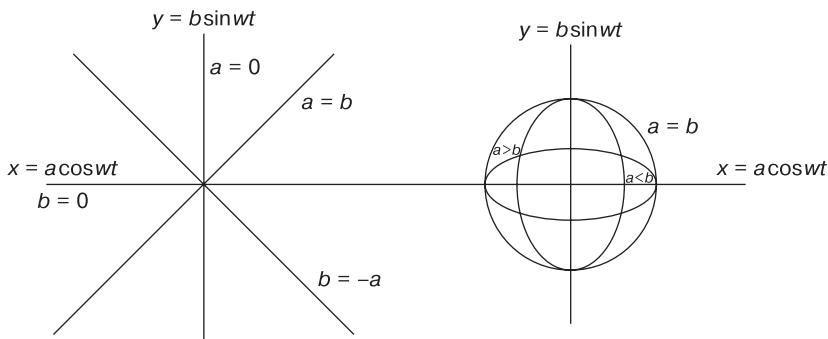


Рис. 1.7. Метод Лиссажу

**Штриховой метод**

Штриховой генератор символа представляет собой аналоговое устройство, которое выдает волны различной формы отдельно для отклонения луча ЭЛТ по направлениям  $X$  и  $Y$ , а также сигнал яркости в виде «вкл./выкл.» (рис. 1.8).

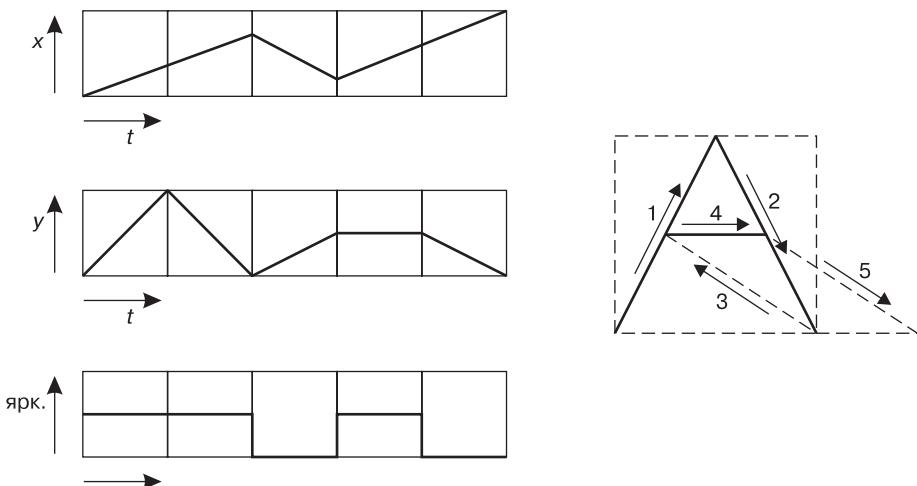


Рис. 1.8. Штриховой метод

Достоинства метода:

- высокая скорость (1 символ  $\approx 10$  мкс);
- хорошее качество;
- можно задавать любые размеры символов.

Недостатком указанного метода является его высокая стоимость.

**Метод точечной матрицы**

Генератор символов с точечной матрицей является чисто цифровым устройством, поэтому он надежный и недорогой. Символы будут удовлетворительного

качества при условии, что они малы. Поэтому они часто используются в алфавитно-цифровых дисплеях.

Символ генерируется путем высвечивания последовательности точек. Луч ЭЛТ последовательно проходит через поля матрицы символа. В каждом поле матрицы луч может быть либо включен, либо выключен (рис. 1.9).

Для символа  $5 \times 7$  необходимо 35 бит.

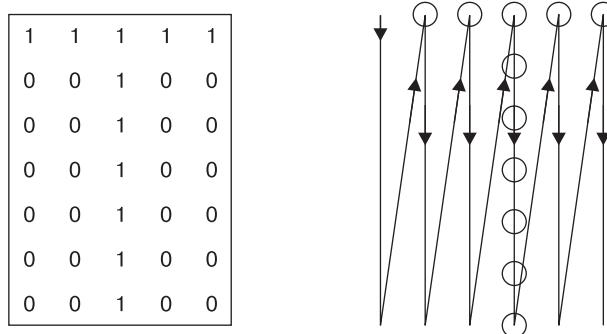


Рис. 1.9. Метод точечной матрицы

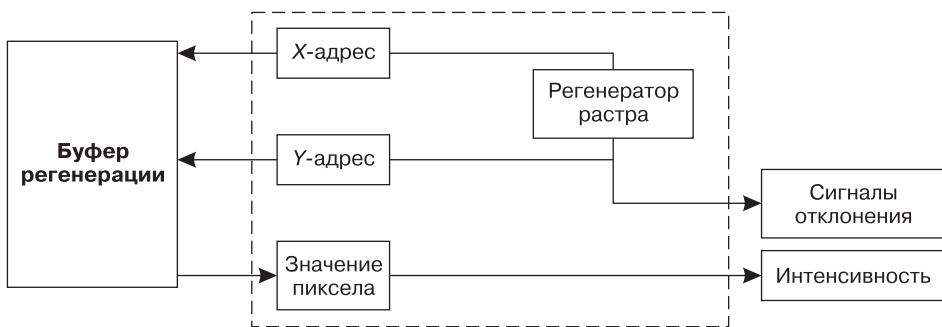
Достоинством данного метода является низкая стоимость, а недостатком — плохое качество при увеличении размера букв.

## 1.6. Получение изображения на растровом дисплее

В системах с растровым сканированием кодирование изображения проще, чем в рассмотренных выше. Рисуемые на экране примитивы разбиваются на составляющие их точки. Основное различие между векторным и растровым дисплеями состоит в организации памяти, хранящей точки. В векторном дисплее точки, составляющие каждый примитив, хранятся в памяти последовательно и рисуются в этом же порядке, примитив за примитивом, так как луч может двигаться по экрану произвольным образом. В растровом дисплее память регенерации организована в виде двумерного массива. Элемент, находящийся на пересечении строки и столбца, хранит значение яркости или цвета соответствующей точки экрана. Верхняя строка массива соответствует верхней строке растра. Регенерация изображения осуществляется последовательным сканированием буфера по строкам растра.

Простой буфер регенерации содержит по 1 биту на пикセル и, таким образом, определяет двухцветное (черно-белое) изображение. Задача системы вывода изображения состоит в циклическом просмотре буфера регенерации по строкам, обычно от 30 до 100 раз в секунду. Адреса памяти генерируются синхронно с координатами растра, и содержимое выбранных элементов памяти используется для управления интенсивностью электронного луча (рис. 1.10).

Генератор растровой развертки формирует сигналы отклонения и управляет адресными  $X$ - и  $Y$ -registрами, определяющими следующий элемент буфера регенерации.



**Рис. 1.10.** Система вывода изображения на растровый дисплей

В начале цикла регенерации в  $X$ -регистр записывается 0, а в  $Y$ -регистр —  $N - 1$  (верхняя строка раstra). Пока регенерируется первая строка раstra,  $X$ -адрес точки увеличивается до  $M - 1$ , при этом значение каждого пикселя извлекается из памяти и используется для управления интенсивностью электронного луча. После генерации первой строки раstra в  $X$ -адрес опять записывается 0, а  $Y$ -адрес уменьшается на 1. Этот процесс продолжается до генерации последней строки раstra ( $y = 0$ ). По окончании сканирования раstra система вырабатывает сигнал прерывания. Компьютер может произвести изменения в хранящемся в памяти изображении. Для этого у него есть время обратного хода (около 1 мс), в течение которого электронный луч перемещается из правого нижнего угла в левый верхний.

Каждый полный проход сканирующего луча по экрану дает неподвижную картинку, или *кадр*. За одну секунду генерируется от 30 до 100 таких кадров. Вследствие генерации зрительного восприятия последовательность постепенно меняющихся кадров может передавать движение плавно, без скачков. Однако даже при такой частоте картинка кадра на экране немного мерцает. Этот нежелательный эффект обычно устраняют чередованием строк каждой пары последовательных кадров.

При реализации метода чередования строк на экран выводится, например, не 30, а 60 кадров за одну секунду, то есть в два раза больше. Электронный луч проходит по нечетным строкам, формируя первый кадр, второй же кадр после возвращения луча в левый верхний угол экрана создается сканированием по четным строкам.

Для получения двухцветного, например черно-белого, изображения достаточно одного бита на пиксель. Он принимает значение 0 (пиксель выключен) или 1 (пиксель включен). Двухцветных изображений обычно недостаточно, и возникает необходимость использовать яркость (в случае двухцветного полутонового изображения) или цвет. Дополнительные возможности можно получить, храня для каждого пикселя несколько бит (два бита — четыре яркости и т. д.). Эти биты могут использоваться для управления не только яркостью, но и цветом.

Сколько бит должен содержать пиксель, чтобы изображение воспринималось наблюдателем как имеющее непрерывные уровни тона? Часто бывает достаточно 5 или 6, но может потребоваться и 8. Таким образом, можно будет по-

лучить соответственно  $2^5$ ,  $2^6$  или  $2^8$  тонов яркости. Для цветных дисплеев требуется в три раза больше бит — по 8 на каждый из основных цветов (красный, зеленый и синий).

Второй путь — включение в состав системы регенерации растровых дисплеев таблицы цветов. Значение пикселя не направляется сразу на ЦАП схемы управления яркостью, как в предыдущем случае, а используется как индекс в таблице цветов. Для управления яркостью или цветом применяется значение, извлеченное из таблицы по этому индексу (рис. 1.11).

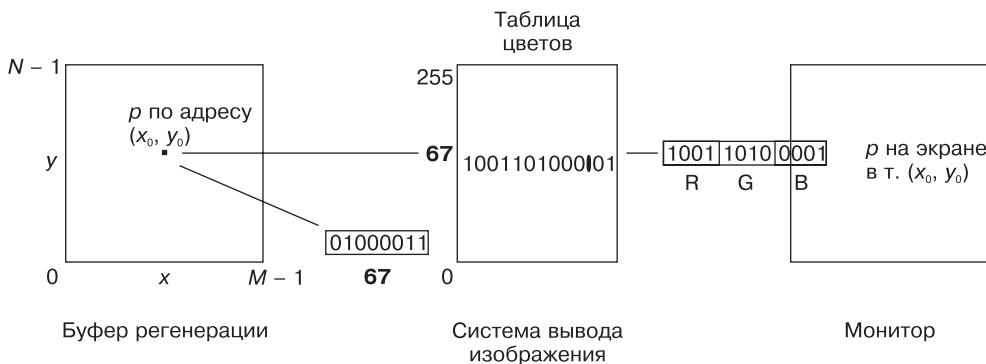


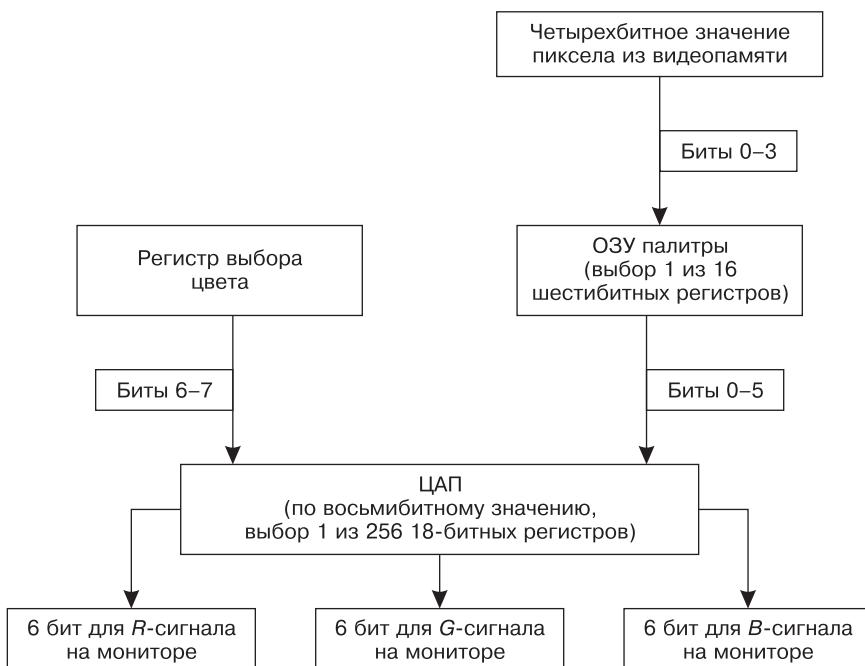
Рис. 1.11. Использование таблицы цветов<sup>1</sup>

Например, значение пикселя 67 привело бы к извлечению из таблицы элемента с номером 67 и использованию его для управления электронным лучом. Эта операция поиска в таблице выполнялась бы для каждого пикселя в каждом цикле регенерации. Поэтому доступ к таблице должен быть очень быстрым. На рис. 1.11 каждый пиксель содержит  $N$ бит ( $N = 8$ ), а элемент таблицы цветов —  $K$ бит ( $K = 12$ ). Тем самым для двухцветных дисплеев в первом случае определены  $2^N$  уровней яркости, а во втором —  $2^K$ . В случае цветной картинки  $K$ бит разбиваются на три равные группы, управляющие красным, зеленым и синим цветами.

В качестве примера можно рассмотреть процесс получения 256 цветов режима VGA (рис. 1.12).

При получении 256 цветов VGA берет четырех- или восьмибитные значения из видеопамяти и переводит их в трех- или шестибитные числа (R, G, B), которые переводятся в аналоговый сигнал, подающийся на монитор. ОЗУ палитры и регистры цифро-аналогового устройства (ЦАП) можно устанавливать либо напрямую, либо через BIOS. Но при одновременной перезагрузке большого блока регистров ЦАП происходит мигание. Эта проблема решается, если один раз переустановить цвета в начале программы. Но при быстром изменении цветов для создания спецэффектов это становится проблемой. Решить ее можно, если менять регистры напрямую. Правильно чередуя цвета, можно легко имитировать эффект плавного движения, не тронув при этом ни один байт дисплейной

<sup>1</sup> Александров В. В., Шнейдеров В. С. Рисунок, картина, чертеж на ЭВМ. — Ленинград: Машиностроение, 1988.



**Рис. 1.12.** Получение 256 цветов

программы. Например, легко изобразить закат, последовательно чередуя цвета от самых ярких внизу к самым темным вверху. Аналогично просто изобразить течение реки. Это удобно при создании реалистичных эффектов трехмерной анимации. ЦАП желательно перезагружать только между кадрами, иначе на экране будет появляться снег. И хотя при однократной установке снежинки можно и не заметить, но при постоянной перезагрузке регистров снег превращается в сильную метель.

## Глава 2

# Геометрические преобразования

---

Данная глава посвящена одной из функций графических систем — преобразованию изображений.

При работе с графическими системами и формировании изображения постоянно возникает необходимость его изменить. В распоряжении пользователя находится большой арсенал команд редактирования: перемещение, поворот, осевая симметрия, подобие, копирование, изменение размеров объекта и многие другие.

### 2.1. Двумерные преобразования

В основе изменения графической информации лежат три основных преобразования: *перенос, масштабирование и поворот*. На их основе строятся все известные изменения объектов в графических системах.

#### Перенос

Точки на плоскости можно перенести в новые позиции путем добавления к координатам этих точек констант переноса. Для каждой точки  $P(x, y)$ , которая перемещается в новую точку  $P'(x', y')$ , сдвигаясь на  $Dx$  единиц по оси  $X$  и на  $Dy$  по оси  $Y$ , можно написать:

$$x' = x + Dx, \quad y' = y + Dy.$$

Определим векторы-строки:

$$P = [x \ y], \quad P' = [x' \ y'], \quad T = [Dx \ Dy].$$

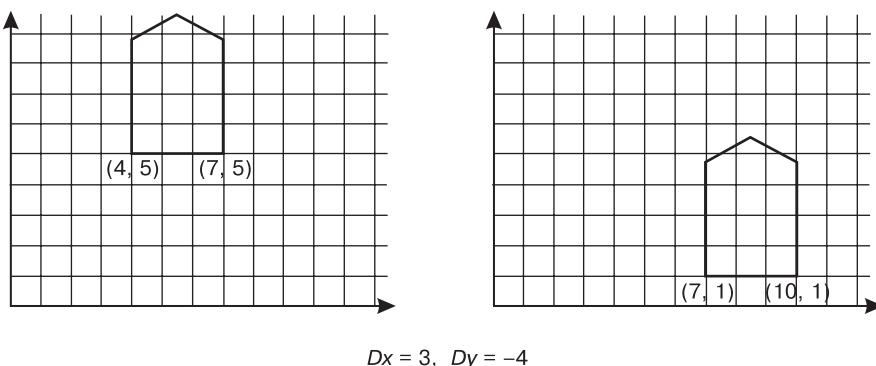
Тогда уравнение:

$$[x' \ y'] = [x \ y] + [Dx \ Dy],$$

или кратко:

$$P' = P + T.$$

Но объект представляет собой множество точек. Его можно переносить, применяя уравнение переноса к каждой точке. Однако каждый отрезок состоит из бесконечного числа точек, и этот процесс длился бы бесконечно долго. Удобнее перенести все точки, принадлежащие отрезку, путем перемещения одних лишь крайних его точек, а затем вычертить новый отрезок между ними (рис. 2.1).

**Рис. 2.1.** Перенос объекта

### Масштабирование

Точки можно масштабировать (растянуть) в  $S_x$  раз по оси  $X$  и в  $S_y$  раз по оси  $Y$ . Получим новые точки с помощью умножения:

$$x' = x \cdot Sx, y' = y \cdot Sy.$$

Определив  $S$  в виде матрицы:

$$S = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix},$$

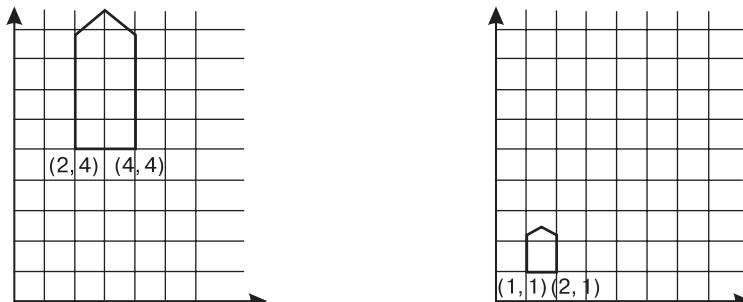
можно записать в матричной форме:

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix},$$

или

$$P' = P \cdot S.$$

Масштабирование производится относительно начала координат (рис. 2.2). В результате домик стал меньше и ближе к началу координат. Если бы масштаб-



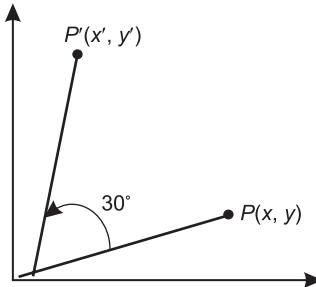
$$S_x = 1/2, S_y = 1/4$$

**Рис. 2.2.** Масштабирование объекта

ные коэффициенты были больше единицы, домик увеличился бы и отдалился от начала координат. Пропорции домика тоже изменились. Было применено неоднородное масштабирование, при котором  $S_x \neq S_y$ . Однородное масштабирование ( $S_x = S_y$ ) не влияет на пропорции.

### Поворот

Точки могут быть повернуты на угол  $\Theta$  относительно начала координат (рис. 2.3).



**Рис. 2.3.** Поворот точки

Тогда координаты точки  $P'$ :

$$\begin{aligned} x' &= x \cos \Theta - y \sin \Theta, \\ y' &= x \sin \Theta + y \cos \Theta. \end{aligned}$$

В матричной форме:

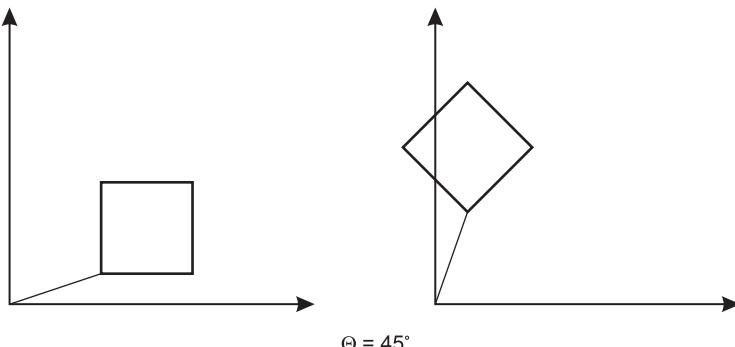
$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} \cos \Theta & \sin \Theta \\ -\sin \Theta & \cos \Theta \end{bmatrix},$$

или

$$P' = P \cdot R,$$

где  $R$  — матрица поворота.

Положительное значение соответствует повороту против часовой стрелки (рис. 2.4), отрицательное — по часовой стрелке.



**Рис. 2.4.** Поворот объекта

## 2.2. Однородные координаты и двумерные преобразования

Преобразования переноса, масштабирования и поворота в матричной форме имеют вид:

$$P' = P + T;$$

$$P' = P \cdot S;$$

$$P' = P \cdot R.$$

Перенос реализуется с помощью операции сложения, а масштабирование и поворот — с помощью умножения. Это вызывает неудобство при осуществлении нескольких преобразований над объектом. Каждую точку объекта придется последовательно подвергать каждому преобразованию. Если объект имеет  $N$  точек и необходимо провести, например, три последовательных преобразования, то понадобится  $3 \cdot N$  действий. Удобно было бы все преобразования представить в единой форме. Тогда можно было бы один раз найти результирующую матрицу преобразования, а затем лишь умножить ее на все точки. В таком случае количество действий стало бы  $N + 2$ , где 2 — это две операции по умножению трех матриц. Если выразить точки в однородных координатах, то все три преобразования можно реализовать с помощью операции умножения. И хотя умножение выполняется аппаратно медленнее сложения и размерность матриц увеличивается на единицу, при большом количестве точек объекта наблюдается ощутимое преимущество.

В однородных координатах точка  $P(x, y)$  записывается как  $P(W \cdot x, W \cdot y, W)$ , где  $W$  — масштабный множитель, не равный нулю.

При этом если точка задана в однородных координатах  $P(X, Y, W)$ , то можно найти ее декартовы координаты:

$$x = \frac{X}{W}, y = \frac{Y}{W}.$$

Если же  $W = 1$ , то операция деления не нужна:

$$P(x, y, 1), P'(x', y', 1).$$

Основные преобразования в однородных координатах выражаются следующим образом.

### Перенос

Уравнение переноса запишется в виде матрицы преобразования:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx & Dy & 1 \end{bmatrix}$$

или

$$P' = P \cdot T(Dx, Dy),$$

где

$$T(Dx, Dy) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx & Dy & 1 \end{bmatrix}.$$

Перемножив, получим:

$$\begin{bmatrix} x' y' 1 \end{bmatrix} = \begin{bmatrix} x + Dx, y + Dy, 1 \end{bmatrix}.$$

Докажем, что если точку  $P$  перенести в  $P'$  на расстояние  $(Dx_1, Dy_1)$ , а затем в точку  $P''$  на расстояние  $(Dx_2, Dy_2)$ , то в результате мы получим перенос на расстояние  $(Dx_1 + Dx_2, Dy_1 + Dy_2)$ .

Доказательство:

$$P' = P \cdot T(Dx_1, Dy_1),$$

$$P'' = P' \cdot T(Dx_2, Dy_2).$$

$$P'' = (P \cdot T(Dx_1, Dy_1)) \cdot T(Dx_2, Dy_2) = P(T(Dx_1, Dy_1) \cdot T(Dx_2, Dy_2)).$$

Матричное произведение  $T(Dx_1, Dy_1)$  и  $T(Dx_2, Dy_2)$ :

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx_1 & Dy_1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx_2 & Dy_2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx_1 + Dx_2 & Dy_1 + Dy_2 & 1 \end{bmatrix},$$

то есть перенос — функция аддитивная.

### Масштабирование

Уравнение масштабирования в матричной форме имеет вид:

$$\begin{bmatrix} x' y' 1 \end{bmatrix} = \begin{bmatrix} x y 1 \end{bmatrix} \cdot \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Определяя

$$S(Sx, Sy) = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

имеем:

$$P' = P \cdot S(Sx, Sy).$$

Перемножив, получим:

$$\begin{bmatrix} x' y' 1 \end{bmatrix} = \begin{bmatrix} x \cdot Sx y \cdot Sy 1 \end{bmatrix}.$$

Докажем, что масштабирование — функция мультипликативная, то есть если точку  $P(x, y)$  промасштабировать в точку  $P'(x', y')$  с  $S = (Sx_1, Sy_1)$ , а потом — в точку  $P''(x'', y'')$  с  $S = (Sx_2, Sy_2)$ , то результат будет иметь вид:  $S = (Sx_1 \cdot Sx_2, Sy_1 \cdot Sy_2)$ .

Доказательство:

$$\begin{aligned} P' &= P \cdot S(Sx_1, Sy_1), \\ P'' &= P' \cdot S(Sx_2, Sy_2). \end{aligned}$$

$$P'' = (P \cdot S(Sx_1, Sy_1)) \cdot S(Sx_2, Sy_2) = P(S(Sx_1, Sy_1) \cdot S(Sx_2, Sy_2)).$$

Матричное произведение  $S(Sx_1, Sy_1)$  и  $S(Sx_2, Sy_2)$ :

$$\begin{aligned} &\begin{bmatrix} Sx_1 & 0 & 0 \\ 0 & Sy_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} Sx_2 & 0 & 0 \\ 0 & Sy_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \\ &= \begin{bmatrix} Sx_1 \cdot Sx_2 & 0 & 0 \\ 0 & Sy_1 \cdot Sy_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = S(Sx_1 \cdot Sx_2, Sy_1 \cdot Sy_2). \end{aligned}$$

С помощью масштабирования легко реализуется осевая симметрия. Для этого используются отрицательные значения коэффициентов масштабирования в матрице.

## Поворот

Уравнение поворота можно представить в виде:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Полагая

$$R(\Theta) = \begin{bmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

имеем:

$$P' = P \cdot R(\Theta).$$

Перемножив, получим:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \cos\Theta - y \sin\Theta & x \sin\Theta + y \cos\Theta & 1 \end{bmatrix}.$$

Докажем, что два последовательных поворотов аддитивны. Если точку  $P$  повернуть на угол  $\Theta_1$  в точку  $P'$ , а точку  $P'$  — на угол  $\Theta_2$  в точку  $P''$ , то общий поворот будет равен  $\Theta_1 + \Theta_2$ .

Доказательство:

$$\begin{aligned} P' &= P \cdot R(\Theta_1), \\ P'' &= P' \cdot R(\Theta_2). \end{aligned}$$

$$P'' = (P \cdot R(\Theta_1)) \cdot R(\Theta_2) = P(R(\Theta_1) \cdot R(\Theta_2)).$$

Найдем  $R(\Theta_1) \cdot R(\Theta_2)$ :

$$\begin{aligned} R(\Theta_1) \cdot R(\Theta_2) &= \begin{bmatrix} \cos\Theta_1 & \sin\Theta_1 & 0 \\ -\sin\Theta_1 & \cos\Theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\Theta_2 & \sin\Theta_2 & 0 \\ -\sin\Theta_2 & \cos\Theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \\ &= \begin{bmatrix} \cos(\Theta_1 + \Theta_2) & \sin(\Theta_1 + \Theta_2) & 0 \\ -\sin(\Theta_1 + \Theta_2) & \cos(\Theta_1 + \Theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} = R(\Theta_1 + \Theta_2). \end{aligned}$$

### 2.3. Композиции двумерных преобразований

Но обычно при работе с графической системой объект подвергается сразу нескольким преобразованиям. Для получения желаемого результата используют композицию преобразований, объединяя матрицы  $T, S, R$ . К точке более эффективно применять одно результирующее преобразование, чем ряд преобразований последовательно.

Рассмотрим, например, поворот объекта относительно некоторой точки  $P_1(x_1, y_1)$ .

Ранее был рассмотрен поворот относительно начала координат. Для решения этой задачи разобьем ее на три части (три элементарных преобразования) (рис. 2.5):

- перенос точки  $P_1$  в начало координат —  $T(-x_1, -y_1)$ ;
- поворот —  $R(\Theta)$ ;
- перенос точки  $P_1$  из начала координат в исходную позицию —  $T(x_1, y_1)$ .

Результирующее преобразование:

$$T(-x_1, -y_1) \cdot R(\Theta) \cdot T(x_1, y_1),$$

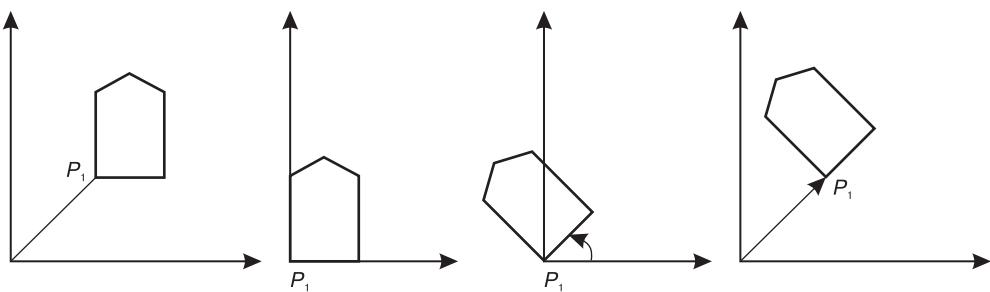
или:

$$\begin{aligned} &\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_1 & -y_1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_1 & y_1 & 1 \end{bmatrix} = \\ &= \begin{bmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ x_1(1-\cos\Theta) + y_1 \sin\Theta & y_1(1-\cos\Theta) + x_1 \sin\Theta & 1 \end{bmatrix}. \end{aligned}$$

Этот пример хорошо иллюстрирует, как применение однородных координат упрощает задачу.

Аналогично, если нужно отмасштабировать объект относительно точки  $P_1(x_1, y_1)$ , а не начала координат, следует:

- перенести точку  $P_1$  в начало координат —  $T(-x_1, -y_1)$ ;
- масштабировать —  $S(Sx, Sy)$ ;
- перенести точку  $P_1$  назад —  $T(x_1, y_1)$ .



**Рис. 2.5.** Поворот объекта относительно точки  $P_1$

Результат имеет вид:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_1 & -y_1 & 1 \end{bmatrix} \cdot \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_1 & y_1 & 1 \end{bmatrix} = \\ = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ x_1(1-Sx) & y_1(1-Sy) & 1 \end{bmatrix}.$$

Если нужно масштабировать, повернуть и расположить в нужном месте домик (центром поворота и масштабирования является точка  $P_1$ ), то необходимо выполнить:

- перенос точки  $P_1$  в начало координат —  $T(-x_1, -y_1)$ ;
- масштабирование —  $S(Sx, Sy)$ ;
- поворот —  $R(\Theta)$ ;
- перенос точки  $P_1$  из начала координат назад —  $T(x_1, y_1)$ .

В структуре данных, в которой содержится это преобразование, могут находиться масштабный коэффициент  $S$ , угол поворота  $\Theta$  и координаты  $(x_1, y_1)$ . И тогда матрица результирующего преобразования будет иметь вид:

$$T(-x_1, -y_1) \cdot S(Sx, Sy) \cdot R(\Theta) \cdot T(x_1, y_1).$$

## 2.4. Матричное представление трехмерных преобразований

Аналогично тому, как двумерные преобразования описываются матрицами размером  $3 \times 3$ , трехмерные могут быть представлены в виде матриц  $4 \times 4$ . И тогда трехмерная точка  $P(x, y, z)$  записывается в однородных координатах как  $P(W \cdot x, W \cdot y, W \cdot z, W)$ , где  $W \neq 0$ . Если же  $W = 1$ , то точка представляется в виде  $P(x, y, z, 1)$ .

### Перенос

Трехмерный перенос является простым расширением двумерного:

$$T(Dx, Dy, Dz) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Dx & Dy & Dz & 1 \end{bmatrix};$$

$$P' = P \cdot T(Dx, Dy, Dz);$$

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot T(Dx, Dy, Dz) = \begin{bmatrix} x + Dx & y + Dy & z + Dz & 1 \end{bmatrix}.$$

### Масштабирование

Расширяется аналогичным образом:

$$S(Sx, Sy, Sz) = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$P' = P \cdot S(Sx, Sy, Sz),$$

или

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot S(Sx, Sy, Sz) = \begin{bmatrix} x \cdot Sx & y \cdot Sy & z \cdot Sz & 1 \end{bmatrix}.$$

### Поворот

Двумерный поворот, рассмотренный ранее, является в то же время трехмерным поворотом вокруг оси  $Z$ :

$$Rz(\Theta) = \begin{bmatrix} \cos\Theta & \sin\Theta & 0 & 0 \\ -\sin\Theta & \cos\Theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Матрица поворота вокруг оси  $X$ :

$$Rx(\Theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\Theta & \sin\Theta & 0 \\ 0 & -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Матрица поворота вокруг оси  $Y$ :

$$Ry(\Theta) = \begin{bmatrix} \cos\Theta & 0 & -\sin\Theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\Theta & 0 & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

При сложном повороте он раскладывается на составляющие (рис. 2.6):

- $\Theta_1$  — поворот вокруг оси  $Z$  до совмещения с плоскостью  $XZ$ ;
- $\Theta_2$  — поворот вокруг оси  $Y$  до совмещения с полуосью  $X$ .

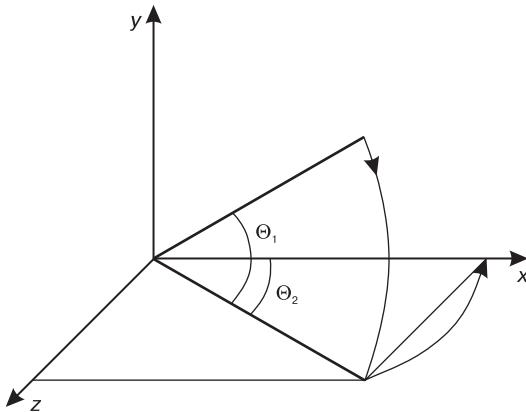


Рис. 2.6. Разложение сложного поворота

## 2.5. Композиции трехмерных преобразований

Как и в случае двумерных преобразований, работая с трехмерными объектами, можно выполнять более сложные действия путем комбинации элементарных операций. Ниже рассмотрен пример преобразования трехмерного отрезка (рис. 2.7).

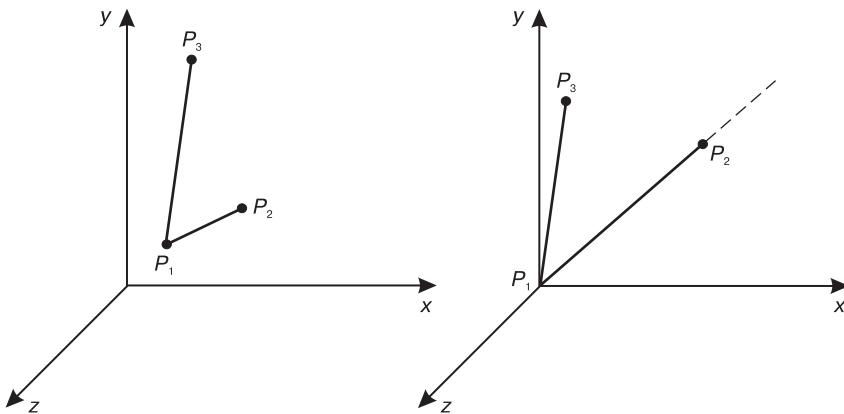


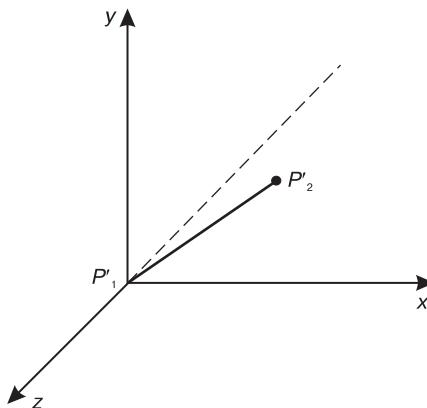
Рис. 2.7. Трехмерное преобразование отрезка<sup>1</sup>

Необходимо преобразовать отрезок  $P_1P_2$  из начальной позиции в конечную таким образом, чтобы точка  $P_1$  совпала с началом координат, а отрезок  $P_1P_2$  располагался вдоль отрицательной полуоси  $Z$ . На длины отрезков преобразование не воздействует.

<sup>1</sup> Фолли Дж., А Вэн Дэм. Основы интерактивной машинной графики. — М.: Мир, 1989.

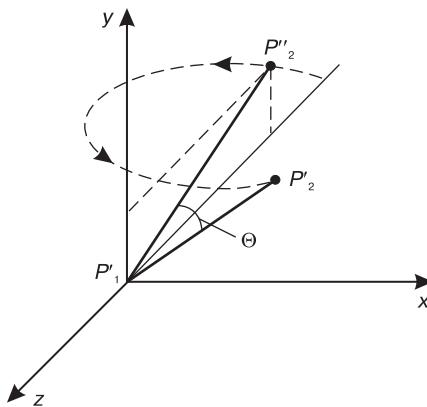
Для решения этой задачи следует выполнить три шага:

- перенос точки  $P_1$  в начало координат (рис. 2.8);



**Рис. 2.8.** Перенос точки  $P_1$  в начало координат

- поворот вокруг оси  $Y$  до совмещения отрезка  $P_1P_2$  с плоскостью  $YZ$  (рис. 2.9);

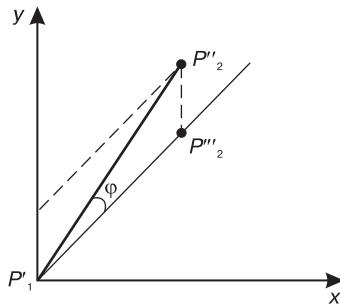


**Рис. 2.9.** Поворот вокруг оси  $Y$

- поворот вокруг оси  $X$  до совмещения отрезка  $P_1P_2$  с отрицательной полуосью  $Z$  (рис. 2.10).

Матрица преобразования при переносе точки  $P_1$  в начало координат имеет вид:

$$T(-x_1, -y_1, -z_1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{bmatrix}.$$



**Рис. 2.10.** Поворот вокруг оси  $X$

Применим преобразование переноса к точкам  $P_1, P_2, P_3$ :

$$\begin{aligned} P'_1 &= P_1 \cdot T(-x_1, -y_1, -z_1) = [0 \ 0 \ 0 \ 1]; \\ P'_2 &= P_2 \cdot T(-x_1, -y_1, -z_1) = [x_2 - x_1 \ y_2 - y_1 \ x_2 - z_1 \ 1]; \\ P'_3 &= P_3 \cdot T(-x_1, -y_1, -z_1) = [x_3 - x_1 \ y_3 - y_1 \ x_3 - z_1 \ 1]. \end{aligned}$$

При повороте вокруг оси  $Y$  на угол  $\Theta$  (угол положительный) (см. рис. 2.9) определяется:

$$\begin{aligned} \cos \Theta &= \frac{-z'_2}{D_1} = \frac{-(z_2 - z_1)}{D_1}; \\ \sin \Theta &= \frac{x'_2}{D_1} = \frac{x_2 - x_1}{D_1}, \end{aligned}$$

где

$$D_1 = \frac{1}{\sqrt{(z_2 - z_1)^2 + (x_2 - x_1)^2}}.$$

Подставляя эти выражения в матрицу поворота, находим:

$$\begin{aligned} P''_2 &= P'_2 \cdot R_y(\Theta) = \\ &= \begin{bmatrix} 0 & y_2 - y_1 - \frac{(x_2 - x_1)^2}{D_1} & -\frac{(z_2 - z_1)^2}{D_1} & 1 \end{bmatrix}. \end{aligned}$$

Поворот вокруг оси  $X$  на угол  $\phi$  (угол отрицательный) (см. рис. 2.10) выражается:

$$\cos(-\phi) = \cos \phi = \frac{-z''_2}{|P_1 P_2|};$$

$$\sin(-\phi) = -\sin \phi = \frac{y''_2}{|P_1 P_2|},$$

где

$$|P_1P_2| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

Общий результат поворота после выполнения трех действий:

$$P''_2 = P'_2 \cdot R_x(\varphi) = P'_2 \cdot R_y(\Theta) \cdot R_x(\varphi) = P_2 \cdot T \cdot R_y(\Theta) \cdot R_x(\varphi) = \begin{bmatrix} 0 & 0 & -|P_1P_2| & 1 \end{bmatrix},$$

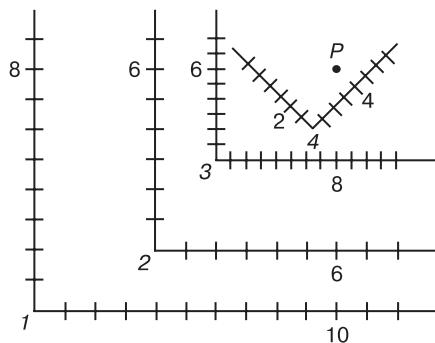
теперь отрезок  $P_1P_2$  совпадает с осью  $Z$ .

## 2.6. Преобразования как изменение систем координат

Рассмотренное преобразование множества точек, принадлежащих объекту, в некоторое другое множество точек производилось в одной и той же системе координат. Таким образом, система координат остается неизменной, а сам объект преобразуется относительно начала координат до получения желаемого результата.

Другим способом описания преобразования является смена систем координат. Такой подход оказывается полезным, когда желательно собрать вместе много объектов, каждый из которых описан в своей собственной локальной системе координат (ЛСК), и выразить их координаты в одной глобальной (ГСК).

Положение точки, заданной в одной системе координат (СК), можно описать в любой другой СК (рис. 2.11).



**Рис. 2.11.** Описание точки в разных системах координат<sup>1</sup>

Точка  $P$  имеет следующие координаты в разных СК:

- в первой СК –  $P^1(10; 8)$ ;
- во второй СК –  $P^2(6; 6)$ ;
- в третьей СК –  $P^3(8; 6)$ ;
- в четвертой СК –  $P^4(4; 2)$ .

Преобразования СК имеют вид:

- из первой СК во вторую СК:  $T_{12} = T(-4, -2)$ ;
- из второй СК в третью СК:  $T_{23} = T(-2, -3) \cdot S(2, 2)$ ;
- из третьей СК в четвертую СК:  $T_{34} = T(-6, -2) \cdot R(-45^\circ)$ .

<sup>1</sup> Фолли Дж., А Вэн Дэм. Основы интерактивной машинной графики. – М.: Мир, 1989.

Тогда координаты точки  $P$  во второй СК на базе координат первой СК:

$$P_2 = P_1 \cdot T_{12} \Rightarrow [6 \ 6 \ 1] = [10 \ 8 \ 1] \cdot T(-4, -2).$$

Обратное преобразование из второй СК в первую СК:

$$T_{21} = T_{12}^{-1} = (-4, -2)^{-1} = (4, 2);$$

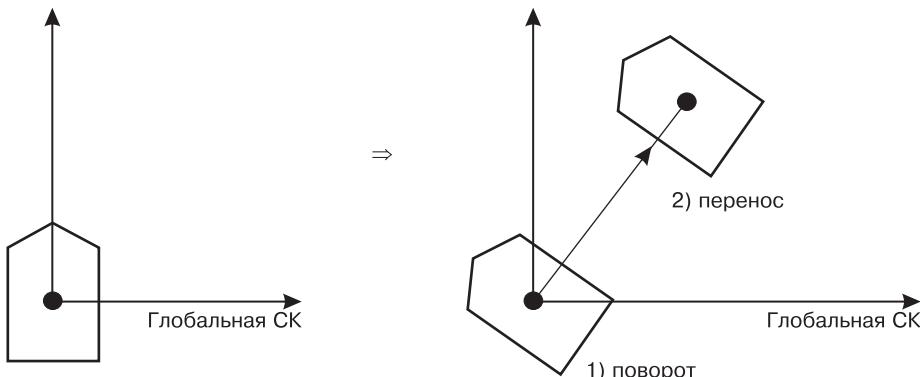
$$T_{32} = T_{23}^{-1} = (T(-2, -3) \cdot S(2, 2))^{-1} = S^{-1}(2, 2) \cdot T^{-1}(-2, -3) = S\left(\frac{1}{2}, \frac{1}{2}\right) \cdot T(2, 3).$$

Преобразование из первой в третью:

$$T_{13} = T_{12} \cdot T_{23} = T(-4, -2) \cdot T(-2, -3) \cdot S(2, 2) = T(-6, -5) \cdot S(2, 2).$$

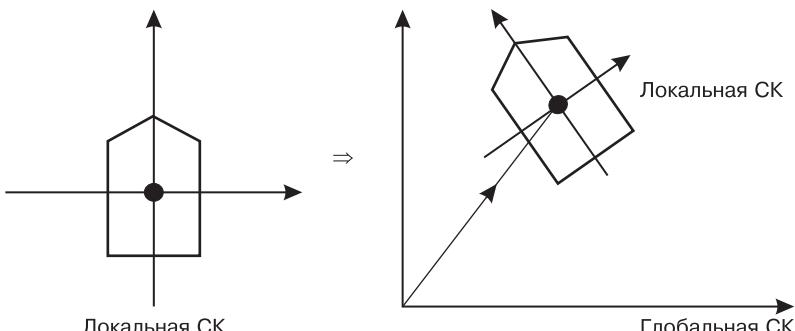
Таким образом, существуют три способа преобразования объектов:

- все объекты описаны в глобальной СК и с помощью преобразований приводятся к новым позициям в той же глобальной СК (рис. 2.12);



**Рис. 2.12.** Преобразование объектов в рамках одной глобальной СК<sup>1</sup>

- каждый объект задан в собственной локальной СК и затем преобразуется в глобальную СК (рис. 2.13);

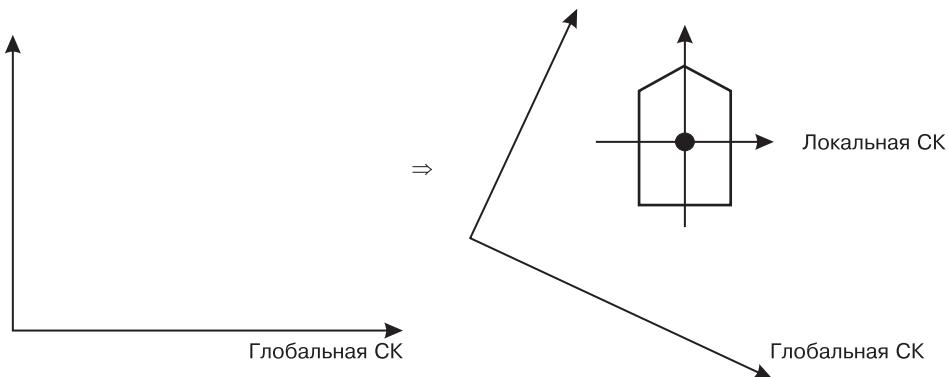


**Рис. 2.13.** Преобразование объектов из локальной СК в глобальную СК<sup>2</sup>

<sup>1</sup> Фолли Дж., А Вэн Дэм. Основы интерактивной машинной графики. — М.: Мир, 1989.

<sup>2</sup> Там же.

- происходит преобразование систем координат с помощью определения новой глобальной СК относительно локальной СК (рис. 2.14).



**Рис. 2.14.** Преобразование объектов путем преобразования глобальной СК относительно локальной СК<sup>1</sup>

Таким образом, можно проводить преобразования как самих объектов, так и системы координат, в которой они описаны.

<sup>1</sup> Фолли Дж., А Вэн Дэм. Основы интерактивной машинной графики. — М.: Мир, 1989.

# Глава 3

## Алгоритмы растровой графики

---

Для получения изображения на растровом дисплее необходимо предварительное преобразование примитивов из векторного вида в растровый. Пользователь задает информацию о примитиве в векторном виде. Для отрезка это будут координаты его конечных точек. Далее необходимо рассчитать внутренние точки. Для этого используются алгоритмы развертки, которые вызываются при каждом создании или каждой модификации изображения. Поэтому алгоритмы должны не только давать хорошее качество изображения, но и делать это как можно быстрее.

Основные критерии алгоритмов растровой развертки такие:

- скорость;
- качество изображения.

На сегодняшний день предпочтение отдается быстродействию.

### 3.1. Преобразование отрезков из векторной формы в растровую

Главной задачей алгоритма развертки отрезков является вычисление координат пикселов на двумерной растровой сетке. При решении этой задачи предполагается, что конечные точки отрезков имеют целые координаты. Простой алгоритм заключается в пошаговом увеличении  $x$ , вычислении  $y = mx + b$ , его округлении (Round( $y$ )) и высвечивании пикселя в точке  $(x, \text{Round}(y))$ . Вычисление произведения  $m \cdot x$  и округление требуют много времени и замедляют процесс разложения в растр.

Преобразование отрезков из векторной формы в растровую можно реализовать с помощью следующих алгоритмов:

- пошагового;
- алгоритма Брезенхема.

Рассмотрим подробнее указанные алгоритмы.

#### Пошаговый алгоритм

Операцию умножения можно устраниТЬ, если заметить, что при увеличении  $x$  на 1 значение  $m = \Delta y / \Delta x$  сводится к  $m = \Delta y$ , то есть изменение  $x$  на 1 приводит к изменению  $y$  на  $m$  (тангенс угла наклона).

Таким образом, если

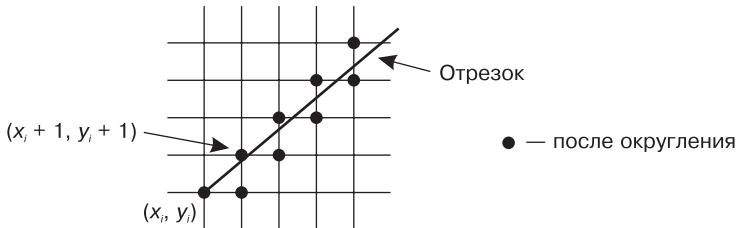
$$x_{i+1} = x_i + 1,$$

то

$$y_{i+1} = y_i + m.$$

После этого полученное значение  $y$  округляется до ближайшего целого.

Последующие значения точки отрезка определяются исходя из предыдущих (рис. 3.1).



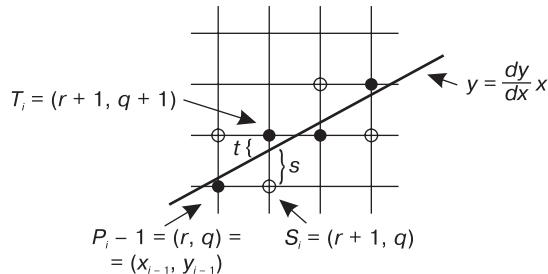
**Рис. 3.1.** Разложение в растр отрезка

Если  $m > 1$ , то шаг по  $x$  будет приводить к увеличению шага по  $y$  более чем на 1 ( $\Delta y > 1$ ). Поэтому, чтобы не было разреженности в изображении, нужно  $x$  и  $y$  поменять местами:  $y$  увеличивать на 1, а  $x$  — на  $1/m$ . После вычисления очередной точки происходит округление ее координат до ближайших целых.

Пошаговый алгоритм работает быстрее, чем предыдущий, так как отсутствует умножение на вещественное число. Но сохранилось одно медленное действие — округление. Полностью избавиться от всех этих недостатков позволяет алгоритм Брезенхема.

### Алгоритм Брезенхема

В данном алгоритме используется только целая арифметика. Вещественные переменные не применяются совсем, и, значит, округление не нужно. Схема алгоритма Брезенхема приведена на рис. 3.2.



**Рис. 3.2.** Схема алгоритма Брезенхема<sup>1</sup>

В алгоритме используется управляющая переменная  $d_i$ , которая на каждом шаге пропорциональна разности между  $s$  и  $t$ , где  $s$  — расстояние от идеальной

<sup>1</sup> Роджерс Д. Алгоритмические основы машинной графики. — М.: Мир, 1989.

линии до нижнего пикселя для текущего  $x$ ,  $t$  — расстояние от идеальной линии до верхнего пикселя. Для точки  $P_i$  нужно определить, какой из пикселов будет выбран:  $T_i$  или  $S_i$ . Если  $s < t$ , то  $S_i$  ближе к отрезку, в противном случае ближе будет  $T_i$ .

То есть если  $s - t < 0$ , то выбирается  $S_i$ , если  $s - t \geq 0$ , выбирается  $T_i$ .

Подробнее алгоритм выглядит следующим образом.

Есть отрезок  $(x_1, y_1) - (x_2, y_2)$ . Пусть первая точка находится ближе к началу координат, тогда перенесем обе точки с помощью  $T(-x_1, -y_1)$  так, чтобы начальная точка отрезка имела координаты  $(0, 0)$ , а конечная —  $(\underbrace{x_2 - x_1}_{=dx}, \underbrace{y_2 - y_1}_{=dy})$ , или  $(dx, dy)$ .

Уравнение прямой будет иметь вид:  $y = \frac{dy}{dx} \cdot x$ .

Обозначим координаты после переноса точки  $P_{i-1}$  через  $(r, q)$ . Тогда координаты:

$$S_i = (r + 1, q);$$

$$T_i = (r + 1, q + 1);$$

$$s = \frac{dy}{dx}(r + 1) - q;$$

$$t = q + 1 - \frac{dy}{dx}(r + 1).$$

Найдем разность:

$$s - t = 2 \frac{dy}{dx}(r + 1) - 2q - 1;$$

$$dx(s - t) = 2(r \cdot dy - q \cdot dx) + 2dy - dx.$$

Величина  $dx > 0$ , поэтому  $dx(s - t)$  можно использовать в качестве проверки условия.

Обозначив  $d_i = dx(s - t)$ , имеем:

$$d_i = 2(r \cdot dy - q \cdot dx) + 2dy - dx.$$

Так как  $r = x_{i-1}$ ,  $q = y_{i-1}$ , то

$$d_i = 2 \cdot x_{i-1} \cdot dy - 2 \cdot y_{i-1} \cdot dx + 2dy - dx.$$

Прибавив единицу к каждому индексу, получим:

$$d_{i+1} = 2 \cdot x_i \cdot dy - 2 \cdot y_i \cdot dx + 2dy - dx.$$

Вычитая  $d_i$  из  $d_{i+1}$ , имеем:

$$d_{i+1} - d_i = 2dy(x_i - x_{i-1}) - 2dx(y_i - y_{i-1}).$$

Но так как  $x_i - x_{i-1} = 1$ , то

$$d_{i+1} = d_i + 2dy - 2dx(y_i - y_{i-1}).$$

То есть сначала вычисляется  $d_i$ .

Если  $d_i \geq 0$ , то выбирается точка  $T_i$ . Тогда  $x_i = x_{i-1} + 1$ ,  $y_i = y_{i-1} + 1$  и

$$d_{i+1} = d_i + 2(dy - dx).$$

Если  $d_i < 0$ , то выбирается  $S_i$ . Тогда  $x_i = x_{i-1} + 1$ ,  $y_i = y_{i-1}$  и

$$d_{i+1} = d_i + 2dy.$$

Таким образом, был получен итерационный способ вычисления  $d_{i+1}$  по предыдущему значению  $d_i$  и выбора между  $S_i$  и  $T_i$ .

Начальное значение  $d_1$  находят при  $i = 1$ ,  $(x_0, y_0) = (0, 0)$ :  $d_1 = 2dy - dx$ .

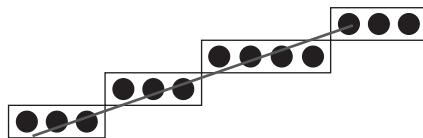
### Ускорение алгоритма Брезенхема

При всех достоинствах алгоритма Брезенхема существует возможность сделать его еще более быстрым. Горизонтальные, вертикальные и диагональные линии не требуют специальных расчетов по алгоритму.

При разложении отрезка можно заметить, что в случае движения вдоль основной оси ( $x$ ) на 1 пиксель получаются группы пикселов, имеющие постоянную координату по неосновной оси ( $y$ ).

В стандартном алгоритме Брезенхема проверка производится на каждом шаге. Используя же группы пикселов, количество проверок можно уменьшить.

Пусть есть отрезок длиной 35 пикселов по  $x$  и 10 пикселов по  $y$  (рис. 3.3).



**Рис. 3.3.** Пример отрезка

Коэффициент наклона  $m = \left[ \frac{1}{3}; \frac{1}{4} \right]$ . Тогда группы пикселов с неизменной  $y$ -координатой будут иметь то 3, то 4 пикселя.

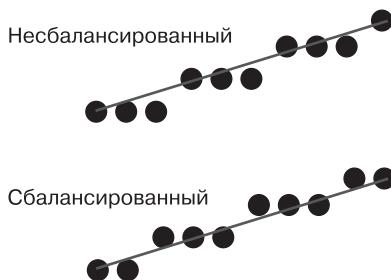
Поэтому достаточно один раз вычислить  $m$ , чтобы определить группы пикселов. Количество вычислений уменьшается примерно на 70 %.

Минимальная длина группы:  $n_{\min} = \text{int}\left(\frac{\Delta X}{\Delta Y}\right) = \text{int}\left(\frac{35}{10}\right) = \text{int}(3,5) = 3$ .

Максимальная длина группы:  $n_{\max} = n_{\min} + 1$ .

Как же располагать группы пикселов с максимальной и минимальной длиной? На каждый шаг по неосновной оси ставится  $n_{\min}$  пикселов вдоль основной. После этого решается, ставить ли еще один пиксель или увеличить  $y$  на 1. Для этого анализируется накопленная ошибка отклонения.

Второй момент — балансирование групп пикселов. Это делается так: сначала оценивается ошибка накопления не за полный шаг на пиксель, а за полпикселя. Если первая и последняя группы состоят из нечетного количества пикселов, получатся несимметричные начальная и конечная группы (рис. 3.4).



**Рис. 3.4.** Пример несбалансированного и сбалансированного отрезков

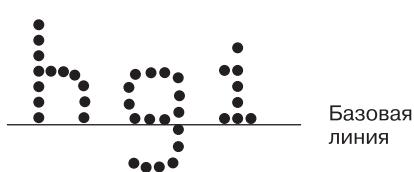
### 3.2. Растровая развертка литер

Метод изображения литер, основанный на матрице точек, используется и при построении растровых изображений. Наименьшей сеткой, с помощью которой можно описывать литеры с приемлемым качеством, является  $5 \times 7$ . Для представления прописных и строчных литер нужна матрица  $7 \times 9$  (клетка  $8 \times 10$ ).

Дисплей разбивают на клетки размером  $8 \times 8$  ( $8 \times 10$ ). Дополнительные пиксели используются для разделения литер и для строчных букв с выносными элементами. Маски литер хранятся в ПЗУ.

При изображении литер необходимо учитывать две особенности.

- Пропорциональное размещение литер, то есть изменение интервалов между центрами литер с учетом их ширины. В случае пропорционального размещения не все литеры будут занимать одно и то же количество пикселов по горизонтали.
- Нижние выносные элементы букв (части литер, опущенные ниже базовой линии (g, p, q, y)) изображаются путем сдвига матриц вниз по отношению к другим литерам (рис. 3.5).



**Рис. 3.5.** Символы с верхним и нижним выносными элементами

Каждая литера представляет собой код матрицы из нулей и единиц.

Пропорциональное размещение литер и их нижние выносные элементы реализуются путем связывания с каждой литературой ширины матрицы и булевой переменной, истинность которой означает, что литера имеет выносной элемент.

### 3.3. Растровая развертка окружностей

Разложение в растр окружности является более сложной задачей. Но учитывая осевую симметрию окружности, можно не только снизить трудоемкость, но и улучшить качество получаемого изображения.

### Четырехсторонняя симметрия

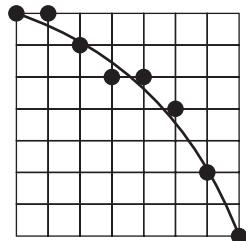
Существует несколько простых, но не очень эффективных способов развертки окружности. Уравнение окружности с центром в начале координат имеет вид:

$$x^2 + y^2 = R^2;$$

$$y = \pm\sqrt{R^2 - x^2}.$$

Чтобы изобразить  $1/4$  окружности, можно увеличивать  $x$  от 0 до  $R$  и на каждом шаге вычислять  $y$  (рис. 3.6). Остальные четверти получаются симметричным отображением.

Этот метод неэффективен, так как в него входят операции умножения и извлечения корня. Более того, при значениях  $x$ , близких к  $R$ , в окружности появляются незаполненные промежутки, то есть в нижней части дуги получается более разреженной. Можно было бы воспользоваться расчетом координат окружности, заданной в полярных координатах —  $R\cos\theta$ ,  $R\sin\theta$ , путем пошагового изменения  $\theta$  от 0 до  $90^\circ$ . Но недостатки по скорости остаются все те же. Качество же изображения повышается за счет равномерности расположения пикселов.



**Рис. 3.6.** Разложение в растр окружности

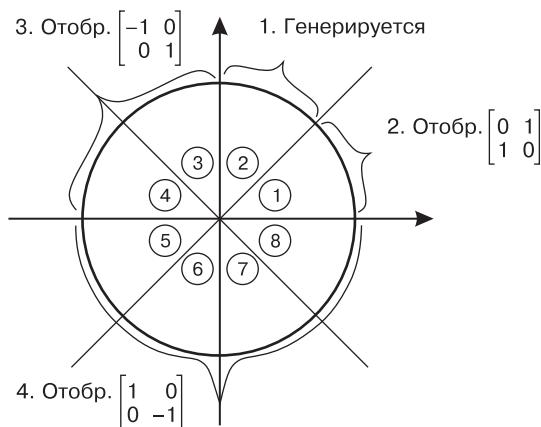
### Восьмисторонняя симметрия

Процесс преобразования окружностей в растровую форму можно улучшить, если полнее использовать симметрию окружности. Поэтому удобно вычислить значения окружности на дуге в  $45^\circ$ , а потом симметрично отобразить их.

### Алгоритм Брезенхема для окружностей

Брезенхем разработал пошаговый генератор дуг, который более эффективен, чем рассмотренные ранее. Необходимо сгенерировать только  $1/8$  часть окружности. Остальные части получаются путем отображения, использованием осевой симметрии (рис. 3.7):

- первый октанта — отражение относительно оси  $y = x$  первого октанта. Матрица отражения:  $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ ;
- второй квадрант — отражение относительно оси  $x = 0$  первого квадранта. Матрица отражения:  $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$ ;
- нижняя полуокружность — отражение относительно оси верхней полуокружности. Матрица отражения:  $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ .

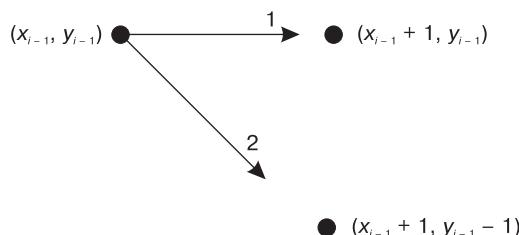


**Рис. 3.7.** Схема получения окружности

Разложение первой четверти окружности с центром в начале координат начинается в точке  $x = 0, y = R$ , окружность генерируется по часовой стрелке,  $y$  — монотонно убывающая функция аргумента  $x$  до точки  $x = \frac{R}{\sqrt{2}}$ .

Для любой заданной точки на окружности существуют только две возможности выбрать следующий пиксель, наилучшим образом приближающий окружность.

Алгоритм выбирает пиксель, для которого минимален квадрат расстояния между одним из этих пикселов и окружностью (рис. 3.8).



**Рис. 3.8.** Схема выбора соседнего пикселя

Расстояния до соседних пикселов определяются:

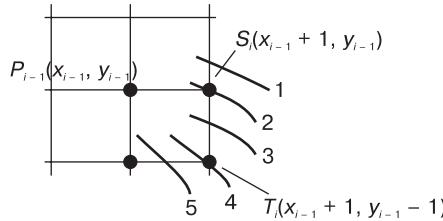
$$D_1 = (x_{i-1} + 1)^2 + y_{i-1}^2 - R^2;$$

$$D_2 = (x_{i-1} + 1)^2 + (y_{i-1} - 1)^2 - R^2.$$

Как и для отрезка, вводится управляющая переменная  $d_i$ , значение которой можно вычислить в пошаговом режиме, используя лишь небольшое число сложений, вычитаний и сдвигов.

Возможны семь способов прохождения истинной окружности через сетку. Пусть пиксель  $P_{i-1}$  был выбран как ближайший к окружности при  $x = x_{i-1}$ . Теперь

найдем, какой пиксель ( $T_i$  или  $S_i$ ) расположен ближе к окружности при  $x = x_{i-1} + 1$  (рис. 3.9).



**Рис. 3.9.** Варианты прохождения окружности

Расстояния до соседних пикселов определяются:

$$D(S_i) = (x_{i-1} + 1)^2 + y_{i-1}^2 - R^2;$$

$$D(T_i) = (x_{i-1} + 1)^2 + (y_{i-1} - 1)^2 - R^2.$$

Если  $|D(S_i)| \geq |D(T_i)|$ , то пиксель  $T_i$  ближе к окружности, чем  $S_i$ .

Если  $|D(S_i)| < |D(T_i)|$ , то пиксель  $S_i$  ближе к окружности.

Введем упрощающую переменную  $d_i$ :

$$d_i = |D(S_i)| - |D(T_i)|.$$

Если  $d_i \geq 0$ , то выбирается пиксель  $T_i$ .

Если  $d_i < 0$ , то выбирается пиксель  $S_i$ .

Для случаев 1, 2:  $D(T_i) < 0$  ( $T_i$  — внутри окружности),  $D(S_i) \leq 0$  ( $S_i$  — внутри окружности).

$$D(S_i) < D(T_i).$$

Так как  $d_i < 0$ , выбирается пиксель  $S_i$ .

Для случая 3:  $D(T_i) < 0$  ( $T_i$  — внутри),  $D(S_i) > 0$  ( $S_i$  — снаружи). В зависимости от знака  $d_i$  выбирается пиксель  $T_i$  или  $S_i$ .

Для случаев 4, 5:  $D(T_i) \geq 0$  ( $T_i$  — снаружи),  $D(S_i) > 0$  ( $S_i$  — снаружи).

$$D(S_i) > D(T_i).$$

Так как  $d_i \geq 0$ , выбирается пиксель  $T_i$ .

После некоторых математических выводов было получено:

$$d_1 = 3 - 2R.$$

Если  $d_i < 0$  (выбирается пиксель  $S_i$ ), то

$$d_{i+1} = d_i + 4x_{i-1} + 6.$$

А если  $d_i \geq 0$  (выбирается пиксель  $T_i$ ), то

$$d_{i+1} = d_i + 4(x_{i-1} - y_{i-1}) + 10.$$

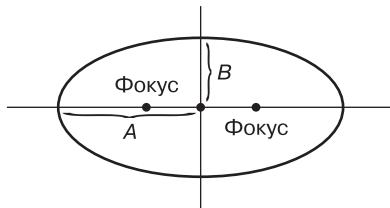
### 3.4. Растровая развертка эллипсов

Еще более сложный случай — разложение в растр эллипса. В отличие от окружности он не обладает восьмисторонней симметрией. Существует несколько способов разложения эллипса в растровую форму.

#### Простой метод

Эллипс — это сплюснутая или расплошнатая окружность.

Эллипс имеет два фокуса, сумма расстояний от любой точки эллипса до каждого фокуса — величина постоянная (рис. 3.10).



**Рис. 3.10.** Эллипс и его характеристики

Окружность — это вырожденный случай эллипса, когда фокусы совпадают.

Эллипс имеет два радиуса —  $A$  и  $B$ .

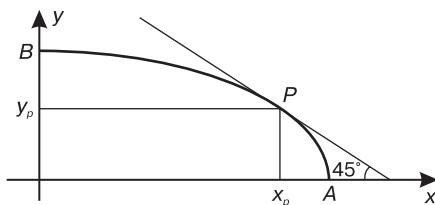
Для окружности  $A = B$ :

$$X^2 + Y^2 = R^2.$$

Для эллипса  $A \neq B$ :

$$\frac{X^2}{A^2} + \frac{Y^2}{B^2} = 1.$$

Используя симметрию, достаточно сгенерировать  $1\backslash 4$  элемента. Сначала основная ось —  $x$ ,  $x$  изменяется от 0 до точки  $P$ , в которой наклон касательной к эллипсу составляет  $45^\circ$  (рис. 3.11).



**Рис. 3.11.** Касательная к эллипсу

После этого основная ось —  $y$ .

Сначала  $x = 0, y = B, x = 1, y = y_1$ , где

$$\frac{x^2}{A^2} + \frac{y^2}{B^2} = 1 \Rightarrow y = \sqrt{B^2 - \frac{B^2 x^2}{A^2}}.$$

При  $x = 1$   $y = \left( \sqrt{B^2 - \frac{B^2}{A^2}} \right) Round$  и т. д. до тех пор, пока  $\frac{y^2}{B^2} \leq \frac{x^2}{A^2}$ . Далее  $y$  изменяется на единицу и вычисляется  $x$ .

### Инкрементивный метод

Уравнение элемента можно переписать:

$$B^2 x^2 + A^2 y^2 - A^2 B^2 = 0.$$

Решив приведенное уравнение для  $x = 0$  и  $y = B - 0,5$ , можно оценить, как далеко ушло  $x$  от того значения, при котором  $y = B - 0,5$  ( $B - 0,5$  — это точка перехода на следующий пиксель по оси  $y$ ). Потом пересчитывается уравнение для  $x + 1$  и т. д.

Когда результат станет положительным, пора декрементировать  $y$ .

Итак, установив  $x = 0, y = B - 0,5$ , получаем:

$$B^2 \cdot 0^2 + A^2 \cdot (B - 0,5)^2 - A^2 \cdot B^2 = 0;$$

$$0,25 \cdot A^2 - A^2 \cdot B = 0.$$

Значит, начальная ошибка накопления:

$$\frac{A^2}{4} - A^2 \cdot B = 0.$$

Если  $A^2/4$  проигнорировать, то останется целочисленная арифметика. Вычисленную вначале ошибку накопления нужно корректировать с каждым шагом по оси  $x$ , пока она не станет положительной, указывая изменить  $y$ .

Это делается так. Для текущей точки ошибка накопления равна  $B^2 \cdot x^2$ . Для следующей точки она составит  $B^2 \cdot (x+1)^2 = B^2 \cdot x^2 + B^2 \cdot x + B^2$ .

$B^2 \cdot x^2$  уже содержится в текущей ошибке, поэтому  $2B^2x + B^2$  добавляется к  $x$  при каждом шаге (все вычисляется в целых числах).

Когда ошибка больше либо равна нулю, делается шаг по оси  $y$  и пересчитывается ошибка для следующего шага. Для этого вместо  $y$  в уравнение эллипса подставляется  $y - 1$ :

$$A^2 \cdot (y - 1)^2 = A^2 \cdot y^2 - 2A^2 \cdot y + A^2.$$

Так как  $A^2 \cdot y^2$  уже есть, то инкрементивная часть равна  $2A^2 \cdot y + A^2$ .

Рисование прекращается по достижении наклона касательной  $45^\circ$ . Это определяется тем, что приращение по неосновной оси больше либо равно приращению по основной оси.

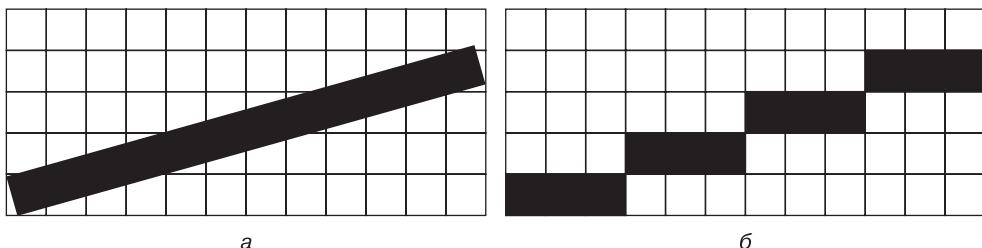
## 3.5. Методы устранения искажений в растровых изображениях

При разложении примитивов в растровую форму хорошо видны возникающие искажения. Это связано с необходимостью преобразовывать непрерывные объекты в дискретные пиксели. В связи с чем очень актуальна задача повышения качества полученных изображений. Для этого были разработаны разные алгоритмы.

## Лестничный эффект

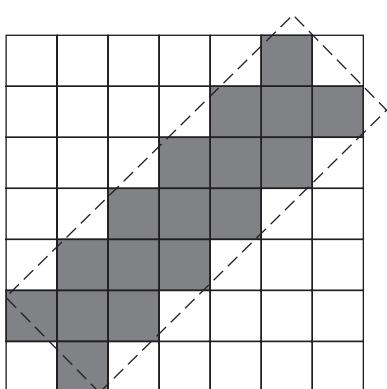
Часто этот эффект можно заметить на границах объектов или на линиях, близких к вертикальным или горизонтальным, но не строго вертикальных или горизонтальных.

Причиной лестничного эффекта является матрица пикселов на компьютерном мониторе. Она имеет конечное и достаточно небольшое разрешение, а пиксели располагаются в строго фиксированных местах. Например, отрезок на рис. 3.12, *a* будет изображен в виде ступенек, образуемых пикселями (рис. 3.12, *б*).



**Рис. 3.12.** Изображение отрезка в виде ступенек

Ступенчатость возникает, когда точки на линиях пересекают строки или столбцы пикселов под небольшим углом. Часть тонкой линии (толщиной 1 пикセル) может попасть на один пиксель экрана, а часть — на другой. Таким образом, получается неопределенность: можно рисовать эту часть как один пиксель на одной строке, один пиксель на другой строке или закрашивать оба пикселя. К сожалению, все три способа вносят хорошо заметные дефекты в изображение. Если закрашивать только один пиксель, линия может получиться тоньше, чем нужно, и будет хорошо заметен разрыв в том месте, где линия переходит с одного ряда пикселов на другой. Если закрашивать оба, толщина линии в этом месте будет 2 пикселя. Аналогичные артефакты возникают не только при рисовании линий, но и практически на всех границах, встречающихся в изображении.



**Рис. 3.13.** Иллюстрация принципа выравнивания отрезка ненулевой толщины

Пусть задан отрезок ненулевой толщины. Какие пиксели необходимо выставить для его отображения? Те, которые более чем на половину покрыты отрезком. Результат такого подхода имеет недостаток — отрезок изображается ступеньками, то есть наблюдается лестничный эффект (рис. 3.13).

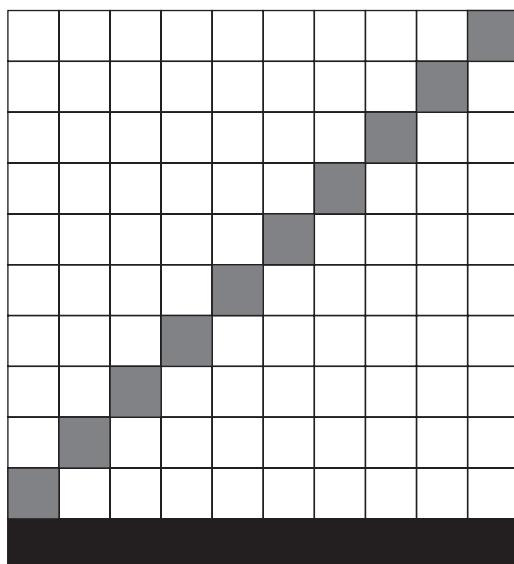
Избавиться от него помогает принцип выравнивания: каждый пиксель выставляется с яркостью, пропорциональной площади пикселя, которую занимает отрезок.

Но использование этого принципа замедляет процесс разложения в растр, так как требуются дополнительные вычисления.

Другой путь — увеличение разрешения экрана.

При увеличении разрешающей способности экрана в два раза по осям  $x$  и  $y$  общее количество пикселов увеличивается в четыре раза, а скорость разложения в растр — в два раза.

Еще одна интересная особенность, которая заметна для отрезков разного наклона, — плотность пикселов на единицу длины отрезка. Диагональный и горизонтальный отрезки (рис. 3.14) имеют одинаковое количество пикселов (10) при отображении, но длины у них разные. Это приведет к разной яркости отрезков на экране. Горизонтальный отрезок имеет максимальную яркость, а диагональный отрезок — минимальную.



**Рис. 3.14.** Пример диагонального и горизонтального отрезков

Следовательно, яркость отрезков зависит от угла их наклона. Чтобы сделать яркость отрезков одинаковой, нужно менять интенсивность пикселов.

Так, если яркость пикселов в отрезке II равна 1, то яркость пикселов в отрезке I должна быть равна  $\frac{1}{\sqrt{2}}$ .

### Мелкие и движущиеся объекты

Интересная ситуация возникает в случае работы с мелкими объектами (соизмеримыми с пикселом).

Используя принцип высвечивания тех пикселов, центр которых покрыт объектом, объекты I и III будут проигнорированы, так как не покрывают центр пикселя, а объект II будет высечен в виде целого пикселя с максимальной яркостью (рис. 3.15).

Это не соответствует реальной картине расположения объектов.

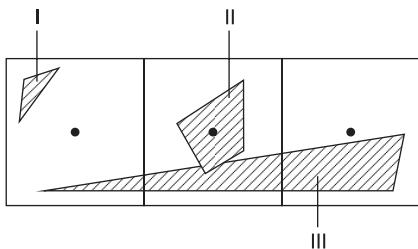


Рис. 3.15. Пиксели и мелкие объекты

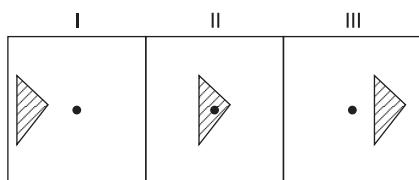


Рис. 3.16. Пиксели и движущиеся объекты

Недостатки подобного плана возникают также при движении мелких объектов (рис. 3.16). В первом кадре объект не высовчивается, так как не покрывает центр пикселя, во втором кадре пиксель горит, в третьем опять не высовчивается, то есть будет наблюдаться мерцание. Кадры I, III – объекта нет, кадр II – объект есть.

Для устранения подобных недостатков физический пиксель разбивается на несколько мнимых пикселов (тем самым увеличивается разрешение). Значение интенсивности физического пикселя определяется как усредненное значение интенсивностей мнимых пикселов (рис. 3.17).

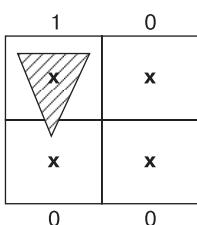


Рис. 3.17. Разбиение физического пикселя на четыре мнимых

Для данного примера объектом покрывается центр одного мнимого пикселя. Интенсивность физического пикселя определяется как среднее значение интенсивностей мнимых:

$$i_{\bar{p}} = \frac{1+0+0+0}{4} = \frac{1}{4}.$$

Для цветных изображений вместо 256 независимых цветов оцениваются RGB-составляющие для каждого из четырех мнимых пикселов.

Графический процессор визуализирует экранное изображение с разрешением, значительно большим, чем текущее разрешение дисплея. Существует достаточно много методов выполнения данной операции, при этом их все можно охарактеризовать количеством дополнительно используемых пикселов. После рисования изображения с высоким разрешением процессор уменьшает размер картинки до разрешения дисплея.

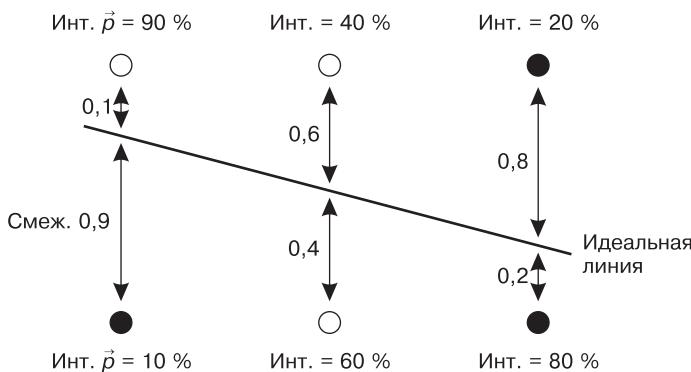
### 3.6. Сглаживание линий. Алгоритм Ву

Есть итальянская поговорка: «Необязательно говорить правду, главное, чтобы то, что вы скажете, хорошо звучало». Этот принцип часто используется в компьютерной графике. И на то есть веские основания. Сегодня приходится работать с большим количеством графических данных, показывая сложные трехмерные сцены. Естественно, для этого требуются скоростные алгоритмы. Но на сегодняшний день технические средства не позволяют выполнять все эти действия в реальном времени с нужной скоростью. Вот и приходится часто использовать упрощения, о которых пользователь даже не догадывается в процессе работы. Кроме того, фи-

зиологические особенности восприятия информации человеком вполне позволяют воспользоваться этим.

Это относится и к сглаживанию линий. Не нужно рисовать примитивы идеально. Необходимо, чтобы они лишь выглядели таковыми. Не следует углубляться в сложную математику, чтобы получить идеально сглаженные линии. Зрительная система человека видит то, что хочет видеть, так почему бы не дать ей лишь ключ к распознаванию?

Идея алгоритма Ву заключается в том, что при рисовании линий обычным образом с каждым шагом по основной оси высвечиваются два пикселя по неосновной оси (рис. 3.18).



**Рис. 3.18.** Выбор интенсивностей пикселов

Их интенсивность подбирается пропорционально расстоянию от центра пикселя до идеальной линии. Чем дальше пикセル, тем меньше его интенсивность. Значения интенсивности двух пикселов дают в сумме 1, то есть это интенсивность одного пикселя, в точности попавшего на идеальную линию. Это придает линии одинаковую интенсивность на всем ее протяжении, создавая иллюзию, что пиксели расположены вдоль линии не по два, а по одному в точности по идеальной линии.

Горизонтальные, вертикальные и диагональные линии не требуют сглаживания. Для других линий алгоритм Ву проводит их вдоль основной оси, подбирая координаты по неосновной оси. Смещение вдоль неосновной оси вычисляется одним целочисленным делением. Это значение называется *ошибкой смещения*. Ошибка накопления ( $d$ ) показывает, как далеко ушли пиксели от идеальной линии по неосновной оси, и как только она достигает критического значения, делается шаг на один пиксель вдоль неосновной оси. Если основной осью является  $x$ , то будут установлены два пикселя с координатами  $(x, y)$  и  $(x, y + 1)$ . Короче говоря, продвижение вдоль линии аналогично алгоритму Брезенхема, только на каждом этапе устанавливается не один пиксель, а два. Осталось определить их интенсивность. Ошибка смещения суммируется с ошибкой накопления.

Пусть количество уровней интенсивности кратно двум, минимальная интенсивность будет равна  $2^n - 1$ , максимальная — 0.

Старшие  $n$  бит ошибки накопления покажут необходимую интенсивность для одного из пикселов. Интенсивность второго пикселя пары будет  $(2^n - 1)$ . Ошибка накопления содержит соотношение расстояний от центра каждого из двух пикселов до идеальной линии (рис. 3.19).

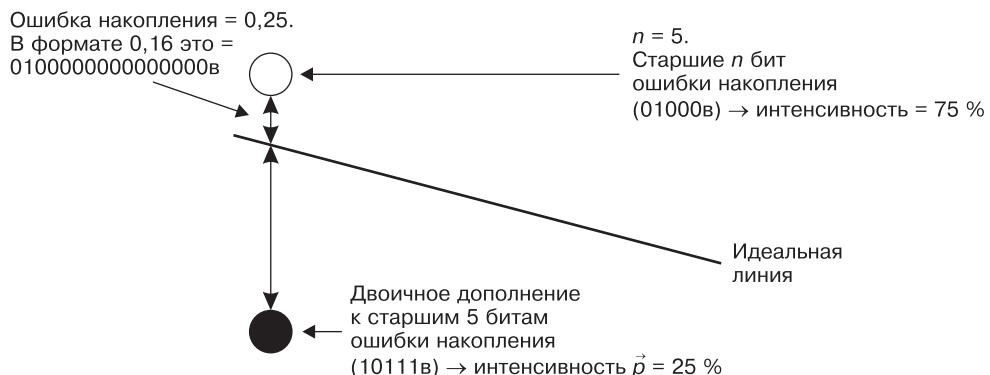


Рис. 3.19. Определение интенсивностей пикселов

### 3.7. Заполнение области

Часто возникает необходимость заполнить уже разложенную в растр область. Существует много алгоритмов, позволяющих решить подобную задачу. Они делятся на два типа в зависимости от того, как задается область. Можно выбрать замкнутый контур или указать точку внутри замкнутого контура.

#### Алгоритм построчного сканирования

Пусть имеется область, граница которой разложена в растр. Требуется заполнить область внутри. Пусть внутри задана точка и  $a$  — значение граничных пикселов,  $b$  — значение пикселов внутри области до заполнения,  $c$  — новое значение пикселов внутри области. Пример области представлен на рис. 3.20.

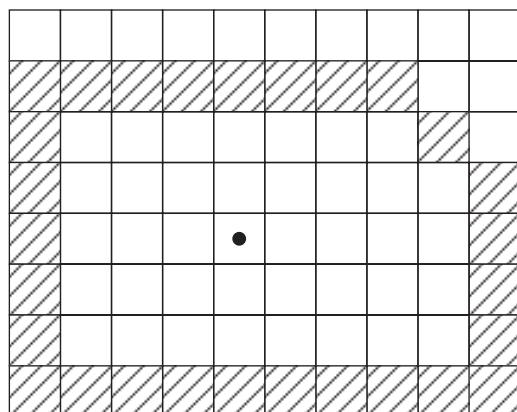
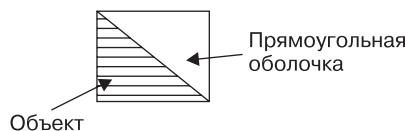


Рис. 3.20. Пример области заполнения

Объект заключается в прямоугольную оболочку, и проверяется принадлежность пикселя объекту.

Проводится построчное сканирование. Находится пикセル со значением  $a$ , затем пикセル, следующий за ним и имеющий значение  $b$ , которое меняется на  $c$ , и так до тех пор, пока не будет встречен еще один пикセル со значением  $a$ . После этого осуществляется переход на следующую строку.

Недостатком этого метода является необходимость просматривать больше пикселов, чем необходимо (рис. 3.21).

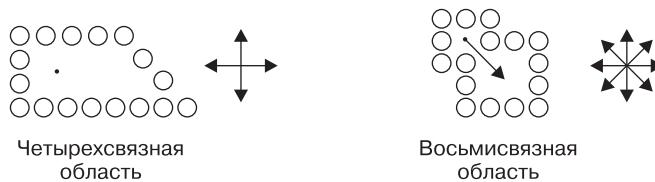


**Рис. 3.21.** Объект и его прямоугольная оболочка

### Метод заполнения с затравкой

Если предыдущий алгоритм требует выбора замкнутого контура, то в методе заполнения с затравкой указывается точка внутри контура. В зависимости от связности области рассматривается разное количество направлений. Обычно их четыре или восемь.

Область называется четырехсвязной, если из любой внутренней точки можно достичь любой другой точки, двигаясь в одном из четырех направлений (рис. 3.22).



**Рис. 3.22.** Примеры четырех- и восьмисвязной областей

Алгоритм включает в себя следующие шаги.

1. Поместить затравочный пиксель в стек.
2. Пока стек не пуст:
  - 1) извлечь пиксель из стека;
  - 2) присвоить пиксели требуемое значение;
  - 3) для каждого из соседних четырехсвязных пикселов проверить:
    - является ли он граничным;
    - не присвоено ли ему требуемое значение;
  - 4) проигнорировать пиксель в любом из этих двух случаев, иначе поместить пиксель в стек.

Ниже рассмотрен пример заполнения области алгоритмом с затравкой (рис. 3.23). Приоритет направлений — вправо, вверх, влево, вниз. Сначала в стек заносится затравочный пиксель (номер 0), потом извлекается и закрашивается. На его место записывается пиксель 1 (правое направление), так как он еще не закрашен и не является граничным. Аналогично записываются ниже в стек пиксели 2, 3 и 4. Извлекается последний (пиксель 4), закрашивается, на его место записывается пиксель 5 (правое направление). Пиксель 0 игнорируется, так как уже закрашен. Потом в стек заносится пиксель 6 (левое направление). Пиксель по нижнему направлению

игнорируется, так как является граничным. И так далее, пока не будет заполнена вся область. Если в процессе заполнения встречается тупик, то есть текущий пиксель не имеет незакрашенных соседних, происходит возвращение по стеку вверх. Как только находится некоторый незакрашенный пиксель, к нему применяются эти же проверки, и алгоритм продолжает работу. И так до тех пор, пока все пиксели стека не будут закрашены. В рассматриваемом примере встречается один тупик. После закраски пикселя 3 траектория заполнения прерывается, происходит подъем вверх по стеку на одну позицию, где хранится пиксель 2. Проверяется, что он не закрашен, и далее продолжается закраска по алгоритму.

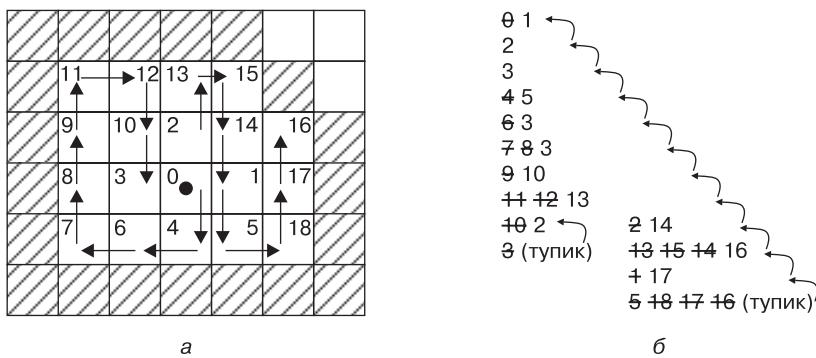


Рис. 3.23. Траектория заполнения области (а) и стек (б)

### Заполнение линиями

Область для заполнения задается точкой внутри. Для заданной точки  $(x, y)$  определяется и заполняется максимальный отрезок, содержащий эту точку и лежащий внутри области. Потом в поисках еще не заполненных пикселов проверяются отрезки, лежащие выше и ниже. Если такие пиксели находятся, то функция рекурсивно вызывается дальше. Алгоритм эффективен и для областей с отверстиями.

## 3.8. Разложение в растр сплошных многоугольников

В отличие от заполнения области, где граница многоугольника (области) уже была разложена в растр, в данном случае требуется получить растровый образ сплошной области, граница которой задана в векторном виде. Можно, конечно, воспользоваться уже известными алгоритмами разложения в растр сначала области, а потом ее заполнения. Но существует более быстрое решение, позволяющее совместить эти две операции.

### Когерентность сканирующих строк

В самом простом случае, чтобы определить пиксели, которые нужно закрасить, необходима проверка каждого пикселя экрана на принадлежность многоугольни-

ку. Но такой подход очень трудоемкий, так как приходится проделывать много лишних операций.

Упростить данную задачу можно, если использовать тот принцип, что соседние пиксели часто ведут себя одинаково, то есть мы можем работать сразу с группой пикселов, включая их в изображение многоугольника или отбрасывая. На этом и основан принцип пространственной *когерентности* — перемещаясь от пикселя к пикселю или от одной сканирующей строки к другой, многоугольник чаще всего остается постоянным. То есть, определяя понятие когерентности, можно связать его с постоянством. Когерентность широко используется во всей растровой графике, в том числе и в обработке трехмерных сцен, позволяя сократить вычисления.

Используя этот принцип, можно предположить следующий алгоритм.

1. Определить точки пересечения текущей сканирующей строки со сторонами многоугольника.
2. Отсортировать точки пересечения по увеличению  $x$ -координаты.
3. Попарно их закрасить.

Для сканирующей строки 2 области (рис. 3.24) алгоритм работает следующим образом.

1. Точки пересечения: 3, 6.
2. Сортировка: 3, 6.
3. Закраска: 3 – 6.

Для строки 6 алгоритм будет таким.

1. Точки пересечения: 8, 6, 3, 1.
2. Сортировка: 1, 3, 6, 8.
3. Закраска: 1 – 3, 6 – 8.

Для строки 5 алгоритм следующий.

1. Точки пересечения: 8, 4, 4, 1.
2. Сортировка: 1, 4, 4, 8.
3. Закраска: 1 – 4, 4 – 8.

Для строки 3 алгоритм такой.

1. Точки пересечения: 3, 8, 8.
2. Сортировка: 3, 8, 8.
3. Закраска: 3 – 8, 8 – правая граница буфера.

Для строки 3 закраска произойдет неверно. Можно заметить, что строка 3 пересекает вершину 3 и количество точек пересечения становится нечетным. В то же время строка 5 также пересекает вершину, но количество точек пересечения является четным, то есть четыре. Было замечено, что на получаемый результат влияет тип вершины. Вершины разбиваются на два типа:

- промежуточные (1, 3);
- локальные и глобальные экстремумы (2, 5, 6, 4).

Если вершина является промежуточной, то одно из ребер, ее составляющих, уменьшается на 1 по  $y$ .

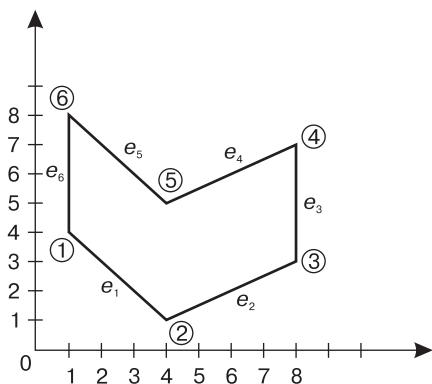


Рис. 3.24. Область для закраски

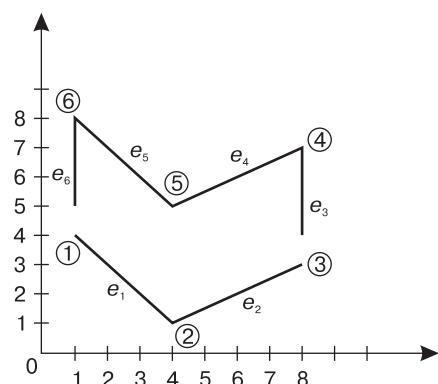


Рис. 3.25. Область для закраски с укороченными ребрами

Тогда для строки 3 (ребро  $e_3$  укоротили на 1) (рис. 3.25) алгоритм следующий.

1. Точки пересечения: 3, 8.
2. Сортировка: 3, 8.
3. Закраска: 3 — 8.

Также необходимо укоротить одно из ребер, составляющих вершину 1. Например, ребро  $e_6$ .

### Когерентность ребер

Используемая когерентность сканирующих строк позволила повысить скорость работы. Но можно заметить и еще один вид когерентности, который тоже можно было бы учесть в работе, — когерентность ребер.

Принцип когерентности ребер: если ребро пересекается строкой  $i$ , то велика вероятность, что оно будет пересекаться и строкой  $(i + 1)$ .

$$y_{i+1} = y_i + 1, x_{i+1} = x_i + \Delta x;$$

$$m = \frac{\Delta y}{\Delta x} = \frac{1}{\Delta x} \Rightarrow \Delta x = \frac{1}{m}, x_{i+1} = x_i + \frac{1}{m}.$$

Учитывая два типа когерентности, алгоритм разложения в растр сплошной области состоит в следующем. Создается таблица ребер (TP), в которой все ребра сортируются по увеличению  $y$ -координаты. В ней содержатся все ребра многоугольника, и каждое ребро представлено в виде:

$$y_{\max}, x(y_{\min}), \frac{1}{m}, \text{ связь.}$$

На основе TP создается таблица активных ребер (ТАР), которая содержит только текущие ребра для каждой сканирующей строки. Они сортируются по увеличению  $x$ -координаты (рис. 3.26).

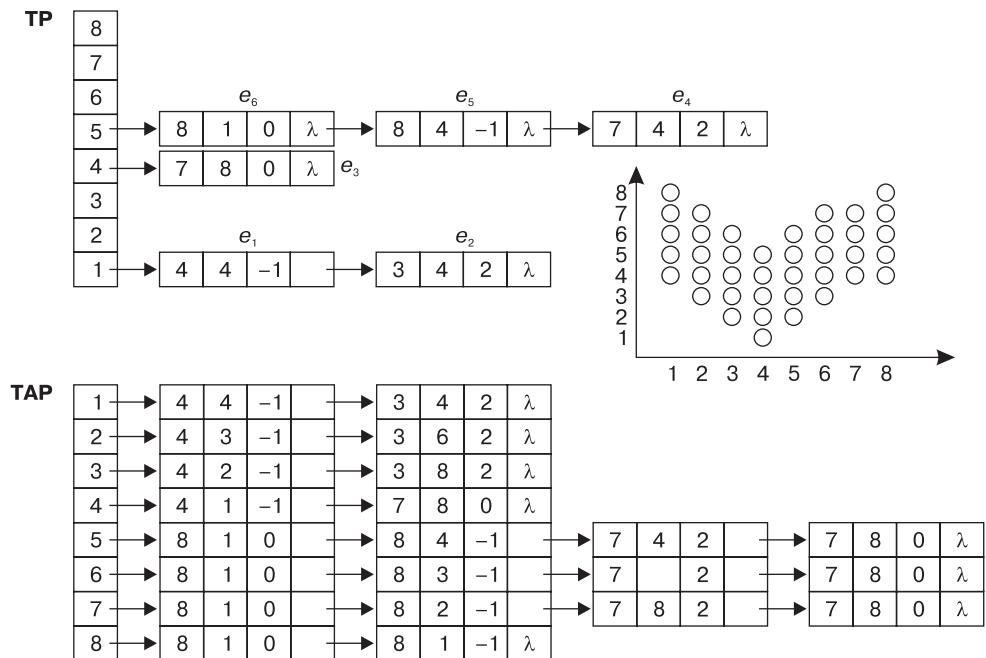


Рис. 3.26. Таблица ребер и таблица активных ребер

Каждое ребро в ТАР представлено в виде:

$$y_{\max}, x_i, \frac{1}{m}, \text{ связь.}$$

Алгоритм создания ТАР следующий.

1. Установить  $y$  равным минимальному значению координаты  $y$  среди элементов ТР.
2. Инициализировать ТАР, сделать ее пустой.
3. Повторять шаг 2 до тех пор, пока ТАР и ТР не станут пустыми.
4. Слить информацию из группы  $y$  таблицы ребер (ТР) с информацией в ТАР, сохраняя упорядочивание по  $x$ -координате.
5. Занести желаемые значения в пиксель на сканирующей строке, определяемой текущим значением  $y$ , используя пары  $x$ -координат из ТАР.
6. Удалить из ТАР те элементы, которые  $y > y_{\max}$ .
7. Для всех элементов, содержащихся в ТАР, заменить  $x$  на  $x + \frac{1}{m}$ .
8. Так как на предыдущем шаге могла нарушиться упорядоченность ТАР по  $x$ , провести пересортировку ТАР.
9. Увеличить  $y$  на 1 и таким образом перейти к следующей сканирующей строке.

# Глава 4

## Отсечение линий

В процессе работы с графическими системами пользователь формирует графические объекты произвольных размеров и сложности в системе координат наблюдателя. Используемые графические устройства имеют фиксированные границы (обычно прямоугольные). Иногда нужно задать некоторую прямоугольную область на экране, которая определяет размеры желаемого изображения. Она называется *областью вывода*. Следовательно, попадание частей объекта за пределы области вывода может привести к определенным затруднениям. Иногда не попадающие полностью в область вывода линии просто не вычерчиваются, но это не всегда приемлемо. Поэтому приходится отсекать части отрезков прямых линий, выходящих за пределы области.

### 4.1. Алгоритм Коэна—Сазерленда

Пусть область вывода задана прямоугольником  $ABCD$  (рис. 4.1), а отсекаемый треугольник —  $PRQ$ .

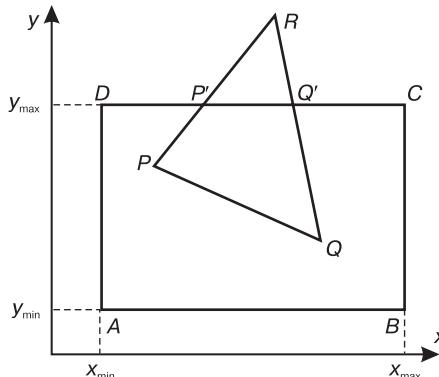


Рис. 4.1. Область вывода и отсекаемый объект<sup>1</sup>

Все видимые отрезки прямых линий должны лежать внутри окна. Процесс отсечения должен выполняться автоматически. Команды на вычерчивание треугольника  $PRQ$  интерпретируются как команды на вычерчивание отрезков  $P'P$ ,  $PQ$ ,  $QQ'$ .

Координаты точек  $P'$  и  $Q'$  неизвестны. Их нужно вычислить, зная координаты точек  $P$ ,  $R$  и  $Q$ . Наклон отрезка  $PR$  вычисляется следующим образом:

<sup>1</sup> Аммерал Л. Принципы программирования в машинной графике. — М.: СолСистем, 1992.

$$m = \frac{\Delta y}{\Delta x} = \frac{y_R - y_p}{x_R - x_p} = \frac{y_{p'} - y_p}{x_{p'} - x_p};$$

$$y_{p'} = y_{\max};$$

$$x_{p'} = x_p + \frac{(x_R - x_p)(y_{\max} - y_p)}{x_R - y_p}.$$

В рассмотренном случае точка  $P$  находилась внутри окна, а точка  $R$  удовлетворяла неравенствам:

$$x_{\min} < x_R < x_{\max} \text{ и } y_R > y_{\max}.$$

Однако необходимо рассмотреть значительно больше ситуаций взаимного расположения точек отрезка и области вывода. Большое разнообразие логических операций, которые нужно выполнять для решения этой задачи, делает вопрос отсечения линий очень интересным с алгоритмической точки зрения. Коэн и Сазерленд разработали алгоритм для отсечения отрезков прямых линий. Его суть в том, что конечным точкам отрезка ставится в соответствие некоторый четырехбитный код:

$$b_0 b_1 b_2 b_3,$$

где  $b_i$  может быть либо 0, либо 1. Этот код содержит информацию о положении точки  $P$  относительно окна. Возможны девять комбинаций.

1001	0001	0101	$b_0 = 0$ , если $x \geq x_{\min}$
1000	0000	0100	$b_0 = 1$ , если $x < x_{\min}$
1010	0010	0110	$b_1 = 0$ , если $x \leq x_{\max}$ $b_1 = 1$ , если $x > x_{\max}$ $b_2 = 0$ , если $y \geq y_{\min}$ $b_2 = 1$ , если $y < y_{\min}$ $b_3 = 0$ , если $y \leq y_{\max}$ $b_3 = 1$ , если $y > y_{\max}$

Ниже рассмотрен пример, иллюстрирующий работу алгоритма (рис. 4.2).

Пусть для отрезка  $P_1P_2$  получены коды:  $\text{cod}(P_1)$ ,  $\text{cod}(P_2)$ .

- Если коды содержат только 0, то  $P_1P_2$  целиком лежит внутри окна и должен быть начертан полностью.
- Если коды содержат единичный бит в одной и той же позиции, то отрезок целиком лежит за границами окна и не вычерчивается.

Подобные проверки легко реализуются с помощью побитных логических операций «и» и «или» над кодами конечных точек отрезка.

Остальные случаи необходимо рассмотреть подробнее. Если хотя бы один из кодов содержит единичный бит, то либо  $P_1$ , либо  $P_2$  перемещается из области вне окна к одной из границ окна или к ее продолжению (точка  $P_1$  перемещается в точку  $R$ , а точка  $P_2$  — в точку  $U$ ). В последнем случае точка по-прежнему будет находиться вне окна, и понадобится еще одно перемещение (точка  $R$  переместится

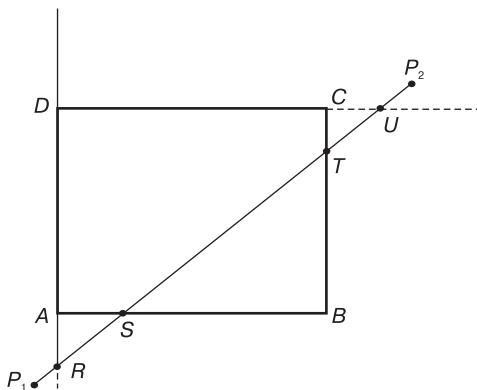


Рис. 4.2. Отсекаемый отрезок. Пример 1

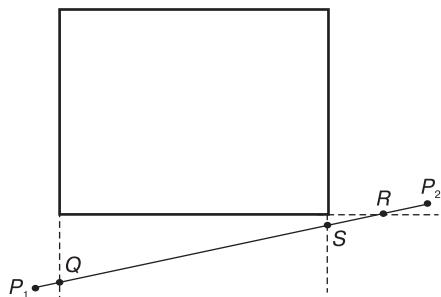


Рис. 4.3. Отсекаемый отрезок. Пример 2

в точку  $S$ , а точка  $U$  — в точку  $T$ ). Таким образом, процесс отсечения может быть многоступенчатым, на каждом шаге расстояние между точками  $P_1$  и  $P_2$  уменьшается. Процесс завершается, как только обе точки окажутся в пределах окна (код 0000). Оставшаяся часть отрезка будет вычерчена (отрезок  $ST$ ).

Расположение отрезка, показанного на рис. 4.3, тоже не соответствует двум рассмотренным случаям (тривиальная видимость и тривиальная невидимость), не требующим дальнейшего отсечения. Поэтому проводится перенос точек. Точка  $P_1$  перемещается в точку  $Q$ .

Коды конечных точек имеют вид:

$$\text{cod}(P_1) = 1010, \text{cod}(P_2) = 0100.$$

После переноса точки  $P_1$  в точку  $Q$  коды будут иметь вид:

$$\text{cod}(Q) = 0010, \text{cod}(P_2) = 0100.$$

Требуется дальнейшее отсечение, точка  $P_2$  переносится в точку  $R$ :

$$\text{cod}(Q) = 0010, \text{cod}(R) = 0100.$$

Далее точка  $R$  переносится в точку  $S$ :

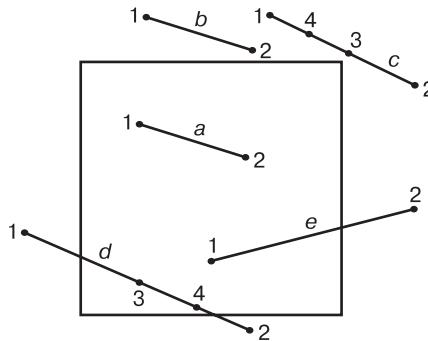
$$\text{cod}(Q) = 0010, \text{cod}(S) = 0010.$$

Теперь условие соответствует второму случаю (тривиальная невидимость), и обе точки находятся ниже окна. Следовательно, отрезок  $P_1P_2$  (или его усеченный вид  $QS$ ) чертить не нужно.

## 4.2. Алгоритм разбиения средней точкой

В предыдущем алгоритме нужно было вычислить пересечение отрезка со стороны окна, что является достаточно длительной операцией. Этого можно избежать, если реализовать двоичный поиск такого пересечения путем деления отрезка его средней точкой. Алгоритм был предложен Спруллом и Сазерлендом. Программ-

ная реализация его медленнее, но аппаратная быстрее и эффективнее, так как аппаратно сложение и деление на 2 достаточно просты (деление на 2 эквивалентно побитному сдвигу вправо).



**Рис. 4.4.** Варианты расположения отрезков

В алгоритме используются коды концов отрезка и проверки, выявляющие полную видимость (отрезок *a*) и тривиальную невидимость (отрезок *b*), как и в предыдущем методе. Остальные отрезки, не определяемые тривиально, разбиваются на две равные части. Затем те же проверки применяются к каждой из половин до тех пор, пока не будет обнаружено пересечение со стороной окна или длина разделяемого отрезка не станет пренебрежительно малой, то есть пока он не выродится в точку. После вырождения определяется видимость полученной точки. Максимальное количество разбиений пропорционально точности задания координат концов отрезка.

Рассмотрим отрезок *c*. Хотя этот отрезок невидим, он пересекает диагональную прямую окна и не может быть тривиально отвергнут. Разбиение его средней точкой 3 позволяет тривиально исключить половину 3–2, а половина 1–3 тоже пересекает диагональ окна. Делим 1–3 пополам точкой 4 и отвергаем невидимый отрезок 1–4. Разбиение отрезка 3–4 продолжается до тех пор, пока не будет найдено пересечение этого отрезка с правой стороной окна. Затем исследуется обнаруженная точка, и она оказывается невидимой, следовательно, весь отрезок невидим.

Рассмотрим отрезок *d*. Он также не определяется тривиально. Разбиение его средней точкой 3 приводит к одинаковым результатам для обеих половин. Разобьем отрезок 3–2 пополам точкой 4. Отрезок 3–4 полностью видим, а отрезок 4–2 видим частично. Отрезок 3–4 можно было бы начертить, но это привело бы к неэффективному изображению видимой части отрезка (серия коротких кусков). Поэтому точка 4 запоминается как текущая видимая точка, которая наиболее удалена от точки 1. А разбиение отрезка 4–2 продолжается. Как только обнаружится видимая средняя точка, она объявляется текущей, наиболее удаленной от точки 1 до тех пор, пока не будет обнаружено пересечение с нижней стороной окна с заранее заданной точностью. Это пересечение и будет объявлено самой удаленной от точки 1 видимой точкой. Затем точно так же обрабатывается

отрезок 1–3. Наиболее удаленной от точки 2 видимой точкой будет его пересечение с левой стороной окна.

Таким образом, для отрезков, подобных  $d$ , реализуются два двоичных поиска двух видимых точек, наиболее удаленных от концов отрезка. Это точки пересечения отрезка со сторонами окна. Для отрезков типа  $e$  один из двух поисков не нужен.

### 4.3. Трехмерное отсечение отрезков

Существуют две наиболее распространенные формы трехмерных отсекателей:

- прямоугольный параллелепипед (рис. 4.5, *a*);
- усеченная пирамида (рис. 4.5, *б*).

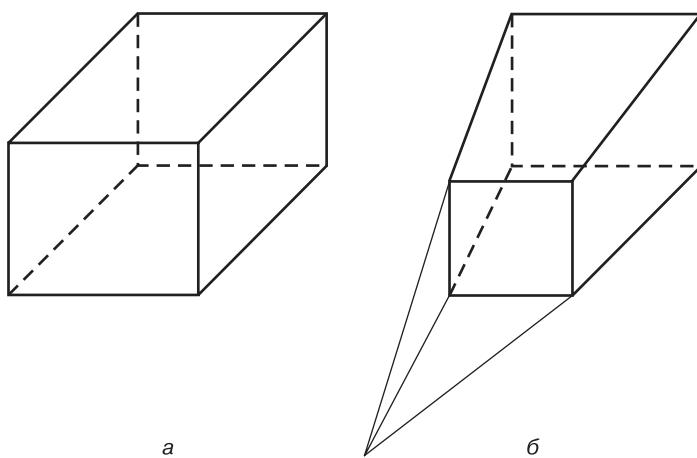


Рис. 4.5. Трехмерное отсечение по объему

У каждой из них шесть граней. Для определения видимости отрезков обобщим алгоритм, используемый для двумерного случая. В трехмерном случае используется шестибитный код:

$$a_0 \ a_1 \ a_2 \ a_3 \ a_4 \ a_5.$$

- $a_0$  — если конец отрезка левее объема;
- $a_1$  — если конец отрезка правее объема;
- $a_2$  — если конец отрезка ниже объема;
- $a_3$  — если конец отрезка выше объема;
- $a_4$  — если конец отрезка ближе объема;
- $a_5$  — если конец отрезка дальше объема.

В противном случае в соответствующие биты заносятся нули.

Далее производится анализ кодов.

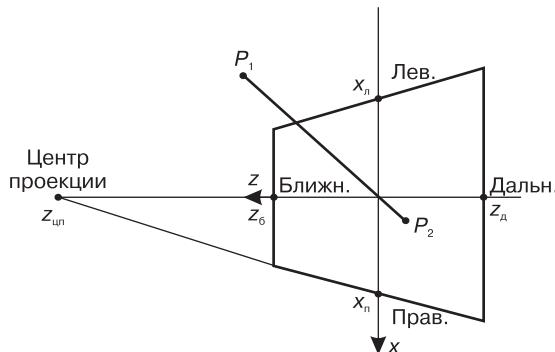
- Если коды обоих концов отрезка равны 0, то оба конца видимы и отрезок тоже будет полностью видим.

- Если коды содержат единичный бит в одной и той же позиции, то отрезок полностью невидим.

В остальных случаях отрезок может быть частично видим или полностью невидим. Необходимо определить пересечения отрезка с гранями отсекающего объема.

Поиск точки пересечения с гранями параллелепипеда является обобщением соответствующего двумерного алгоритма.

Сложнее ситуация, когда отсекатель — усеченная пирамида (рис. 4.6).



**Рис. 4.6.** Отсечение по пирамиде<sup>1</sup>

Рассмотрев усеченную пирамиду сверху, можно легко найти уравнение прямой на плоскости  $XZ$ , несущей проекцию правой грани отсекателя:

$$x = \frac{(z - z_{\text{ни}})x_n}{(z_d - z_{\text{ни}})} = z\alpha_1 + \alpha_2, \text{ где } \alpha_1 = \frac{x_n}{z_d - z_{\text{ни}}}, \alpha_2 = -\alpha_1 z_{\text{ни}}.$$

Зная это уравнение, можно определить местоположение точки: справа (вне), на или слева (внутри) от прямой. Подстановка координат  $x$  и  $z$  точки  $P$  в функцию правой грани дает результат:

$$f_n = x - z\alpha_1 - \alpha_2.$$

Если  $f_n > 0$ , то точка  $P$  находится справа от плоскости; если  $f_n = 0$ , то точка  $P$  находится на плоскости; если  $f_n < 0$ , то точка  $P$  находится слева от плоскости.

Аналогично находятся функции для остальных пяти граней.

Для левой грани:

$$f_l = x - z\beta_1 - \beta_2, \text{ где } \beta_1 = \frac{x_n}{z_d - z_{\text{ни}}}, \beta_2 = -\beta_1 z_{\text{ни}}.$$

Если  $f_l > 0$ , то точка  $P$  находится справа от плоскости; если  $f_l = 0$ , то точка  $P$  находится на плоскости; если  $f_l < 0$ , то точка  $P$  находится слева от плоскости.

Для верхней грани:

$$f_b = y - z\gamma_1 - \gamma_2, \text{ где } \gamma_1 = \frac{y_n}{z_d - z_{\text{ни}}}, \gamma_2 = -\gamma_1 z_{\text{ни}}.$$

<sup>1</sup> Аммерал Л. Принципы программирования в машинной графике. — М.: СолСистем, 1992.

Если  $f_u > 0$ , то точка  $P$  находится выше плоскости; если  $f_u = 0$ , то точка  $P$  находится на плоскости; если  $f_u < 0$ , то точка  $P$  находится ниже плоскости.

Для нижней грани:

$$f_u = y - z\delta_1 - \delta_2, \quad \text{где } \delta_1 = \frac{y_u}{z_u - z_{\text{ни}}}, \quad \delta_2 = -\delta_1 z_{\text{ни}}.$$

Если  $f_u > 0$ , то точка  $P$  находится выше плоскости; если  $f_u = 0$ , то точка  $P$  находится на плоскости; если  $f_u < 0$ , то точка  $P$  находится ниже плоскости.

Для ближней грани:

$$f_b = z - z_b.$$

Если  $f_b > 0$ , то точка  $P$  находится ближе к плоскости; если  $f_b = 0$ , то точка  $P$  находится на плоскости; если  $f_b < 0$ , то точка  $P$  находится дальше от плоскости.

Для дальней грани:

$$f_d = z - z_d.$$

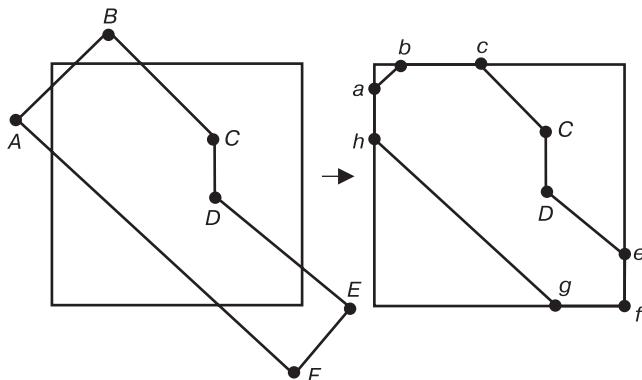
Если  $f_d > 0$ , то точка  $P$  находится ближе к плоскости; если  $f_d = 0$ , то точка  $P$  находится на плоскости; если  $f_d < 0$ , то точка  $P$  находится дальше от плоскости.

При  $z_{\text{ни}} \rightarrow \infty$  формы отсекателя стремятся к прямоугольному параллелепипеду, функции стремятся к функциям прямого параллелепипеда.

Неточности могут возникнуть, если концы отрезка лежат за центром проекции,  $z > z_{\text{ни}}$ . Для этого необходимо обратить значения первых четырех бит кода.

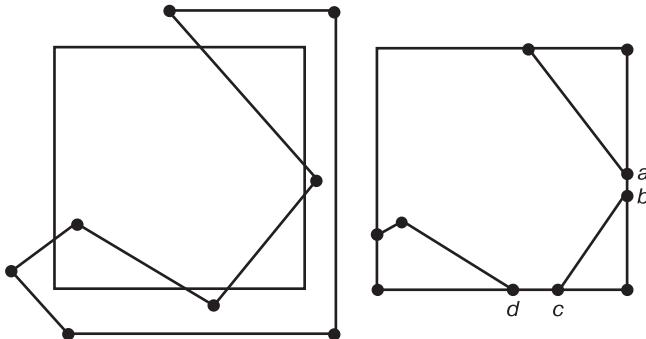
#### 4.4. Отсечение многоугольников

Ранее было рассмотрено отсечение отрезков. Многоугольник можно рассматривать как набор отрезков в том случае, если это контурное изображение. Тогда исходная фигура может превратиться в один или более открытых многоугольников или просто стать совокупностью отдельных отрезков (рис. 4.7).



**Рис. 4.7.** Отсечение многоугольника

Но если многоугольники рассматриваются как сплошные области, необходимо, чтобы замкнутость сохранялась и у результата, то есть отрезки  $bc$ ,  $ef$ ,  $fg$ ,  $ha$  должны быть добавлены к описанию результирующего многоугольника. Добавление  $ef$  и  $fg$  вызывает особые трудности. Много сложностей возникает и тогда, когда результат отсечения представляет собой несколько несвязанных областей (рис. 4.8).



**Рис. 4.8.** Отсечение с несвязанными областями

Иногда отрезки  $ab$  и  $cd$  включаются в описание результата. И если исходный многоугольник красный на синем фоне, то отрезки  $ab$  и  $cd$  тоже будут красными на синем фоне, что противоречит ожидаемому результату.

**Алгоритм Сазерленда—Ходжмена для отсечения многоугольника.** В этом алгоритме исходный и каждый промежуточный многоугольник отсекаются последовательно относительно одной прямой. Исходный многоугольник задается списком вершин:

$$P = \{P_1, P_2, \dots, P_n\},$$

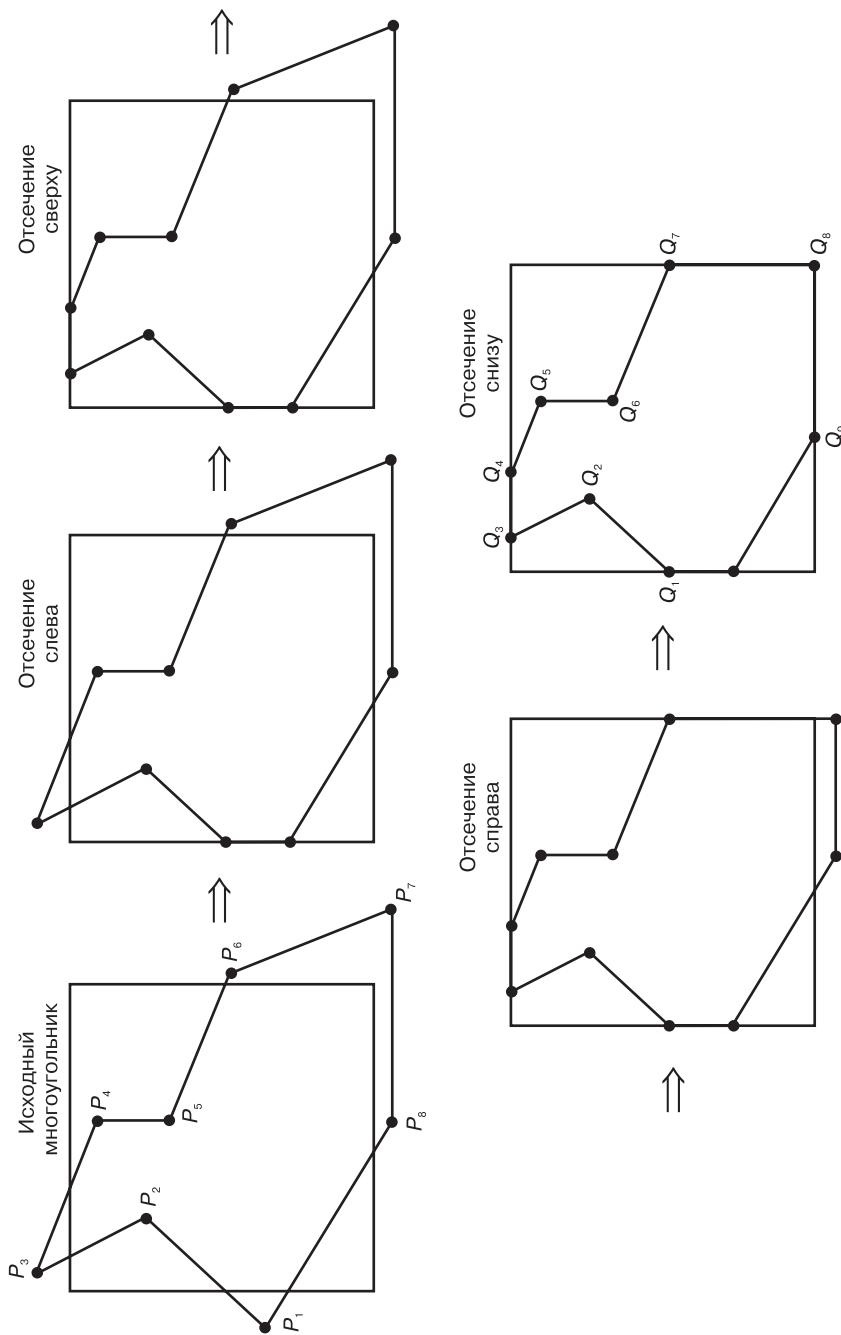
который порождает список его ребер:

$$R = \{P_1P_2, P_2P_3, \dots, P_{n-1}P_n, P_nP_1\}.$$

Шаги алгоритма показаны на рис. 4.9.

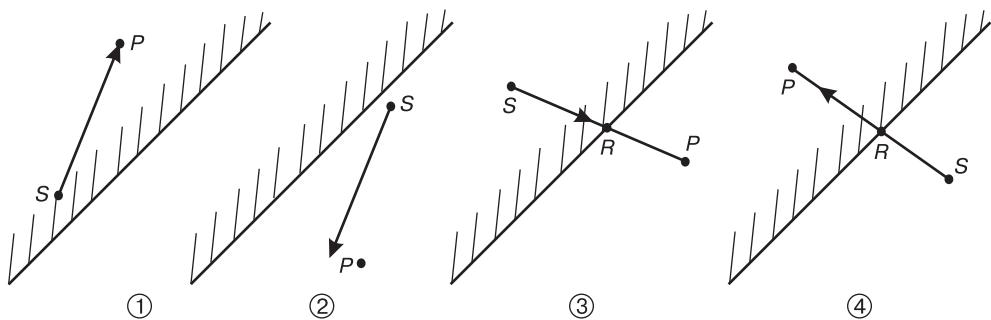
Добавление точки  $Q_8$  теперь стало тривиальным. Этот алгоритм может отсекать любой многоугольник (выпуклый и невыпуклый, плоский и неплоский) относительно любого окна, являющегося выпуклым многоугольником. Порядок отсечения многоугольника разными сторонами непринципиален. Результат работы алгоритма — список новых вершин многоугольника. Так как каждая сторона многоугольника отсекается независимо от других, достаточно рассмотреть только возможные ситуации расположения одного отрезка относительно одной отсекающей плоскости.

Рассмотрим каждую точку  $P$  из списка, за исключением первой, как конечную точку ребра, начальной точкой  $S$  которого является предыдущая  $P_{i-1}$  точка в этом списке. Возможны четыре варианта расположения ребра и отсекающей плоскости (рис. 4.10).



**Рис. 4.9.** Шаги отсечения многоугольника по алгоритму Сазерленда—Ходжмена<sup>1</sup>

<sup>1</sup> Аммерал Л. Принципы программирования в машинной графике. — М.: СолСистем, 1992.



**Рис. 4.10.** Варианты расположения ребра и отсекающей плоскости

Итогом будет занесение в список вершин результирующего усеченного многоугольника нуля, одной или двух вершин.

- **Вариант 1. Полная видимость.** Результат — вершина  $P$  (одна точка) (занести в результат начальную точку  $S$  не нужно, так как если вершины рассматриваются поочередно, то точка  $S$  уже была конечной точкой предыдущего ребра и уже попала в результат).
- **Вариант 2. Полная невидимость.** Результат — ноль точек.
- **Вариант 3. Выход из области видимости.** Результат — точка  $R$  (одна точка).
- **Вариант 4. Вход в область видимости.** Результат — точки  $R, P$  (две точки) (так как конечная вершина  $P$  видима, она тоже должна попасть в результат).

Для первой вершины многоугольника нужно определить только факт ее видимости. Если вершина видима, то она попадает в результат и становится начальной точкой  $S$ . Если же вершина невидима, она тоже становится начальной точкой, но в результат не попадает.

## 4.5. Отсечение литер

Литеры, или текст, можно генерировать программно либо аппаратно. Они могут состоять из отдельных отрезков (штрихов) или быть образованными точечной матрицей.

Штриховые буквы, сгенерированные программно, можно обрабатывать как любые отрезки: поворачивать, переносить, масштабировать, отсекать по любым окнам, используя рассмотренные ранее алгоритмы (рис. 4.11).

Программно сгенерированные символы в форме точечной матрицы можно обрабатывать так же. Если прямоугольная оболочка буквы пересекается с окном, то нужно проверить, будет ли каждый пиксель маски символа находиться внутри окна. В этом случае пиксель активизируется, иначе — нет.

На отсечение аппаратно сгенерированных букв накладываются больше ограничений. Обычно любая не полностью видимая буква удаляется (для этого прямоугольная оболочка буквы сравнивается с границами окна) (рис. 4.12).

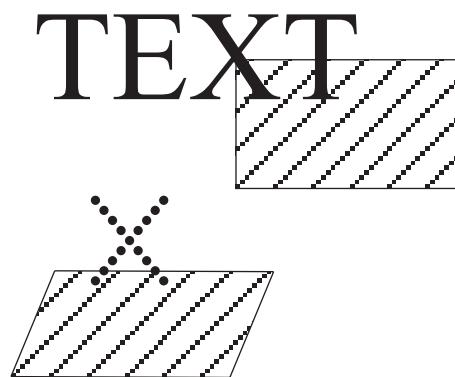


Рис. 4.11. Отсечение текста

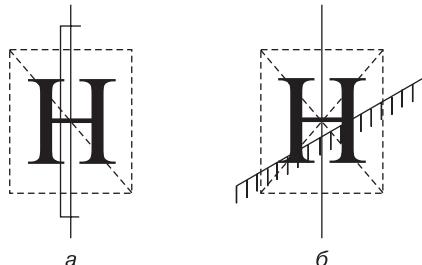


Рис. 4.12. Отсечение символа

Если прямоугольная оболочка литеры ориентирована так же, как граница окна (см. рис. 4.12, *a*), то тест видимости можно провести только для одной из диагоналей оболочки. Если прямоугольная оболочка литеры ориентирована иначе, чем окно, то тесты видимости нужно провести для двух диагоналей (см. рис. 4.12, *б*).

# **Глава 5**

## **Проектирование графического диалога**

---

При проектировании графической интерактивной системы особое внимание необходимо уделить организации диалога. Подобно архитектуре, проектирование интерфейса пользователя является в какой-то мере искусством, а не наукой.

В данной главе рассмотрен некоторый набор правил, указывающих, что следует делать, а чего не следует. При творческом применении этот подход и эти правила могут помочь обратить внимание на эргономику интерактивной системы.

Правильная организация диалога влияет на скорость работы и частоту совершаемых ошибок. Порой эти характеристики при разной организации диалога могут отличаться в два и более раз. Системы должны строиться так, чтобы не оттолкнуть пользователя сразу же после первой попытки. Общение между человеком и компьютером обойдется без недоразумений только в том случае, если человек будет знать, что подразумевается в каждом конкретном случае, а компьютер сможет умело ориентироваться в мире понятий пользователя. Любимое дело программистов — заставлять пользователя изучать контекст программы. Самое трудное в обучении пользованию нетривиальными программами — понять внутреннюю семантическую модель, которая в том или ином виде присутствует в любой программе. Те графические системы, семантические модели которых строились на общекультурном контексте, имели большой успех.

Простота использования, но не простота реализации — важнейший принцип проектирования интерактивных систем.

Ранее особое внимание уделялось оптимизации использования двух компьютерных ресурсов — времени и памяти. Эффективность программы была наивысшей целью. Теперь же, когда снижается стоимость аппаратуры и возрастает графическая оснащенность персональных компьютеров, можно обратить особое внимание на эффективность работы пользователя.

### **5.1. Языковая аналогия**

Существует полезная аналогия между диалогом пользователя с компьютером и человеческим общением. «Словами» компьютерной графики являются картинки и действия (нажатие кнопок, выбор элементов с помощью мыши, задание координат). Тем не менее, многие атрибуты общения человека с человеком являются основой приятного графического диалога.

Язык диалога должен быть языком пользователя без излишней ориентации на компьютер, то есть необходимо обходиться простыми правилами, простым словарем и использовать понятия, которые пользователю уже знакомы или которые ему легко изучить.

Требования к языку диалога следующие.

- **Эффективность.** На эффективном языке команды можно давать оперативно и кратко.
- **Полнота.** Позволяет выразить любую идею, относящуюся к области исследования.
- **Естественная грамматика.** Минимальное количество простых и легких для изучения правил. Это позволяет пользователю сконцентрировать внимание на решаемой проблеме. Основной принцип — нужно избегать сложных грамматических правил, которые могут прервать мысли пользователя.

В разговоре человека с человеком один задает вопрос или что-то утверждает, а другой отвечает и, как правило, быстро. Если же ответ задерживается, то видно, что человек еще обдумывает его. Аналогично должно быть и при общении с компьютером. И то и другое является формой обратной связи. Иногда говорящий может сделать ошибку, но тут же исправить ее. Возможность исправлять ошибки важна также и в диалоге с компьютером.

## 5.2. Языковая модель

При разработке интерфейса пользователя с компьютером существуют два языка: *входной*, когда пользователь задает команды для компьютера (язык выражается в действиях с различными диалоговыми устройствами), и *выходной*, когда компьютер отвечает пользователю (язык выражается графически с помощью примитивов и их атрибутов).

Языковая модель включает четыре части:

- концепция;
- семантика;
- синтаксис;
- лексика.

Подробно эти части рассмотрены на примере модели проектирования обстановки в комнате (рис. 5.1).

При *концептуальном проектировании* определяются ключевые прикладные понятия:

- объекты или классы объектов;
- взаимосвязи между ними;
- операции над ними.

Например, в текстовом редакторе объектами являются строки и файлы. Связь между объектами состоит в том, что файлы — это последовательность строк. Над объектами-строками можно выполнять операции включения, удаления, перемещения, копирования; над объектами-файлами — операции создания, удаления, включения, переименования, копирования.

Концептуальная модель проектирования обстановки в комнате имеет один объект — комнату и один класс объектов — предметы мебели. Связь между ними в том, что комната содержит мебель. Действия — добавить, убрать, переместить, повернуть.

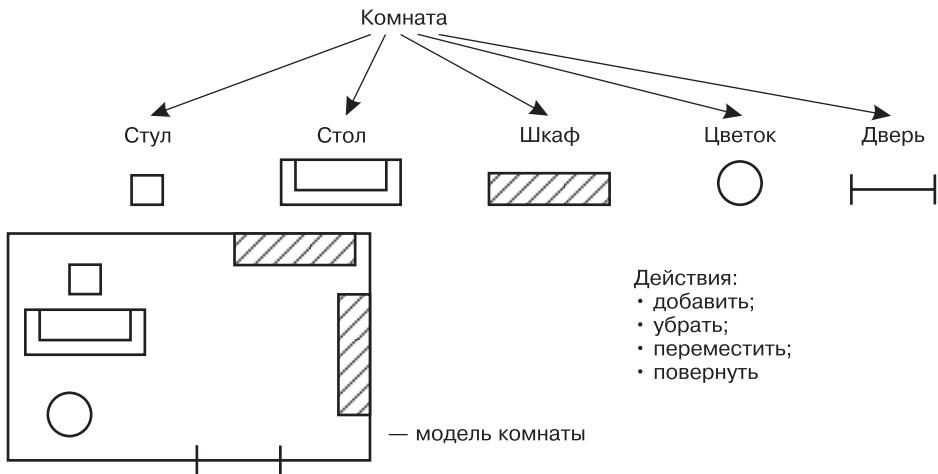


Рис. 5.1. Модель проектирования обстановки в комнате

При *семантическом проектировании* подробно определяются функциональные характеристики языка:

- необходимая информация для каждой операции над объектом;
- возможные семантические ошибки и способы их устранения;
- результат каждой операции.

При размещении мебели в комнате семантикой входного языка являются операции: добавить, убрать, переместить и повернуть.

При *синтаксическом проектировании* определяется последовательность, в которой должны появляться элементарные единицы языка. Для входного языка эта последовательность является грамматикой — набором правил, руководствуясь которыми можно формировать последовательности слов, то есть предложения. Сюда относятся команды, имена, значения, координаты и произвольный текст.

Начинается предложение с команды, за ней следуют несколько вопросов в определенном порядке. Например, операция добавления объекта представляет собой такую последовательность действий.

1. Ввести имя команды.
2. Выбрать добавляемый объект.
3. Определить параметры смещения.

При *лексическом проектировании* определяются способы формирования отдельных знаков (команд, имен, значений, координат) из имеющихся аппаратных средств и устройств ввода. Таким образом, лексическое проектирование — это привязка аппаратных возможностей к аппаратно-независимым знакам языка.

Так, для программы размещения мебели лексические компоненты входного языка определяются привязкой, например, к планшету, с помощью которого определяется позиция, осуществляются выбор команды из меню и выбор объекта.

### 5.3. Обеспечение обратной связи

Представьте себе разговор с человеком, который, слушая вас, никак не реагирует — не улыбается, не кивает и отвечает только тогда, когда его к этому вынуждают. С таким человеком вряд ли захочется разговаривать, более того, вы не имеете никакого подтверждения, что собеседник действительно вас слушает. То же самое относится и к общению с компьютером. Обратная связь является существенной составляющей диалога с компьютером, так же как и диалога человека с человеком. Различие состоит в том, что в человеческом разговоре есть много источников обратной связи (жесты, мимика), графический же терминал не представляет почти никакой автоматической обратной связи (исключение составляет индикатор «включено»). Так что обратную связь необходимо разработать заранее.

Обратная связь осуществляется на трех уровнях:

- лексическом;
- синтаксическом;
- семантическом.

*Лексическая обратная связь* является самым низшим уровнем обратной связи. Каждое лексическое действие на входном языке может сопровождаться лексическим откликом на выходном языке (например, буквы, набираемые пользователем на клавиатуре, могут немедленно отражаться на экране, а изменение позиции мыши может сопровождаться перемещением экранного указателя).

Обратная связь на *синтаксическом уровне* возникает при вводе в систему каждого знака входного языка (команды, позиции, выбранного объекта). Команда, выбранная из меню, или перемещаемый объект могут выделяться изменением яркости, в результате чего пользователь знает, что действия восприняты.

Также формами обратной связи на синтаксическом уровне являются приглашение на ввод следующего знака, подсвечивание кнопки программируемой функциональной клавиатуры, нажатой пользователем, повторение с помощью синтезатора слов, введенных голосом.

Другая форма обратной связи на синтаксическом уровне — реакция системы не на каждый синтаксический знак, а на полное предложение, которое система нашла правильным, что она и подтверждает.

Наиболее полезной и приятной формой обратной связи на *семантическом уровне* является извещение пользователя о завершении введенной команды. Это выполняется путем предъявления пользователю нового или модифицированного изображения. Но не все команды имеют графическое исполнение (например, сохранение рисунка на внешней памяти). Поэтому в данном случае используются подсказки или сообщения о завершении операции.

Другим типом семантической обратной связи, нужным только в тех случаях, когда выполнение команды требует большего количества времени, является извещение пользователя о том, что компьютер работает над его командой. Известны случаи, когда отсутствие такой обратной связи приводило к тому, что пользователи физическиправлялись с дисплеем!

Подобная обратная связь может принимать много форм, особенно удобен круг с вращающейся стрелкой, которая совершает полный оборот за время выполнения команды, или линейная шкала, постепенно заполняемая цветом.

Иногда удобно постепенно выводить на экран частичные результаты. Получаемая мультиплексия помогает лучше понять конечный результат.

Важно также и то, какое место на экране отводится обратной связи. Можно выделить некоторую фиксированную область. Однако такой подход нарушает визуальную непрерывность, так как взгляд пользователя должен постоянно переходить от рабочей области к области системных сообщений и наоборот. Звуковая обратная связь устраняет этот недостаток, но она не везде может использоваться. Удобно отводить для обратной связи то место на экране, куда смотрит пользователь, то есть где находится указатель мыши.

## 5.4. Помощь пользователю

Многие графические системы создаются для широкого круга пользователей, которые имеют разный опыт работы с программой. Так, новичкам нужно больше подсказок, чем опытным пользователям, которым дополнительные сообщения могут мешать при работе. Некоторые системы позволяют выбирать необходимый режим работы в зависимости от опыта пользователя.

Для помощи пользователю используются два метода:

- подсказка;
- запрос помощи.

Подсказка говорит пользователю, какие действия он может предпринять в данный момент. Чем опытнее пользователь, тем меньше он нуждается в подсказках. Поэтому неплохо, когда системы имеют несколько уровней подсказок, а человек сам выбирает наиболее удобный. Неопытные пользователи могут выбрать режим, при котором их «ведут за руку», в то время как опытные обходятся без подсказок.

Подсказки могут быть представлены в различных формах. Наиболее простая — вывод сообщения, указывающего, что делать дальше (например, «Введите координаты»). Синтезатор речи дает пользователю устные указания. Имеются также более тонкие формы подсказок, менее навязчивые для пользователя. Например, подсвечивать те кнопки, которые можно нажимать в данный момент. Если нужно задать позицию — высветить на экране следящее перекрестие, если необходимо ввести текст — сделать мигающим курсор и т. д. Такие ненавязчивые подсказки удобны для пользователя с опытом, а для новичков могут оказаться слишком сложными.

Кроме подсказки, может использоваться запрос помощи, который позволяет получать информацию о командах и способах их использования. Пользователь должен иметь возможность воспользоваться этой командой в любой момент диалога с компьютером и всегда одним и тем же способом. При возврате из режима помощи система должна оказаться в том состоянии, в каком она была до вызова команды.

## 5.5. Возможность исправления ошибок

Во время работы с компьютером неизбежно совершаются ошибки, поэтому требуются простые и удобные средства для их исправления, чтобы не наделать еще более серьезных ошибок.

Возможности исправления ошибок существуют на всех трех уровнях обратной связи: лексическом, синтаксическом и семантическом.

Большинство графических систем имеют клавишу возврата на одну позицию для удаления последней введенной буквы. Представьте себе, как тяжело было бы работать с компьютером без этого простого механизма исправления ошибок!

Многие системы также позволяют удалить всю текущую входную строку. Часто это служит способом прекращения выполнения команды до ее завершения. Система обычно возвращается в состояние, в котором она была до ввода удаленной команды.

Более глубокой является возможность полностью ликвидировать результаты выполнения последней команды вплоть до первой текущего сеанса редактирования, даже если и произошли изменения в файлах.

Отсутствие средств для исправления ошибок угнетает пользователя и снижает его производительность. Их наличие способствует исследованию неизученных возможностей системы без боязни ошибиться, так как ошибку легко исправить.

Возможность отмены результатов выполнения команды в некоторых системах обеспечивается путем запроса у пользователя явного подтверждения правильности команды: можно согласиться с ее результатами или потребовать их ликвидации. Но при таком подходе возрастают количество действий пользователя, затрачиваются больше времени.

Хорошей альтернативой явному подтверждению является неявное — пользователь подтверждает правильность команды путем ввода следующей команды.

Альтернативой отмене является запрос у пользователя подтверждения на выполнение команд, которые потом сложно отменить. Удаление файла — наилучший пример команд такого типа. Известно множество случаев, когда отсутствие подобного запроса приводило к печальным последствиям.

В процессе работы пользователю предъявляется большое количество информации, которую желательно структурировать. Для этого полезно разделить экран на несколько областей и в каждой области показывать информацию определенного типа. Подсказки, сообщения об ошибках, меню, графические данные могут изображаться в отдельных областях. Это помогает пользователю найти на экране нужную информацию. Некоторые системы даже разрешают пользователю выделять на экране области для организации обратной связи. Эти области могут перекрываться, но границы позволяют отделить одну область от другой.

Восприятие структуры изображения можно улучшить, используя различные способы визуального кодирования:

- цвет;
- тип линии;
- яркость.

Более подробные данные о способах кодирования графической информации приведены в табл. 5.1.

**Таблица 5.1. Способы кодирования графической информации**

<b>Метод кодирования</b>	<b>Максимальное количество кодов для безошибочного распознавания</b>
Цвет	6
Геометрические формы	10
Толщина линии	2
Тип линии	5
Яркость	2

Порядок перечисления методов соответствует снижению степени их эффективности. Можно комбинировать методы кодирования, что увеличивает степень выделяемости объектов. Любым из этих способов можно выделить движущийся объект или только что измененный объект.

Многие графические системы используют в общении различные пиктограммы. Для человека образное представление гораздо экономичнее словесного, отображать на экране компьютера слова ничуть не проще, чем небольшие формализованные изображения. Но полностью обойтись без слов можно крайне редко. Поэтому становятся актуальными проблемы адаптации, русификации и засорения языка новыми словами.

Современные средства интерфейса носят весьма сложный характер, и для их полного освоения требуется специальное обучение. Мощным двигателем прогресса в пользовании компьютером явилась стандартизация интерфейсных элементов как для представления информации, так и для общения с пользователем.

## **Часть II**

# **ГЕОМЕТРИЧЕСКОЕ МОДЕЛИРОВАНИЕ**

---

Во второй части мы рассмотрим вопросы получения геометрической модели объекта, дальнейшей работы с ней, проецирования на плоскость, удаления невидимых линий и получения реалистичного изображения.

# **Глава 6**

## **Общие сведения о геометрическом моделировании**

---

При решении большинства задач в области автоматизированного конструирования и технологической подготовки производства (ТПП) требуется иметь модель объекта проектирования. Вид этой модели, ее сложность и структура во многом определяют дальнейшие возможности этапов проектирования.

### **6.1. Геометрическая модель**

Модель объекта — это его некоторое абстрактное представление, удовлетворяющее условию адекватности объекту и позволяющее осуществлять его визуализацию и обработку с помощью компьютера.

Таким образом, модель — это набор данных, отображающих свойства объекта, и совокупность отношений между этими данными.

В модель объекта проектирования в зависимости от способа ее исполнения может входить ряд разнообразных характеристик и параметров. Чаще всего модели объектов содержат данные об их форме, размерах, допусках, применяемых материалах, механических, электрических, термодинамических и других характеристиках, способах обработки, стоимости, а также о микрогометрии (шероховатости, отклонения формы и размеров).

Для обработки модели в графических системах существенным является не весь объем информации об объекте, а та часть, которая определяет его геометрию, то есть формы, размер, пространственное размещение.

Описание объекта с точки зрения его геометрии называется *геометрической моделью объекта*. Но геометрическая модель может включать в себя еще и некоторую технологическую и вспомогательную информацию.

Информация о геометрических параметрах объекта используется не только для получения графического изображения, но и для расчетов различных характеристик объекта (например, по методу конечных элементов) и подготовки программ для станков с ЧПУ.

Под геометрическим моделированием понимают весь многоступенчатый процесс от описания объекта в соответствии с поставленной задачей до получения его внутренкомпьютерного представления (рис. 6.1).

В системах геометрического моделирования могут обрабатываться двумерные и трехмерные объекты, которые, в свою очередь, могут быть аналитически описываемыми и неописываемыми. Аналитически неописываемые геометрические элементы, такие как кривые и поверхности произвольной формы, используются преимущественно при описании объектов в автомобиле-, самолето- и судостроении.



Рис. 6.1. Схема геометрического моделирования

## 6.2. Основные виды геометрических моделей

Геометрические модели бывают двумерными и трехмерными (рис. 6.2).

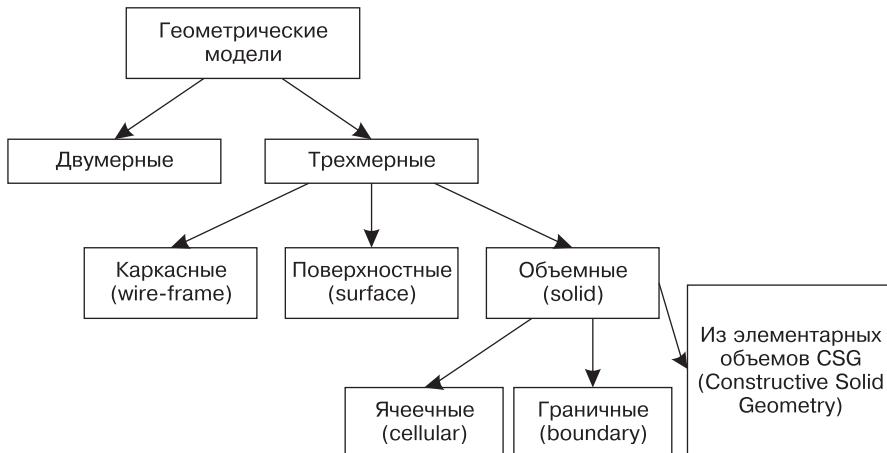


Рис. 6.2. Основные виды геометрических моделей

Двумерные модели, которые позволяют формировать и изменять чертежи, были первыми моделями, нашедшими применение. Такое моделирование применяется и до сих пор, так как оно намного дешевле (в отношении алгоритмов, использования) и вполне пригодно для решения разнообразных задач.

В большинстве двумерных систем геометрического моделирования описание объекта осуществляется в интерактивном режиме в соответствии с алгоритмами, аналогичными алгоритмам традиционного метода конструирования. Расширением таких систем является то, что контурам или плоским поверхностям ставится в соответствие постоянная или переменная глубина изображения. Системы, работающие по такому принципу, называются *2,5-мерными*. Они позволяют получать на чертежах аксонометрические проекции объектов.

Но двумерное представление часто неудобно для достаточно сложных изделий. При традиционных способах конструирования пользуются чертежами, где изделие может быть представлено несколькими видами. Если изделие очень сложное, его можно представить в виде макета. Трехмерная модель служит для создания виртуального представления изделия во всех трех измерениях.

В двумерных моделях работают с элементами, которые близки к уровню визуализации (отрезки, дуги). В более сложных моделях элементы ассоциируются

функционально. Например, размерные линии соотносятся с объектами, или же запоминается способ соединения элементов (к примеру, построение окружности, касательной к заданным прямым).

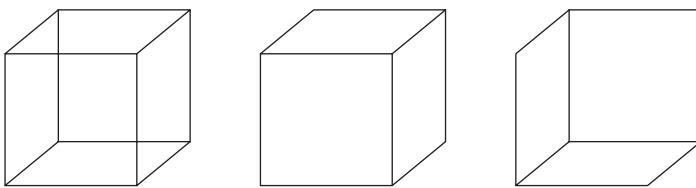
Структуры двумерных моделей весьма разнообразны и зависят от области применения.

Различают три вида трехмерных моделей:

- каркасные (проволочные);
- поверхностные (полигональные);
- объемные (модели сплошных тел).

Исторически первыми появились *каркасные модели*. В них хранятся только координаты вершин ( $x, y, z$ ) и соединяющие их ребра.

Представление модели в каркасном виде может быть воспринято неоднозначно (рис. 6.3).



**Рис. 6.3.** Интерпретация куба на базе каркасной модели

Так как известны только ребра и вершины, возможны различные интерпретации одной модели. Каркасная модель проста, но с ее помощью можно представить в пространстве только ограниченный класс деталей, в которых аппроксимирующие поверхности являются плоскостями. На основе каркасной модели можно получать проекции, но невозможно автоматически удалять невидимые линии и получать различные сечения.

В каркасной модели хранится информация двух типов:

- топологическая (ребра, определяемые вершинами);
- геометрическая (координаты вершин).

*Поверхностные модели* позволяют описывать достаточно сложные поверхности. Поэтому они часто соответствуют нуждам промышленности (автомобиле-, самолето-, судостроение) при описании сложных форм и работе с ними.

При построении поверхностной модели предполагается, что объекты ограничены поверхностями, которые отделяют их от окружающей среды. В модели объекта присутствуют ребра, но эти ребра являются результатом двух касающихся или пересекающихся поверхностей. Вершины объекта могут быть заданы пересечением поверхностей, множеством точек, удовлетворяющих какому-то геометрическому свойству, в соответствии с которым определяется контур.

Возможны различные виды поверхностей, что, в свою очередь, и определяет вид модели. Объекты можно разделить на ограниченные гранями, аппроксимированные гранями, криволинейные аналитически описываемые и криволинейные

аналитически неописываемые. Первые два вида описываются заданием граней. Криволинейные аналитически описываемые поверхности могут задаваться в модели с помощью уравнений. Для сложных поверхностей (аналитически неописываемых) используются различные математические модели — методы Кунса, Безье, Эрмита, *B*-сплайна. Они позволяют изменять характер поверхности с помощью параметров, смысл которых доступен пользователю, не имеющему специальной математической подготовки. Если же необходимы точный расчет и представление поверхности, используют аппроксимацию криволинейными аналитически описываемыми поверхностями.

Аппроксимация поверхностей общего вида плоскими гранями дает преимущество: для обработки таких поверхностей используются простые математические методы. Недостаток: сохранение формы и размеров объекта зависит от количества граней, используемых для аппроксимаций. Чем больше граней, тем меньше отклонение от действительной формы объекта. Но с увеличением количества граней одновременно возрастает и объем информации для внутренкомпьютерного представления. Вследствие этого увеличивается как время на работу с моделью объекта, так и объем памяти для хранения модели.

Если для модели объекта существенно разграничение точек на внутренние и внешние, то говорят об *объемных моделях*. Для получения таких моделей сначала определяются поверхности, окружающие объект, а затем они собираются в объемы.

В объемной модели хранится информация, позволяющая отличать материал от пустоты (при этом пустота может рассматриваться как особый вид материала). В настоящее время обычно используют два метода.

- Объект представлен в модели охватывающей его «оболочкой». Тогда, как и в каркасной модели, сохраняется информация топологического и геометрического типов, но она более полная (грани заданы и ориентированы таким образом, что известны их наружная и внутренняя стороны).
- Объект представлен в модели операциями построения. Сами операции обычно представлены в процедурной форме.

В рамках одного конкретного применения обычно используется не одна модель, а несколько. Во многих системах существуют геометрическая модель весьма высокого уровня (объемная или поверхностная) и модель для визуализации, которая дает возможность работать с информацией, близкой к чисто графической.

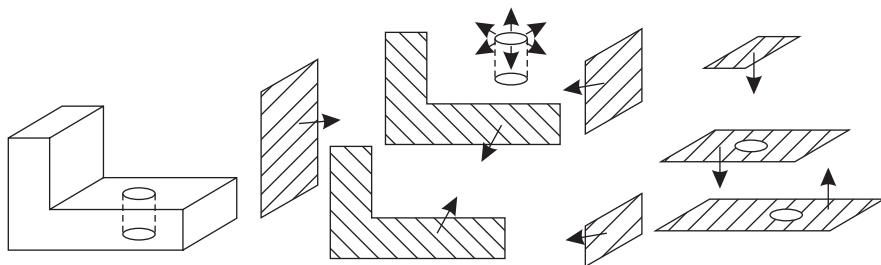
В настоящее время известно несколько способов построения объемных моделей.

В *границных моделях* объем определяется как совокупность ограничивающих его поверхностей.

Структура может быть усложнена внесением действий переноса, поворота, масштабирования (рис. 6.4).

Достоинствами данного способа построения модели являются:

- гарантия генерации правильной модели;
- большие возможности моделирования форм;



**Рис. 6.4.** Представление объекта граничной моделью<sup>1</sup>

- быстрый и эффективный доступ к геометрической информации (например, для прорисовки).

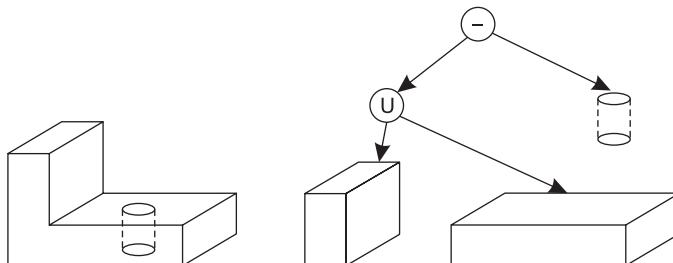
Среди недостатков можно отметить:

- больший объем исходных данных, чем при CSG-способе;
- модель логически менее устойчива, чем при CSG-способе, то есть возможны противоречивые конструкции;
- сложности построения вариаций форм.

В *CSG-моделях* объект определяется комбинацией элементарных объемов с использованием геометрических операций (объединение, пересечение, разность).

Под элементарным объемом понимается множество точек в пространстве.

Моделью такой геометрической структуры является древовидное представление (рис. 6.5). Узлы (нетерминальные вершины) — операции, а листья — элементарные объемы.



**Рис. 6.5.** Представление объекта CSG-моделью<sup>2</sup>

Достоинства этого способа:

- концептуальная простота;
- малый объем памяти;
- непротиворечивость конструкции;

<sup>1</sup> Климов В. Е. Графические системы САПР. — М.: Высшая школа, 1990.

<sup>2</sup> Там же.

- возможность усложнения модели;
- простота представления частей и сечений.

Недостатки:

- ограничение рамками булевых операций;
- вычислительноемкие алгоритмы;
- невозможность использовать параметрически описанные поверхности;
- сложность при работе с функциями более чем второго порядка.

В **ячеичных моделях** ограниченный участок пространства, охватывающий весь моделируемый объект, считается разбитым на большое количество дискретных кубических ячеек (обычно единичного размера).

Моделирующая система должна просто записать информацию о принадлежности каждого куба объекту.

Структура данных представляется трехмерной матрицей, в которой каждый элемент соответствует пространственной ячейке.

Достоинством данного метода является простота, недостатком — большой объем необходимой памяти.

Чтобы справиться с этим недостатком, используют принцип разбиения ячеек на подъячейки в особо сложных частях объекта и на границе.

Объемная модель объекта, полученная любым способом, является корректной, то есть в данной модели нет противоречий между геометрическими элементами, например отрезок не может состоять из одной точки.

Каркасное представление может быть использовано не при моделировании, а при отображении моделей (объемных или поверхностных) как один из методов визуализации.

### **6.3. Требования, предъявляемые к геометрическим моделям**

Цель геометрического моделирования — представление объектов. Эти объекты являются реальными и должны удовлетворять целому ряду требований.

Считается, что модель тем лучше, чем больше она учитывает ограничений, связанных с реальным объектом, его изготовлением и использованием. Так, например, двумерная модель, которая позволяет лишь формировать виды, составленные из отрезков прямой и дуг окружности, и даже не обеспечивает соответствия между этими видами, обладает чрезвычайно ограниченными возможностями. Модель, позволяющая в любой момент рассматривать представленные в ней объекты как сплошные тела, считается хорошей моделью с точки зрения геометрии.

Требования при геометрическом моделировании высокого уровня следующие:

- правильность модели (любая модель не должна противоречить реальному объекту);
- мощность модели (конструирование модели объекта целиком);
- возможность вычисления ряда геометрических величин (объема, площади);

- возможность использования различных функций (программы для станков с ЧПУ, расчет конструкций, расчет по методу конечных элементов).

Для удовлетворения этих требований необходимо, чтобы модель обладала определенным набором математических свойств:

- однородность (тело должно быть заполнено внутри);
- конечность (тело должно занимать конечную часть пространства);
- жесткость (сплошное тело должно сохранять свою форму независимо от положения и ориентации).

# **Глава 7**

## **Двумерное моделирование**

---

Двумерные модели являются наиболее простыми геометрическими моделями. Они используются чаще всего при работе с чертежами.

### **7.1. Типы данных**

Двумерная геометрическая модель оперирует данными следующих типов:

- геометрические (координаты точек, уравнения прямых, окружностей и т. д.);
- топологические (отрезок, соединяющий две точки; контур, определенный базовыми объектами; направления обхода и т. д.);
- структурные (комплекс состоит из базовых элементов, часто структурирование выполняется в виде дерева);
- оформительские (размерные линии, тексты, штриховка, условные обозначения);
- реляционные (отношения между элементами или их совокупностями). Например, элемент  $A$  касается элемента  $B$ .

### **7.2. Построение базовых элементов**

К базовым элементам относятся главным образом точка, отрезок прямой, прямая, дуга окружности, окружность, кривая, текст, контур.

Можно выделить несколько способов построения базовых элементов.

#### **Непосредственное задание с использованием выбранного синтаксиса представления**

При этом способе задания базовых элементов выбирается синтаксис их описания. Он может быть строго ограничен для каждого элемента или иметь несколько вариаций. Пользователь выбирает наиболее удобную для конкретного случая, и в соответствии с этим вводятся параметры элементов.

Например, отрезок может быть задан двумя точками; точка представляет собой пару координат  $(x, y)$ ; окружность задается точкой центра и радиусом и т. д.

#### **С помощью уравнений**

Базовые элементы задаются с помощью уравнений. Общее решение построения можно получить в два этапа.

1. Составляя систему алгебраических уравнений (на основе типов ограничений, элементов и параметров).

2. Решая эту систему (значения, характеризующие искомый элемент, принадлежат множеству решений системы уравнений).

Очевидным достоинством этого способа является его общность, так как для добавления нового ограничения достаточно написать соответствующие уравнения. Недостаток в том, что система уравнений может оказаться нелинейной. Поэтому требуются упрощения, а в некоторых случаях – интерактивный режим для нахождения приближенного решения.

### С помощью ограничений

Построение при ограничениях применяется к объекту. Ограничение определяется следующим образом:

*(тип элемента для построения) ((список ограничений), (тип элемента, к которому относятся ограничения))*

Основные типы ограничений:

- проходит через  $n$  точек;
- касается  $n$  объектов;
- параллелен другому объекту;
- образует некоторый угол с объектом;
- отстоит от другого объекта на некотором расстоянии.

Достоинство этого способа состоит в том, что не приходится прибегать к очень сложным методам вычислений. Поиск решений полностью управляем. Кроме того, можно организовать библиотеку модулей для каждого применения. Недостаток: для добавления нового ограничения или нового типа элемента нужно писать новые модули.

Например, пусть требуется построить окружность, касательную к заданным прямой и окружности, если известны радиус искомой окружности и примерное положение ее центра.

В зависимости от расположения заданных прямой и окружности искомая окружность может размещаться различным образом (рис. 7.1).

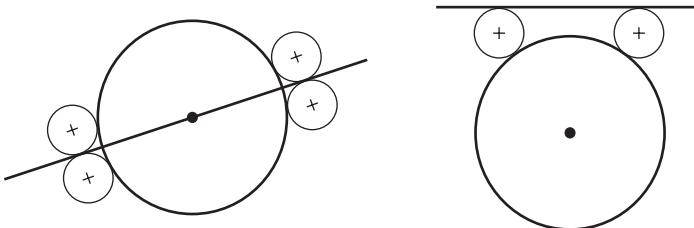
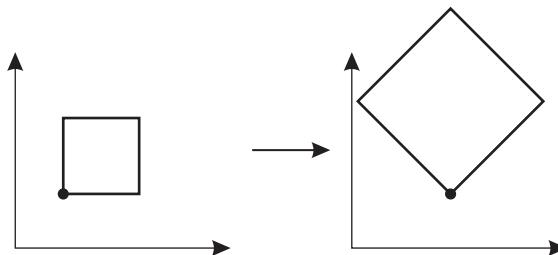


Рис. 7.1. Расположение окружностей и прямой

После выбранного способа анализируется примерное положение центра искомой окружности. Такая последовательность действий приводит в итоге к однозначному решению.

### **С использованием геометрических преобразований**

Новые элементы можно получать, выполняя геометрические преобразования (перенос, поворот, масштабирование) над уже имеющимися элементами или объектами. Для этого используются матрицы преобразования (рис. 7.2).



**Рис. 7.2.** Получение нового объекта путем геометрических преобразований ( $S(2, 2) \cdot R(45^\circ)$ )

## **7.3. Примеры моделей**

В качестве примеров рассмотрим наиболее популярные применения двумерных моделей.

### **Автоматизация черчения**

Под техническим черчением в данном случае понимается использование методов, аналогичных тем, которые традиционно применяются чертежниками, но с использованием средств информатики (дисплеи, средства диалога). Соответствующее программное обеспечение для компьютера дает возможность формировать и изменять (часто в интерактивном режиме) чертежи. Такой вариант модели представляет в расположение пользователя лишь совокупность двумерных элементов (обычно – отрезки и дуги).

В модели содержится только один вид объекта, что соответствует очень низкому уровню знаний о нем. Если сформировать несколько видов, то, как правило, в модели не представлены возможные отношения между видами. Поэтому всякое изменение в одном виде не находит отражения в других видах.

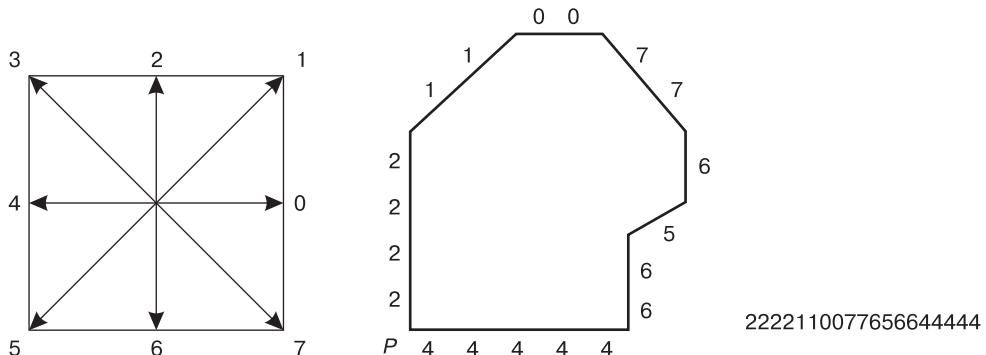
### **Параметризация**

Этот метод строится на основе понятия «семейство деталей». Под семейством деталей понимают набор деталей, состоящих из одинаковых элементов и различающихся лишь значениями некоторых параметров (в данном случае геометрических).

### **Цепное кодирование**

Данный способ позволяет представить линейный чертеж в детализированном виде на клетчатой поверхности. В результате дискретизации кривая описывается по-

следовательностью коротких элементарных векторов, ориентированных по восьми направлениям (рис. 7.3).



**Рис. 7.3.** Пример объекта и его код

Цепь представляет собой упорядоченную последовательность кодов векторов.

Длина вектора равна  $T(\sqrt{2})^P$ , угол наклона вектора равен  $n \cdot 45^\circ$ , где  $n$  — номер кода (от 0 до 7),  $T$  — размер ячейки сетки,  $P = \begin{cases} 0, & \text{если } n \text{ — четн.;} \\ 1, & \text{если } n \text{ — нечетн.} \end{cases}$ .

Таким образом, имея цепочку кодов, можно генерировать изображение. Для уменьшения длины цепочки кодов используются операторы повторения одинаковых элементарных векторов:

4·2 2·1 2·0 2·7 6 5 2·6 5·4

Также вводятся коды операций, которые определяют конец цепи, видимый участок, позволяют повторять какие-либо участки, вводить разные типы линий и т. д.

Такое описание объекта позволяет проводить ряд вычислений над обрабатываемыми цепочками:

- определение длины цепи;
- изменение направления обхода;
- вычисление площади поверхности;
- нахождение кратчайшей цепи;
- построение зеркальной цепи и т. д.

Цепной код удобен при построении сильно изломанных контуров (с многочисленными точками перегиба и очень малыми радиусами кривизны). Главный недостаток, ограничивающий его применение, заключается в том, что его примитивы принадлежат к слишком низкому уровню.

# **Глава 8**

## **Трехмерное моделирование**

---

Использование трехмерных моделей в процессе проектирования позволяет решить ряд сложных задач, так как они являются моделями более высокого уровня. Появляется возможность посмотреть на объект с любой стороны, получить качественное реалистичное изображение с учетом освещения, фактуры и рисунка поверхности, провести ряд сложных расчетов, написать программу для станка с ЧПУ.

### **8.1. Типы данных**

В случае трехмерного моделирования обрабатываемые элементы разнообразнее, чем в двумерном моделировании.

Базовые элементы делятся:

- на элементы нулевого уровня, то есть двумерные элементы (точки, отрезки, окружности, дуги, кривые, контуры);
- на элементы первого уровня, то есть поверхности (плоскости, линейчатые поверхности, поверхности вращения, криволинейные поверхности);
- на элементы второго уровня, то есть объемы (цилиндры, конусы, призмы, произвольные многоугольники, произвольные объемы).

Из этих элементов с помощью различных операций можно создавать комплексы. Мы уже говорили, что построение объемной модели может быть осуществлено двумя методами.

1. Представление объекта с помощью границ (границ, ребра, вершины).
2. Представление с помощью дерева построения (узлы представляют собой операции, листья — базовые объекты).

### **Представление с помощью границ**

Для представления объекта в виде совокупности плоских граней, ограниченных ребрами, которые, в свою очередь, ограничены вершинами, используются данные трех типов:

- геометрические (координаты вершин, уравнения ребер или поверхностей);
- топологические (связь между геометрическими данными);
- вспомогательные (атрибуты данных, например цвет грани, степень ее прозрачности).

Топологические и геометрические данные, как правило, не смешивают. В их разделении есть свои достоинства. Например, если нужно перенести объект, то

координаты вершин умножают на матрицу переноса. Топология объекта остается без изменения.

На рис. 8.1 приведена каркасная модель, в которой хранится информация только о ребрах и вершинах.

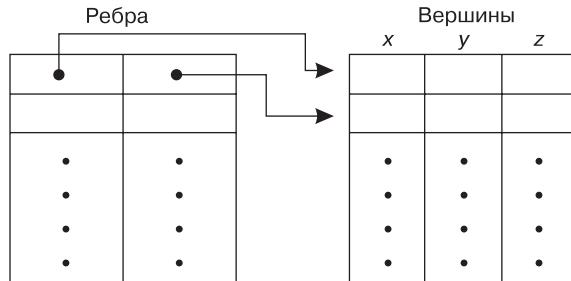


Рис. 8.1. Пример каркасной модели

На рис. 8.2 приведена поверхностная модель, которая является моделью более высокого уровня. В ней топологические и геометрические данные полностью разделены. Даны также информации о связи между ребрами и между гранями, что дает значительные дополнительные возможности при работе с такой моделью.

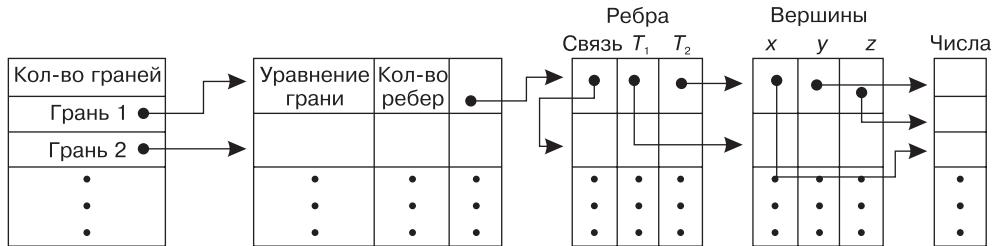


Рис. 8.2. Пример поверхностной модели

### Представление с помощью дерева

В качестве базовых элементов часто используются элементарные сплошные тела. Деталь мысленно разбивается на элементы, модели которых уже имеются в системе.

Обычно применяются следующие операции:

- объединение;
- пересечение;
- вычитание.

Упрощенным вариантом этой модели может быть случай, когда нужно разместить некоторое количество тел, что часто случается при архитектурном проектировании и выполнении планировки цеха. Тогда модель можно представить в виде двух таблиц, в одной из которых размещены модели конструктивных элементов,

а в другой — информация об их вставке (точка вставки, угол поворота и масштабные коэффициенты) (рис. 8.3).

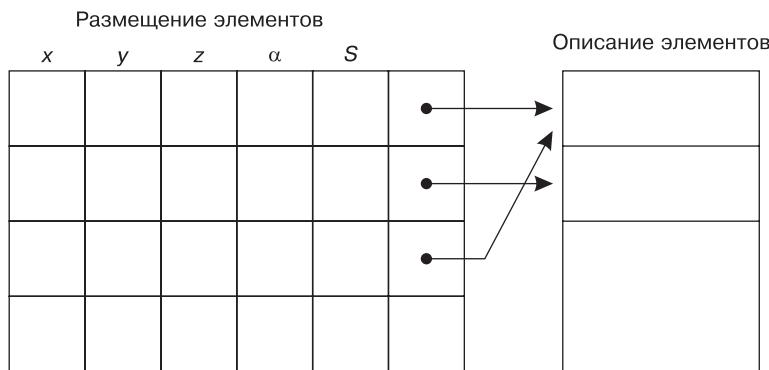


Рис. 8.3. Пример модели CSG

## 8.2. Методы описания трехмерных объектов

Далеко не последнее место при составлении модели занимает то, как пользователь будет вводить информацию об объекте, то есть каким методом он будет ее описывать. Этот процесс нужно формализовать таким образом, чтобы описание объекта было несложным и близким к естественному языку пользователя.

Возможны несколько методов описания трехмерных объектов.

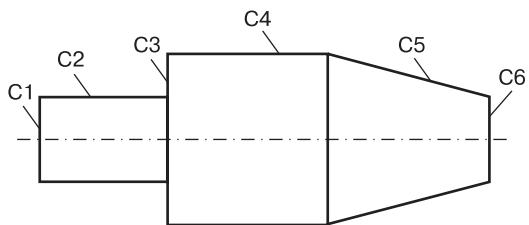
### Описание геометрии объекта с использованием алфавитно-цифрового входного языка

В большинстве систем, оперирующих элементарными объемами, конструирование изделий происходит последовательно, по принципу агрегатирования. Определение элементарных объемов и их синтез проводятся с использованием формализованного языка. С его помощью можно описать размеры элементарных объемов, способ из соединения, установку системы координат и т. д. Формализованный язык может содержать не только геометрические параметры, но и информацию конструктивного и технологического плана (простановка размеров, шероховатость поверхностей, отклонение, допуск). Объект (рис. 8.4) может иметь следующее формализованное описание:

```
C1=TOREC1, D100;
C2=CIL, D100, L200;
C3=TOREC2, DM100, DB200;
C4=CIL, D200, L200;
C5=KONUS, DB200, DM100, L150;
C6=TOREC1, D100;
```

Использование элементарных объемов снижает затраты на описание объекта, так как нет необходимости описывать отдельные контуры или поверхности, они определяются с помощью одного оператора входного языка.

Недостаток данного метода в том, что пользователю нужно освоить специфику используемого формализованного языка.



**Рис. 8.4.** Пример объекта

### Описание объекта в режиме графического диалога

Это возможно только при наличии соответствующих технических средств — дисплеев для генерации динамических изображений, устройств ввода графической информации и программного обеспечения. Например, создание в AutoCAD библиотеки конструктивных элементов (КЭ). Конструирование осуществляется в диалоге путем соединения КЭ с нужными размерами.

### Получение модели объекта путем ввода эскизов и восстановления модели по имеющимся проекциям

Этот метод соответствует традиционным методам конструирования и осуществляется в два этапа.

1. Ввод эскизов.
2. Восстановление модели.

Для решения таких задач в структуре программного модуля предусмотрены специальные процессоры:

- процессор для обработки эскизов (осуществляет ввод эскизов и на их основе формирует точный контур);
- процессор восстановления (создает проекции и по ним генерирует объемную модель);
- процессор генерации изображения (осуществляет графический вывод точно-го контура, проекций, проволочной и объемной моделей).

Ввод эскизов и их обработка предполагают наличие соответствующего технического и программного обеспечения. В качестве устройства ввода графической информации можно использовать специальное графическое устройство, в котором поверхность съема информации разделена на части для вычерчивания фронтальной, профильной и горизонтальной проекций.

**Этап 1.** Ввод эскиза проекции (рис. 8.5). Система распознавания образов разделяет эскиз проекции на контурные элементы. Эти элементы относятся к аналитически описываемым (отрезок, окружность, дуга). Основой для такого распознавания является упорядоченная последовательность цифровых точек контура проекции. Далее контур, а точнее его точечный образ, преобразуется в последовательность векторов,

которые объединяются в группы в соответствии со своей направленностью. Каждая группа сопоставляется с геометрическими элементами (отрезок, окружность, дуга). Между этими элементами устанавливаются отношения, и с помощью полученных точек пересечения определяют точный контур.

**Этап 2.** Получение точного контура проекции и генерация вспомогательных линий (рис. 8.6).

**Этап 3.** Ввод эскизов остальных проекций (рис. 8.7).

**Этап 4.** Получение полного комплекта проекций (рис. 8.8).

**Этап 5.** Получение каркасной модели (рис. 8.9).

Точки пространственной модели получаются при сопоставлении координат точек проекций. Точки модели соединяются соответствующими контурными элементами.

**Этап 6.** Получение объемной модели (рис. 8.10). Из контурных элементов конструируются поверхности, которые затем объединяются в объемы.

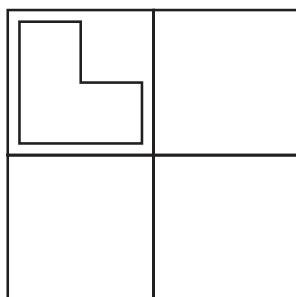


Рис. 8.5. Первый этап

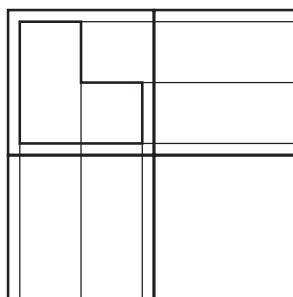


Рис. 8.6. Второй этап

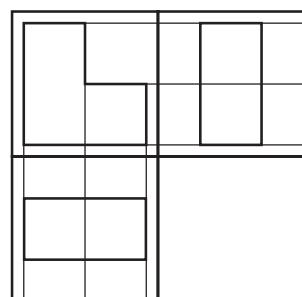


Рис. 8.7. Третий этап

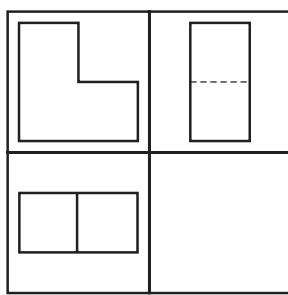


Рис. 8.8. Четвертый этап

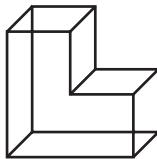


Рис. 8.9. Пятый этап

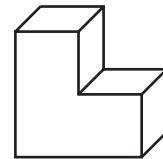


Рис. 8.10. Шестой этап

### 8.3. Методы построения трехмерных моделей

На базе описания объекта строится его трехмерная модель. В зависимости от вида объекта, его сложности и дальнейшего использования могут применяться разные методы построения модели.

## Построение кривых и поверхностей

Способы построения кривой:

- интерполяция по точкам;
- деформация кривой (перемещение точки, изменение полинома);
- вычисление эквидистанты заданной кривой;
- формирование кривой из отрезков и дуг;
- вычисление различных сечений;
- пересечение поверхностей.

Способы построения поверхности:

- интерполяция по точкам;
- деформация поверхности;
- перемещение образующей кривой по заданной траектории;
- вычисление эквидистантной поверхности на заданном расстоянии.

Эти способы построения часто применяются для объектов сложной формы в автомобильной, авиационной промышленности и судостроении (формы Эрмита, Безье, *B*-сплайны). В машиностроении же зачастую используются объекты, описание которых можно выполнить аналитически. Многие объекты можно представить плоскими гранями или аппроксимировать ими более сложные поверхности.

### Задание гранями (кусочно-аналитическое описание)

Модель представляет собой пятиуровневую иерархическую структуру. Тело представляется множеством ограничивающих его граней:  $T = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$ . Каждая грань задается множеством ограничивающих ее ребер  $\alpha = \{l_1, l_2, \dots, l_m\}$  и нормалью  $\vec{n}_\alpha$ , направленной из тела; каждое ребро — двумя точками:  $l = \langle p1, p2 \rangle$ ; а каждая точка — тремя координатами:  $p = (x, y, z)$ . Для реализации аналитических операций над гранями для каждой грани задаются четыре коэффициента:  $A, B, C, D$ , однозначно определяющие уравнение плоскости.

Модель может быть реализована в виде графа (рис. 8.11).

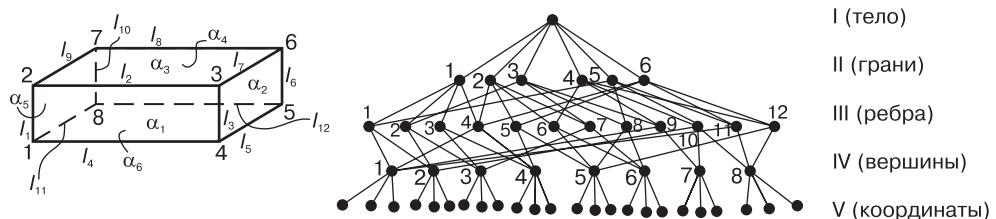


Рис. 8.11. Пример кусочно-аналитического описания

Можно определить, какой объем памяти нужен для хранения этой модели в машинных словах. Считаем, что для записи одной ссылки и одного числа используется 1 байт.

Координаты вершин — 24 байта.

Коэффициенты уравнений плоскостей граней — 24 байта.

Ссылки первого уровня — 6 байт.

Ссылки второго уровня — 24 байта.

Ссылки третьего уровня — 24 байта.

Ссылки четвертого уровня — 24 байта.

Итого — 126 байт.

Причем первые четыре уровня содержат информацию топологического типа, а пятый (координаты вершин) — геометрического.

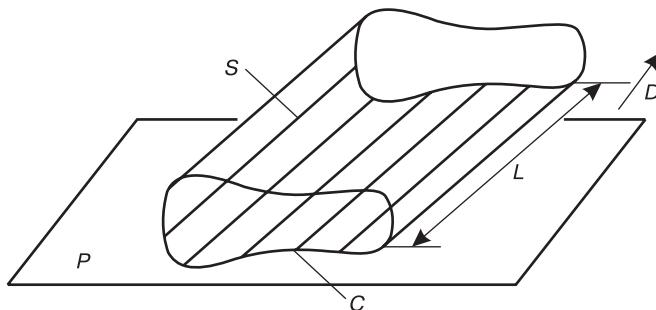
### **Кинематический принцип**

Модель тела может быть задана с применением кинематического принципа — вращением или сдвигом.

Задание сдвигом:

$$S = F1(C, P, D, L).$$

Контур  $C$ , помещенный в плоскости  $P$ , порождает тело  $S$  путем переноса по направлению  $D$  на расстояние  $L$  (рис. 8.12).

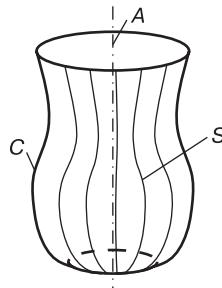


**Рис. 8.12.** Задание сдвигом

Задание вращением:

$$S = F2(C, A, \alpha).$$

Тело получается путем вращения контура  $C$  вокруг оси  $A$  (рис. 8.13).



**Рис. 8.13.** Задание вращением

## Булевы операции

Модель представляет собой дерево, узлы которого — операции, а листья — базовые элементы. Каждый базовый элемент имеет свою геометрию и топологию и представлен в виде геометрической модели (каркасной, поверхностной, объемной).

При вызове базового элемента для присоединения к объекту он должен в общем случае обладать следующими атрибутами:

$$\langle X, Y, Z, \alpha_x, \alpha_y, \alpha_z, S_x, S_y, S_z \rangle,$$

где  $X, Y, Z$  — координаты точки привязки ЛСК и ГСК;

$\alpha_x, \alpha_y, \alpha_z$  — углы поворота ЛСК относительно ГСК;

$S_x, S_y, S_z$  — параметры элемента.

В некоторых ситуациях задача может упрощаться. Например, при конструировании тела вращения (вала) отпадает надобность в задании углов и точки привязки, так как элементы соединяются строго по оси друг за другом. В таком случае необходима информация только о параметрах элементов.

Чаще всего при конструировании объекта используются следующие операции над базовыми элементами:

- объединение;
- пересечение;
- разность.

В настоящее время существуют два метода геометрического объединения:

- контактного соединения;
- соединения с проникновением.

*Метод контактного соединения* применяется при наличии у тел плоских поверхностей, по которым они могут быть соединены. Далее проводится анализ граничных контуров поверхностей, по которым соединяются тела. Существует много разных методик, но общими действиями обычно остаются следующие:

- аналитическое описание граничных контуров двух тел;
- определение точек пересечения контуров и их сегментация;
- анализ вершин.

Метод контактного соединения прост в реализации. Затраты на программирование и время обработки программ невелики.

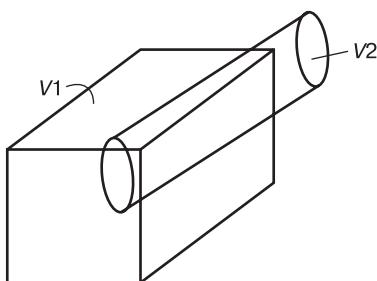
*В методе соединения с проникновением* используются не контурные плоские элементы, а поверхностные и соответственно рассчитываются кривые пересечения. Расчет кривых пересечения требует больших затрат, так как для каждой комбинации поверхностей нужно разрабатывать свой алгоритм.

Обычно существуют два пути:

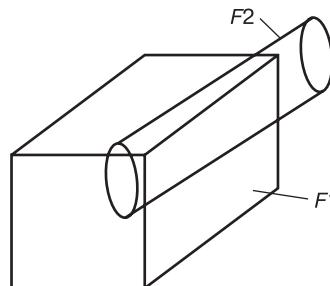
- когда поверхности заданы аналитически (но в таком случае есть ограничение — поверхности должны быть максимум второго порядка);
- численное определение кривой пересечения.

### Этапы метода соединения с проникновением

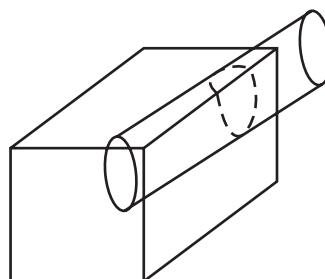
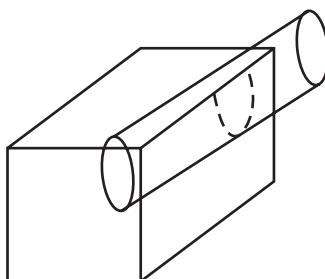
**Этап 1.** Определение объемов  $V1$  и  $V2$  на основе математического представления поверхностей, образующих эти объемы (рис. 8.14).



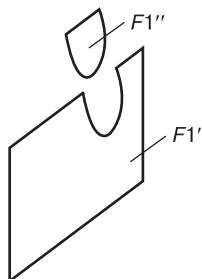
**Рис. 8.14.** Первый этап метода соединения с проникновением<sup>1</sup>



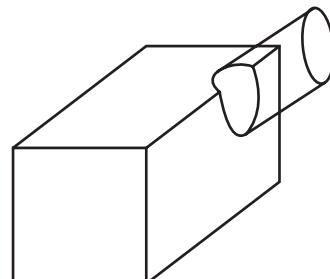
**Рис. 8.15.** Второй этап метода соединения с проникновением<sup>2</sup>



**Рис. 8.16.** Третий этап метода соединения с проникновением<sup>3</sup>



**Рис. 8.17.** Четвертый этап метода соединения с проникновением<sup>4</sup>



**Рис. 8.18.** Пятый этап метода соединения с проникновением<sup>5</sup>

<sup>1</sup> Климов В. Е. Графические системы САПР. — М.: Высшая школа, 1990.

<sup>2</sup> Там же.

<sup>3</sup> Там же.

<sup>4</sup> Там же.

<sup>5</sup> Там же.

**Этап 2.** Определение пар потенциально пересекающихся поверхностей ( $F_1$  и  $F_2$ ) (рис. 8.15).

**Этап 3.** Аналитическое определение кривой пересечения для пары пересекающихся поверхностей (рис. 8.16).

**Этап 4.** Сегментация поверхности в соответствии с полученной кривой (рис. 8.17).

$$F_1 \rightarrow F_1' + F_1''.$$

**Этап 5.** Удаление сегментов поверхностей (рис. 8.18).

### Полигональные сетки

*Полигональной сеткой* называют совокупность связанных между собой плоских многоугольников, с помощью которых можно аппроксимировать сложные криволинейные поверхности. Недостаток метода — его приближенность (рис. 8.19).

Для улучшения качества можно увеличить количество многоугольников для аппроксимации, но это приведет к дополнительным затратам памяти и вычислительного времени.

Полигональные сетки на данный момент являются самым часто используемым представлением, для которого существует большое количество программного обеспечения, позволяющего создавать, редактировать и визуализировать модели.

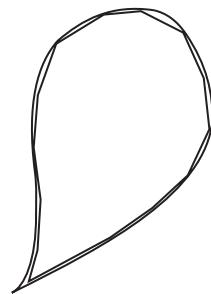
Характерной особенностью полигональных сеток является поддержка связности модели. В силу этого полигональные представления хорошо применимы для описания большого числа синтетических поверхностей.

Существуют три способа описания многоугольников полигональной сетки.

**Явное задание многоугольников.** Каждый многоугольник можно задать в виде списка координат его вершин:

$$P = \left( (x_1, y_1, z_1), (x_2, y_2, z_2), \dots, (x_n, y_n, z_n) \right).$$

Вершины в списке находятся в том порядке, в котором они встречаются при обходе многоугольника. Многоугольник получается путем последовательного соединения всех вершин ребрами, а также первой и последней. Для каждого отдельного многоугольника данный способ записи является эффективным, но для полигональной сетки дает большие потери памяти вследствие дублирования информации о координатах общих вершин. Кроме того, нет явного описания общих ребер и вершин. Например, поиск всех многоугольников, имеющих общую вершину, требует сравнения троек координат одного многоугольника с тройками координат всех остальных многоугольников. Наиболее эффективный способ выполнить такое сравнение — просортировать все  $N$  троек координат, для чего потребуется в общем случае  $M \log_2 N$  сравнений. Но и при этом существует опасность, что одна и та же вершина вследствие ошибок округления может в разных многоугольниках иметь

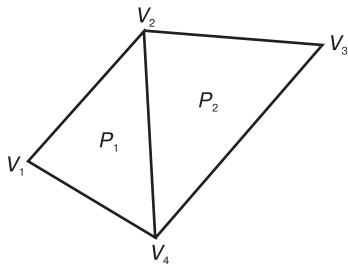


**Рис. 8.19.**  
Поперечное сечение  
криволинейного объекта  
и его полигональная  
аппроксимация

различные значения координат, поэтому правильное соответствие может быть никогда не найдено. Полигональная сетка изображается путем вычерчивания ребер каждого многоугольника, однако это приводит к тому, что общие ребра рисуются дважды, что является еще одним недостатком.

**Задание многоугольников с помощью указателей на вершины.** Каждая вершина запоминается лишь один раз в списке вершин:

$$V = \{(x_1, y_1, z_1), (x_2, y_2, z_2), \dots, (x_n, y_n, z_n)\}.$$



Многоугольник определяется списком указателей на вершины:

$$P = \{v_1, v_2, \dots, v_k\},$$

где  $v_i$  — указатель на  $i$ -ю вершину списка  $V$ .

Для полигональной сетки (рис. 8.20) описание будет следующим:

$$V = (V_1, V_2, V_3, V_4) = \{(x_1, y_1, z_1), \dots, (x_4, y_4, z_4)\}.$$

**Рис. 8.20.** Пример полигональных многоугольников при задании с помощью указателей на вершины

$$P_1 = (v_1, v_2, v_4);$$

$$P_2 = (v_1, v_2, v_3).$$

При таком способе представления полигональной сетки первые два недостатка, свойственные предыдущему методу, исключаются, остается лишь повторная отрисовка ребер.

**Явное задание ребер.** В этом представлении тоже задается список вершин:

$$V = \{(x_1, y_1, z_1), \dots, (x_n, y_n, z_n)\}.$$

Затем задается список ребер, где каждое ребро указывает на две вершины в списке вершин, определяющие это ребро, а также на один или два многоугольника, которым это ребро принадлежит. Если ребро принадлежит одному многоугольнику, то либо  $P_1$ , либо  $P_2$  пусто.

$$E = (v_1, v_2, p_1, p_2),$$

где  $v_i$  — указатель на  $i$ -ю вершину списка  $V$ .

После этого описываются все многоугольники в виде списка указателей на ребра:

$$P = \{e_1, e_2, \dots, e_m\},$$

где  $e_i$  — указатель на  $i$ -е ребро.

Для полигональной сетки (рис. 8.21) описание имеет вид:

$$V = (V_1, V_2, V_3, V_4) = \{(x_1, y_1, z_1), \dots, (x_4, y_4, z_4)\}.$$

$$E_1 = (v_1, v_2, p_1, \lambda);$$

$$E_2 = (v_2, v_3, p_2, \lambda);$$

$$E_3 = (v_3, v_4, p_2, \lambda);$$

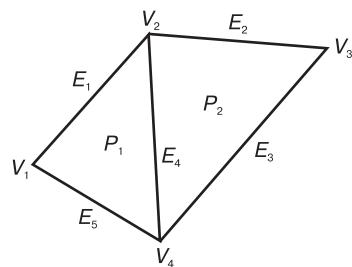
$$E_4 = (v_4, v_2, p_1, p_2);$$

$$E_5 = (v_4, v_1, p_1, \lambda).$$

$$P_1 = (e_1, e_4, e_5);$$

$$P_2 = (e_2, e_3, e_4).$$

Полигональная сетка изображается путем вычерчивания не всех многоугольников, а всех ребер. В результате этого многократной отрисовки ребер не происходит.



**Рис. 8.21.** Пример полигональных многоугольников при явном задании ребер

### Октаантовые деревья

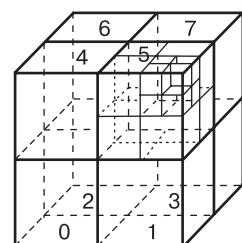
Октаантовые деревья занимаются визуализацией наборов данных, изначально организованных как трехмерные массивы. Чаще всего такая информация приходит с различных сканирующих устройств (например, в медицине) или в результате численного эксперимента (например, в динамике жидкых сред).

Модель строится из трехмерного массива элементов объема — *вокселов* (voxels). Все воксели имеют форму куба и одинаковый размер по трем координатным осям. Октаантовое дерево организует иерархическую структуру вокселов. Весь объект хранится в корневом кубе. Этот куб делится на восемь подкубов плоскостями, перпендикулярными осям координат. Подкубы, в свою очередь, снова делятся на подкубы и т. д. (рис. 8.22). Данный процесс заканчивается на уровне вокселов — элементарных неделимых единиц объема. При такой схеме кодирования воксельного пространства октаантовое дерево может представлять кубический объем пространства из  $(2^N)^3$  вокселов, где  $N$  — глубина дерева.

Каждый воксель внутри модели может быть либо прозрачным («воздух»), либо принадлежать поверхности объекта («тело»). Воксели «воздуха» — это прозрачные области пространства, они не нужны для представления модели. Структура октаантового дерева позволяет избегать хранения «воздушных» подкубов на каждом уровне иерархии, в результате чего существенным образом уменьшается объем памяти, требуемый для хранения моделей.

Воксели «тела» содержат информацию о свойствах поверхности в точке: цвет, внешняя нормаль, коэффициент отражения/преломления и т. д. — для проведения необходимых вычислений во время построения изображения.

Для того чтобы хранить информацию только о поверхности объекта, каждый непустой подкуб имеет один байт специальной информации (инфобайт). Каждый бит этого байта говорит, содержит соответствующий ему подкуб воксели поверхности или нет. Только те подкубы, которые содержат



**Рис. 8.22.** Иерархическое октаантовое разбиение пространства

поверхность, будут иметь потомков на нижних уровнях иерархии. Подбоксы «воздуха» не отслеживаются вниз по дереву.

В сравнении с обычным однородным представлением трехмерного пространства, где каждый воксел содержит информацию, а для хранения всего пространства требуются гигантские объемы памяти, октантные деревья являются существенно более эффективными. Например, для хранения объектов, состоящих из  $256^3$  вокселов, требуется до 98 Мбайт: каждый воксел содержит 3 байта для хранения цвета поверхности плюс 3 байта для хранения нормали плюс 1/8 байта на тип вокселя («воздух» — «тело»), итого 6,125 байт/вокsel. Прямое перемножение  $256^3$  на 6,125 дает 98 Мбайт информации. Применение октантных деревьев позволяет сократить этот объем до 12,5 Мбайт.

## Глава 9

# Описание и характеристика поверхностей

Одним из вариантов представления трехмерных поверхностных и объемных моделей является аналитическое описание поверхностей, ограничивающих объект.

## 9.1. Описание поверхностей

Наиболее часто для аналитического представления поверхности используется параметрическое описание или описание неявными функциями. В некоторых особых случаях может понадобиться поточечное описание.

### Параметрическое описание

Поверхности, заданные в виде:

$$\begin{cases} X = X(u, t); \\ Y = Y(u, t); \\ Z = Z(u, t), \end{cases}$$

где  $u, t$  — параметры, изменяющиеся в заданных пределах, относятся к классу параметрических.

Для одной фиксированной пары  $(u, t)$  можно вычислить положение только одной точки поверхности. Для полного представления всей поверхности необходимо с определенным шагом перебрать множество пар  $(u, t)$  из диапазона их изменений, вычисляя при этом  $X, Y, Z$ .

Плоскость, проходящая через точку  $(x_0, y_0, z_0)$  и векторы  $\vec{n}_1$  и  $\vec{n}_2$ , исходящие из этой точки, определяется:

$$\begin{cases} x = x_0 + u \cdot n_{1x} + t \cdot n_{2x}; \\ y = y_0 + u \cdot n_{1y} + t \cdot n_{2y}; \\ z = z_0 + u \cdot n_{1z} + t \cdot n_{2z}, \end{cases}$$

где  $n_{mx}, n_{my}, n_{mz}$  — проекции  $\vec{n}_m$  ( $m = 1, 2$ ) на оси  $OX, OY, OZ$ .

Приведенное уравнение опишет прямоугольник со сторонами длиной  $|\vec{n}_1|$  и  $|\vec{n}_2|$ , если единичные векторы  $\vec{n}_1$  и  $\vec{n}_2$  будут перпендикулярны друг другу, а параметры  $u$  и  $t$  изменяются от 0 до 1.

Нормаль  $\vec{N}$  к плоскости, заданной параметрически, может быть определена как:

$$\vec{N} = \vec{n}_1 \cdot \vec{n}_2.$$

Эллипсоид вида:

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 + \left(\frac{z}{c}\right)^2 = 1$$

в параметрическом виде записывается так:

$$\begin{cases} x = a \sin \theta \cos \varphi; \\ y = b \sin \theta \sin \varphi; \\ z = c \cos \theta, \end{cases}$$

где  $\theta$  — долгота,  $\varphi$  — широта.

Нормаль к поверхности эллипса:

$$N = ibc \sin \theta \cos \varphi i + jca \sin \theta \sin \varphi j + kab \cos \theta k.$$

В геометрическом моделировании важны бикубические параметрические поверхности. Это простейшие поверхности, с помощью которых достигается непрерывность составной функции и ее первых производных. То есть функция из нескольких смежных бикубических участков будет обладать непрерывностью и гладкостью в местах стыка. Они похожи на гладкие изогнутые четырехугольники, представление о которых могут дать листы металла, бумаги. Такие поверхности могут описывать любые геометрические формы.

Недостатки параметрического описания:

- трудоемкость описания;
- большие вычислительные затраты (нужны численные методы вычисления);
- параметрическое описание подразумевает, что исходной позицией светового луча, строящего изображение, является точка на объекте. Это затрудняет применение алгоритмов компьютерного синтеза изображений, предполагающих иную начальную позицию луча (например, метод трассировки лучей), что ведет к ухудшению изображений: отсутствие теней, прозрачности и зеркального отражения соседних объектов.

Достиныства параметрического описания следующие.

- Возможность передачи геометрической формы очень сложных поверхностей, например винтообразной улитки. Она представляет собой сумму трех векторов: первый — вокруг которого завивается улитка, конец второго очерчивает спираль, а начало скользит по первому, третий — начало которого скользит по спирали, а конец вращается вокруг спирали.

Описание тора, симметричного относительно оси  $OZ$  и плоскости  $XOY$ :

$$\begin{cases} x = (R + a \cos u) \cos \theta; \\ y = (R + a \cos u) \sin \theta; \\ z = a \sin u, \end{cases}$$

где  $a$  — радиус кольцевого «баллона» тора;

$R$  — расстояние от центра тора до оси «баллона»;

$\varphi$  изменяется в пределе  $[0, 2\pi]$ ;

$\theta$  изменяется в пределе  $[0, \pi]$ .

Неявное описание типа  $f(X, Y, Z) = 0$  этих и многих других поверхностей невозможно.

- Приспособленность к физическим процессам управления резцом в станках с ЧПУ. Резец должен вытаскивать деталь, двигаясь в пространстве по законам, заданным в параметрической модели.
- Параметрические поверхности легко ограничиваются в пространстве пределами изменения параметров. Например, наружная поверхность дольки апельсина в виде  $1/8$  шара радиуса  $r$ :

$$\begin{cases} x = r \sin \theta \cos \varphi; \\ y = r \sin \theta \sin \varphi; \\ z = r \cos \theta, \end{cases} \text{ где } \varphi = \left[0, \frac{\pi}{4}\right], \theta = \left[-\frac{\pi}{2}, \frac{\pi}{2}\right].$$

### Описание неявными функциями

Заключается в моделировании поверхностей в виде:

$$f(X, Y, Z) = 0.$$

Могут быть использованы функции различных порядков, но из-за сложности математической обработки ограничиваются первой и второй степенью. Поверхности, описываемые функциями третьей и четвертой степени, незначительно расширяют возможности геометрической имитации форм, а вычисления резко увеличиваются.

Поверхности первого порядка, описывающие плоскости:

$$AX + BY + CZ + D = 0.$$

Поверхности второго порядка, описывающие две плоскости, конусы, гиперболоиды, параболоиды и эллипсоиды:

$$AX^2 + BY^2 + CZ^2 + 2DXY + 2EYZ + 2FZX + 2GX + 2HY + 2JZ + K = 0.$$

Описание неявными функциями удобно для использования в методе твердотельного описания объектов и при трассировании лучей, так как легко определить взаимное положение точки и поверхности такого типа, пересечение прямой и плоскости.

### Поточечное описание

Поверхность представляется множеством отдельных точек, принадлежащих этой поверхности. Теоретически при бесконечном увеличении количества точек такая модель обеспечивает непрерывную форму описания.

Поточечное описание применяют в случаях, когда поверхность очень сложная, не обладает гладкостью, а детальное представление многочисленных геометрических особенностей важно. Например, участки грунта на других планетах; формы малых небесных тел, информация о которых доставлена с искусственного спутника в виде нескольких стереопар; микрообъекты, снятые с помощью микроскопов.

Исходная информация представляется в виде матрицы трехмерных координат точек. Они определяются автоматизированными методами на стереоприборах. Часто используется сопоставление стереопар. Необходимо учитывать требуемую частоту расположения точек.

К недостаткам данного метода можно отнести:

- отсутствие информации о поверхности между точками. В полигональных сетках предполагается, что между точками находятся участки плоскостей;
- трудоемкость снятия данных с объекта;
- большие вычислительные затраты;
- большой объем исходных данных.

## 9.2. Характеристики поверхностей

Наиболее часто в представлении поверхностей используются поверхности первого и второго порядка. Поверхности более высоких порядков почти не используются из-за сильно возрастающей сложности обработки.

### Поверхности первого порядка

Поверхности вида:

$$f_1(X, Y, Z) = AX + BY + CZ + D = 0$$

в матричном представлении имеют вид:

$$\begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \cdot P = 0, \text{ где } P = \begin{bmatrix} A & B & C & D \end{bmatrix}^T.$$

Изменяя компоненты матрицы  $P$ , можно описать плоскость любой ориентации и положения. Это будет бесконечная плоскость. Реальный участок имеет ограничения. Наиболее удобно ограничение выпуклым многоугольником. Все другие случаи, как криволинейного ограничения, так и ограничения невыпуклой фигурой, могут быть сведены к первому путем аппроксимации или разбиения на выпуклые подфигуры. Границные точки многоугольника оцифровываются, и их координаты записываются в матрицу:

$$\Lambda = \begin{bmatrix} X_1 & Y_1 & Z_1 \\ \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n \end{bmatrix}, \text{ где } n \geq 3.$$

Уравнение плоскости определяют на основе трех точек, не лежащих на одной прямой:

$$\begin{vmatrix} X - X_1 & Y - Y_1 & Z - Z_1 \\ X_2 - X_1 & Y_2 - Y_1 & Z_2 - Z_1 \\ X_3 - X_1 & Y_3 - Y_1 & Z_3 - Z_1 \end{vmatrix} = 0.$$

Нормаль  $\vec{N} : N = iA + jB + kC$ .

Нормаль направлена в сторону полупространства, где значение скалярного поля  $f_1(X, Y, Z) > 0$ .

Из поверхностей первого порядка составляются полигональные сетки (серия смежных многоугольников, не имеющих разрывов между собой; каждое ребро является общим для смежных многоугольников).

Описываемая функция обладает непрерывностью, а производная имеет разрывы в местах стыка участков.

Достоинством данной функции является простота обработки.

### **Поверхности второго порядка**

Описываются формулой:

$$\begin{aligned} f_2(X, Y, Z) = & AX^2 + BY^2 + CZ^2 + 2DXY + \\ & + 2EYZ + 2FZX + 2GX + 2HY + 2JZ + K = 0. \end{aligned}$$

Все поверхности, кроме эллипсоида, бесконечны. Поэтому только эллипсоид может самостоятельно образовывать объемный примитив, все другие требуют ограничения в пространстве.

Квадратичная функция в матричном виде:

$$f_2(X, Y, Z) = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \cdot P \cdot \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T, \text{ где } P = \begin{bmatrix} A & D & F & G \\ D & B & E & H \\ F & E & C & J \\ G & H & J & K \end{bmatrix}.$$

Нормаль  $\vec{N}$  к поверхности в точке  $(X, Y, Z)$  определяется:

$$\vec{N} = \Omega \Psi,$$

$$\text{где } \Psi = \begin{bmatrix} AX + DY + EZ + G \\ BY + DX + EZ + H \\ CZ + FX + EY + J \end{bmatrix}, \quad \Omega = \begin{bmatrix} i & j & k \end{bmatrix}; i, j, k - \text{орты осей } OX, OY, OZ.$$

Нормаль  $\vec{N}$  направлена по градиенту скалярного поля  $f_2(X, Y, Z)$ , то есть в сторону увеличения значений  $f_2(X, Y, Z)$ . Так как функция  $f_2(X, Y, Z)$  является монотонной и однократно знакопеременной, то  $\vec{N}$  направлена в ту часть подпространства, где  $f_2 > 0$ . Например,  $\vec{N}$  к поверхности шара ( $X^2 + Y^2 + Z^2 - 1 = 0$ ) направлена внутрь шара, а  $\vec{N}$  к поверхности того же шара ( $-X^2 - Y^2 - Z^2 + 1 = 0$ ) направлена наружу.

Явное задание квадратичной поверхности применяют в методе обратной трассировки лучей. При прямой трассировке используют параметрическую форму:

$$X = X(u, v); \quad Y = Y(u, v); \quad Z = Z(u, v).$$

Например, эллипсоид в параметрическом виде:

$$\begin{cases} X = a \sin u \cos v; \\ Y = b \sin u \sin v; \\ Z = c \cos u. \end{cases}$$

Для переноса квадратичной поверхности используется преобразование:

$$\begin{bmatrix} X' & Y' & Z' & 1 \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \cdot F,$$

где  $(X', Y', Z')$  – точка в другой системе координат,  $F$  – матрица размером  $[4 \times 4]$ .

Новая матрица  $P'$ :

$$P' = F^{-1} P (F^{-1})^T.$$

Для перехода в другую систему координат параметрически заданной поверхности:

$$\begin{bmatrix} X'(u,v) & Y'(u,v) & Z'(u,v) & 1 \end{bmatrix} = \begin{bmatrix} X(u,v) & Y(u,v) & Z(u,v) & 1 \end{bmatrix} \cdot F.$$

Пример: преобразовать шар, описываемый формулой  $f(X, Y, Z) = X^2 + Y^2 + Z^2 - 1 = 0$ , в системе координат  $(X', Y', Z')$  путем сдвига на 5 единиц по всем осям.

Описание шара в матричном виде:

$$\begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \cdot P \cdot \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T = 0.$$

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \quad F = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -5 & -5 & -5 & -1 \end{bmatrix}.$$

Координаты новой точки:

$$P' = F^{-1} P (F^{-1})^T.$$

$$\begin{aligned} f(X', Y', Z') &= \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \cdot P \cdot \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T = \\ &= (x+5)^2 + (y+5)^2 + (z+5)^2 - 1 = 0. \end{aligned}$$

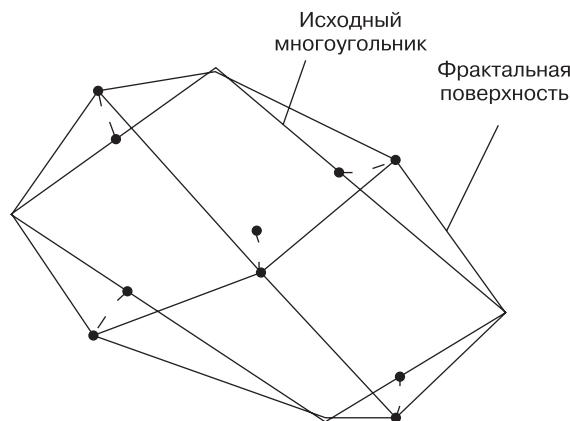
## Фрактальные поверхности

Ранее были рассмотрены различные методы получения моделей объектов и работы с ними. В качестве объектов обычно выступали предметы, сделанные человеком. Они были представлены или аппроксимированы плоскостями и поэтому были достаточно просты для моделирования. Что же делать с объектами живой природы? Например, как представить дерево с множеством листьев разной формы, размера и цвета? Модель становится очень сложной, потребуется много времени для ее обработки, что недопустимо для работы в реальном времени. С другой стороны, при моделировании дерева не очень важны особенности расположения листвы, так как его модель чаще всего используется не для проведения точных расчетов, а для формирования естественной сцены, например в компьютерной игре для фона. Некоторые же объекты вообще не могут быть описаны на основе традиционных приемов, базирующихся на использовании непрерывных функций. Но можно заметить, что большинство природных сцен статически родственны. Например, в результате проведения анализа нерегулярных изображений береговой линии, полученных при съемке с высот 10 и 3 км и при наблюдении с по-

верхности земли, было установлено, что при любом уровне разрешения береговая линия может быть смоделирована и нарисована посредством объединения участков небольших прямолинейных сегментов. Причем при переходе на каждый следующий, более высокий уровень разрешения, который был аппроксимирован первым прямолинейным сегментом, этот сегмент вероятностным способом разбивается на последовательность линейных сегментов и т. д. На основании этого свойства — постоянства статистического закона порождения деталей природных образований при переходе от низких уровней разрешения к более высоким — построен метод использования фрактальных поверхностей.

В переводе с английского слово *fractal* (фрактальный) обозначает «состоящий из частей». Такими поверхностями называется класс нерегулярных геометрических форм, задаваемых вероятностным образом на основе исходного описания низкого разрешения. Случайная закономерность, по которой исходная линия или поверхность дробится на несколько более мелких, подбирается опытным путем по критерию визуального согласования синтезированного изображения с реальной сценой.

Часто фрактальные поверхности используются для моделирования горного ландшафта. Сначала горный массив описывают очень приближенно полигональной сеткой из четырехугольников. Каждый четырехугольник с помощью случайной функции разбивают на четыре фигуры меньших размеров, причем эти фигуры вероятностным образом сдвигают относительно плоскости исходного четырехугольника, сохраняя для каждой фигуры по одной общей вершине с исходным четырехугольником. Каждую фигуру вновь делят, и так до достижения желаемого уровня изрезанности поверхности. Далее удаляются скрытые линии и производится закраска четырехугольников (рис. 9.1).



**Рис. 9.1.** Разбиение полигонального многоугольника при моделировании объема

Изображения, созданные на основе фрактальных поверхностей, только статистически идентичны реальным объектам и не обладают идеальной точностью (рис. 9.2, 9.3).

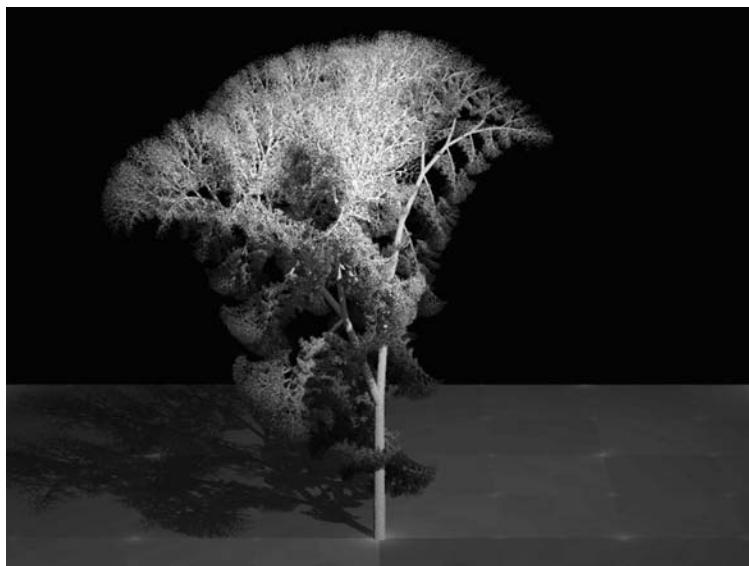


Рис. 9.2. Фрактальное изображение дерева



Рис. 9.3. Фрактальный пейзаж

### **9.3. Моделирование деформации трехмерных полигональных поверхностей в режиме реального времени**

Деформация (или морфинг) трехмерных объектов — это область компьютерной графики, в которой решаются задачи преобразования геометрических характеристик объектов. В качестве примеров задач из этой области можно назвать:

- «превращение» трехмерных объектов друг в друга;
- анимацию — построение промежуточных модификаций объекта (морфопоследовательности) по двум крайним (каждая модификация может отличаться как пространственным положением, так и формой);
- произвольную деформацию объекта с целью получения новых геометрических форм;
- моделирование неупругого взаимодействия физических тел и т. д.

Конкретный алгоритм, по которому осуществляется деформация, зависит от способа задания объекта.

К настоящему времени разработано большое количество методов трехмерной деформации, которые различаются по способу задания деформируемых объектов, степени физической достоверности моделируемого изменения формы, производительности или скорости работы. Как правило, деформация тел на основе физических законов требует больших вычислительных затрат, чем свободная деформация, то есть рассчитываемая без учета физических свойств объекта и влияния других тел.

Рассмотрим некоторые из методов трехмерной деформации.

#### **Метод деформации на основе использования неявного задания поверхности объекта**

Данный метод деформации использует так называемое неявное задание поверхности объекта. В основе метода лежит моделирование физического тела как набора связанных частиц и ассоциированной с ними неявной поверхности. На частицы могут действовать силы притяжения и отталкивания, а также силы жидкого трения.

Каждая частица является источником поля, напряженность которого убывает по мере удаления от частицы и стремится к нулю. Поверхность тела задается множеством точек  $P_i$ .

Этот метод позволяет моделировать изменение объема при столкновении, сложнейшие эффекты взаимодействия аморфных тел с поверхностями (протекание сквозь отверстие, растекание по поверхности) и между собой. Однако вычислительные затраты при применении указанного метода чрезвычайно высоки.

#### **Метод деформации плоских протяженных объектов**

При моделировании деформации плоских протяженных объектов (одежды, поверхности человеческого лица) часто применяется классический способ представления трехмерной поверхности в виде связанных треугольных граней. Для расчета используются такие физические характеристики тела, как модуль Юнга, коэффициент Пуассона, плотность и толщина материала. Деформация происходит под

действием внешних сил и внутренних напряжений в материале. В данном методе элементарный треугольник рассматривается как жесткий элемент, на который действуют силы кручения и поступательные силы. На их основе вычисляются векторы сил, действующих на все вершины объекта, затем уравнения движения каждой из них интегрируются и вычисляются их новые координаты. При этом деформация под воздействием силы  $F$ , меньшей, чем некоторый порог  $F_0$ , считается упругой и подчиняющейся закону Гука:

$$F(t) = k(x(l)) - x(0)).$$

Если  $F > F_0$ , то деформация считается неупругой.

Указанный метод весьма трудоемкий, однако вместе с алгоритмом отслеживания столкновения дает впечатляющие по своей реалистичности результаты при анимации одежды на движущемся человеке, флага на ветру, ряби на поверхности воды.

### **Метод деформации тела, заданного полигональной сеткой**

Недостатком двух предыдущих методов является необходимость проводить сложные расчеты, что не позволяет использовать их в реальном времени. И хотя моделирование деформации происходит достоверно, не всегда в этом есть необходимость. В случае использования деформации трехмерных объектов в приложениях, не требующих точности, можно применять алгоритмы, дающие лишь приблизительные результаты, но визуально воспринимаемые как точные. Это позволит сократить время на расчеты. При разработке алгоритма деформации тела, заданного полигональной сеткой, исходят из того, что модель поверхности и алгоритм должны обеспечивать легко предсказуемую деформацию как реакцию тела на конкретное воздействие извне. Скорость работы алгоритма должна позволять его использование в реальном или близком к реальному времени.

Поверхность рассматривается как набор плоских граней, а грань — как набор вершин, соединенных ребрами, причем между любыми двумя вершинами должен существовать путь по ребрам, то есть поверхность должна быть связанной. С каждой вершиной ассоциируется ее масса, а с каждым ребром — коэффициент жесткости. Шаги алгоритма распространения деформации следующие.

1. Выделенная вершина тела, называемая *вершиной поколения 0*, смещается в трехмерном пространстве на некоторый вектор. Эта вершина помечается как обработанная.
2. Для вершины поколения 0 находятся все смежные по какому-либо из ребер вершины — это вершины поколения 1. Для каждой из них вычисляется вектор смещения, зависящий только от вектора смещения вершины поколения 0. Эти вершины также помечаются как обработанные.
3. Далее для каждой из вершин текущего поколения находятся все ее потомки — смежные и еще не помеченные вершины. Для текущего потомка в его список родителей помещается текущая вершина-родитель. Все потомки помечаются и становятся вершинами следующего поколения, смещение каждой из которых вычисляется на основе смещений вершин из списка родителей.

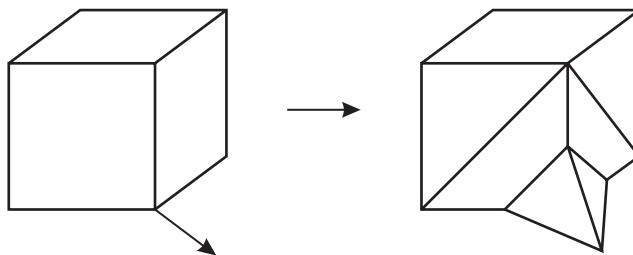
Распространение деформации продолжается до тех пор, пока не останется непомеченных вершин.

Данный алгоритм может быть использован для решения следующих задач:

- произвольная деформация объекта с целью получения новых геометрических форм;
- моделирование неупругого взаимодействия физических тел (трехмерные игры, компьютерная мультиплексия, компьютерное искусство и т. д.).

Описанный алгоритм можно также рассматривать как базовый, на основе которого легко далее строить более сложные модели. Кроме того, он может быть использован для решения более широкого класса задач.

1. Введение порога растяжения/сжатия отдельного ребра. Если данное отношение больше/меньше определенного порогового значения, то это ребро можно разбить на несколько более мелких и считать новую вершину родителем для старого потомка. Для реализации такого дополнения, конечно, потребуется генерация новых граней и ребер, смежных с вновь созданной вершиной (рис. 9.4).



**Рис. 9.4.** Эффект увеличения количества граней

2. Закрепление некоторых выбранных вершин, то есть такие вершины не будут менять свои координаты при смещениях остальных свободных вершин. В данном случае в алгоритме этого можно достичь путем присваивания вершинам такого рода бесконечно большой массы.
3. Деформирующее воздействие сразу на несколько вершин. Меняются лишь начальные условия: они стартуют не с нулевого, а с первого поколения, как будто бы у необрабатываемой вершины нулевого поколения есть  $N$  потомков, где  $N$  – количество первоначально смещаемых вершин.

## 9.4. Триангуляция поверхностей

Генерация объемных изображений является сложной вычислительной задачей, в связи с чем на практике выполняют ее декомпозицию. Сложные изображения разбивают на составные части. Процесс разбиения поверхности объектов на полигоны называется *тесселяцией*. Но наиболее часто производится разбиение изображений на треугольники. Это объясняется следующими причинами:

- треугольник является простейшим полигоном, вершины которого однозначно задают грань;

- любую область можно гарантированно разбить на треугольники;
- вычислительная сложность алгоритмов разбиения на треугольники существенно меньше, чем при использовании других полигонов;
- реализация процедур визуализации более проста для области, ограниченной треугольником;
- для треугольника легко определить три его ближайших соседа, имеющих с ним общие грани.



**Рис. 9.5.** Триангулированная поверхность тора

Процесс разбиения полигональной области со сложной конфигурацией на набор треугольников называется *триангуляцией* (рис. 9.5).

Любая поверхность может быть аппроксимирована с необходимой точностью сеткой треугольников. Точность аппроксимации определяется количеством треугольников и способом их выбора. Для качественной визуализации объекта, находящегося вблизи точки наблюдения, требуется учесть во много раз больше треугольников, чем в ситуации, когда тот же объект расположен в отдалении.

В компьютерной графике задача триангуляции рассматривается в двух

направлениях: триангуляция полигональных областей и триангуляция набора точек. Триангуляцию точек на плоскости используют при описании поверхности набором точек и интенсивностями их цветов.

Поточечное описание поверхностей применяют в тех случаях, когда поверхность очень сложная и не обладает гладкостью, а детальное представление многочисленных геометрических особенностей важно для практики.

Простейшее решение задачи триангуляции состоит в разделении полигона вдоль некоторой хорды на две части и в дальнейшем рекурсивном разбиении их до тех пор, пока подлежащий триангуляции полигон не станет треугольником.

Данный способ применим лишь для триангуляции выпуклых полигонов.

Триангуляция невыпуклых полигонов более сложна, поэтому лучше предварительно разбить невыпуклые многоугольники на выпуклые для упрощения их последующей обработки. Сначала желательно перенести и повернуть многоугольник так, чтобы одна из его вершин совпала с началом координат, а исходящая из нее сторона — с осью  $OX$ . При расположении каких-либо сторон ниже оси происходит их отсечение. Алгоритм рекурсивно повторяют для полученных полигонов, пока они не станут выпуклыми (рис. 9.6).

В рассмотренном примере выбирается крайняя левая вершина, и между двумя ее смежными сторонами проводится диагональ. При этом могут получиться

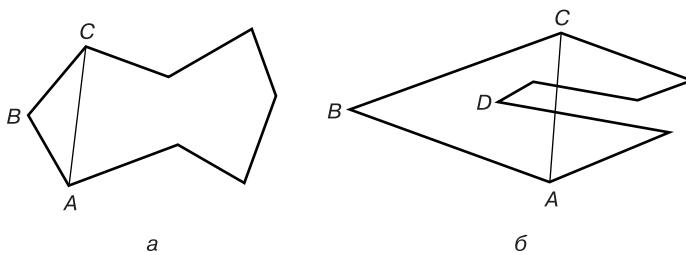


Рис. 9.6. Разбиение невыпуклого многоугольника

следующие два случая: рис. 9.6, а — диагональ  $AC$  является хордой; рис. 9.6, б — диагональ  $AC$  не является хордой, так как внутри треугольника  $ABC$  попадает вершина  $D$  полигона (в общем случае их может быть несколько). Из всех вершин внутри треугольника  $ABC$  вершина  $D$  наиболее удалена от стороны  $AC$ . Эта вершина называется *вторгающейся*. Если такой вершины нет, полученный треугольник заносится в сетку треугольников и алгоритм рекурсивно обрабатывает оставшийся полигон до тех пор, пока он не выродится в треугольник.

При обнаружении вторгающейся вершины проводится диагональ из текущей вершины до вторгающейся. Полученные полигоны рекурсивно обрабатываются до получения треугольников.

Иногда для разбиения многоугольника на треугольники находят внутреннюю точку области, ограниченной полигоном, и соединяют с ней все вершины (рис. 9.7).

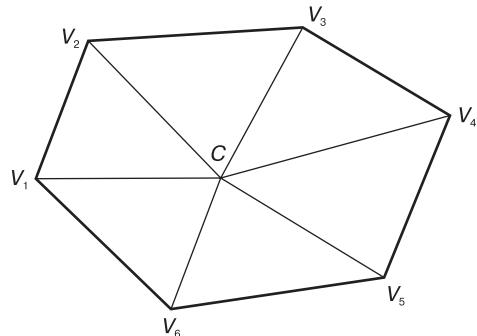


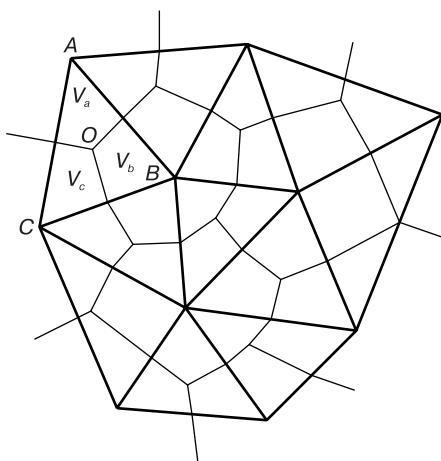
Рис. 9.7. Разбиение средней точкой

### Области Вороного и триангуляция Делоне

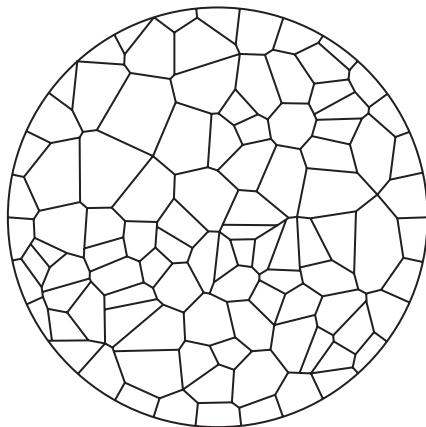
Понятие *области Вороного* было введено Г. Ф. Вороным в 1908 году. Суть этого разбиения состоит в разграничении плоскости на области, состоящие из точек, объединенных общей ближайшей опорной точкой.

Назовем  $V_i$  область Вороного точки  $i$ . Если точек только две, то очевидно, что срединный перпендикуляр к соединяющему их отрезку делит плоскость на две соответствующие области Вороного. В общем случае область Вороного точки  $i$  получается как пересечение всех полуплоскостей, содержащих эту точку и ограниченных срединными перпендикулярами к отрезкам, соединяющим  $i$  с остальными опорными точками. Поэтому области  $V_i$  будут ограничены выпуклой ломаной (рис. 9.8).

У полученной решетки ребра — это отрезки срединных перпендикуляров. Поэтому, если некоторая вершина  $O$  области  $V_b$  получена пересечением срединных перпендикуляров к отрезкам  $AB$  и  $BC$ , то она равноудалена от точки  $A$  и точки  $B$ , от точки  $B$  и точки  $C$ , то есть точка  $O$  — это центр описанной окружности треугольника  $ABC$ ,



**Рис. 9.8.** Разбиение многоугольника на области Вороного



**Рис. 9.9.** Решетка из областей Вороного

и срединный перпендикуляр к отрезку  $AC$  тоже проходит через точку  $O$ . Так как точка  $O$  принадлежит области  $V_b$ , то не существует опорных точек, расположенных к точке  $O$  ближе, чем точке  $B$ , но отрезки  $OA$ ,  $OB$  и  $OC$  равны, поэтому точка  $O$  принадлежит области  $V_a$ , и точка  $O$  принадлежит области  $V_c$ .

Кроме того, области Вороного образуют решетку, в каждой вершине которой сходятся три ребра и три области (рис. 9.9).

Если соединить между собой точки, области Вороного которых граничат друг с другом, получится триангуляция, причем вершины областей Вороного будут центрами описанных окружностей для полученных треугольников. Такая конструкция называется *триангуляцией Делоне*.

### Алгоритм Рапперта

Этот алгоритм удобен для разбиения на треугольники для получения конечно-элементных сеток. Он учитывает следующие требования:

- треугольники не должны быть сильно вытянутыми, так как наличие таких треугольников отрицательно сказывается на точности результатов расчета;
- должны быть учтены характер работы конструкции и ее геометрия, то есть в местах концентрации напряжений сетка должна сгущаться.

В первичном разбиении входными данными являются массивы точек (координаты, нагрузка, закрепление) и граней полигонов (сегментов), соединяющих эти точки. Выходные данные — это массив вершин без изменений, массив граней треугольников и массив собственно треугольников, заполняющих весь объем конструкции.

Алгоритм работает следующим образом.

1. Выбирается начальный сегмент (любой) на внешнем полигоне исходной геометрии.

2. Из всех вершин выбирается такая, в которой лучи, соединяющие эту вершину и концы текущего сегмента, образуют максимальный угол.
3. Добавляется полученный таким образом треугольник, и текущим сегментом становится одна из созданных сторон этого треугольника.
4. Если существует подходящий текущий сегмент, то осуществляется переход на шаг 2.

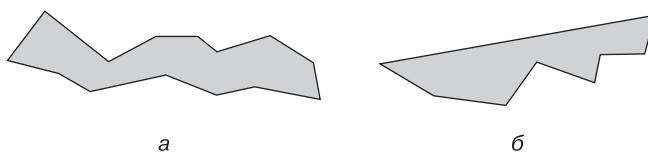
Первым действием является нахождение первого «правильного» треугольника. После этого появляются две первые грани. Далее треугольники образуются путем правильного добавления к грани еще двух граней, соединяющих концы этой грани и определенную точку, принадлежащую сегменту и т. д.

При выборе третьей вершины для формирования треугольника возникает множество вопросов. Во-первых, необходим критерий, позволяющий легко избежать пересечений вновь создаваемых граней с уже существующими. Во-вторых, следует ограничить треугольники от распространения за пределы полигональной области. В-третьих, нужно правильно генерировать первый треугольник. В-четвертых, необходимо учесть то обстоятельство, что в конструкции присутствуют как отверстия, так и заполненные области, описываемые одинаково.

Критерием для оптимального выбора третьей точки треугольника является угол, под которым видна грань из этой точки, то есть угол, образуемый двумя лучами, выходящими из выбранной точки и проходящими через концы грани. Для каждой грани выбирается такая точка, для которой данный угол максимальен. Это позволяет сэкономить время на проверке вновь создаваемых граней на пересечение с уже существующими.

### Триангуляция монотонных полигонов

Цепочка вершин является *монотонной*, если любая вертикальная линия пересекает ее не более одного раза. Начиная с самого левого конца все вершины монотонной цепочки расположены в порядке увеличения их координаты  $x$ . Полигон называется *монотонным*, если его граница составлена из двух монотонных цепочек вершин полигона: верхней и нижней. Каждая цепочка заканчивается в самой левой и в самой правой вершинах полигона и содержит либо ни одной, либо несколько вершин между ними. На рис. 9.10, *a* и *б* показана пара примеров. Заметим, что пересечение вертикальной прямой и монотонного полигона состоит либо из отрезка прямой вертикальной линии, либо из точки.



**Рис. 9.10.** Монотонные полигоны

Алгоритм декомпозиции полигона на монотонные части основывается на теореме о монотонности полигона и понятии излома. Суть теоремы состоит в том, что любой полигон, не содержащий излома, является монотонным. При этом вогнутая

вершина (вершина, внутренний угол которой превышает  $180^\circ$ ) является изломом, если обе ее соседние вершины лежат либо слева, либо справа от нее. Алгоритм выполняется путем декомпозиции полигона  $P$  на подполигоны без изломов. В первой из двух фаз по мере продвижения сканирующей линии по прямоугольнику слева направо алгоритм удаляет изломы, направленные влево, формируя набор подполигонов  $P_1, P_2, \dots, P_m$ , ни один из которых не содержит изломов, направленных влево. Во время второй фазы алгоритм удаляет изломы, направленные вправо, продвигая сканирующую линию справа налево по подполигонам  $P_i$  по очереди. Полученная коллекция полигонов и будет декомпозицией исходного полигона  $P$  на подполигоны, в которых нет ни право-, ни левонаправленных изломов и, следовательно, совсем никаких изломов.

В 1991 году Бернард Чезелле разработал оптимальный алгоритм.

Пусть имеется монотонный полигон  $P$  с вершинами  $v_1, v_2, \dots, v_n$  в порядке увеличения координаты  $x$ , поскольку алгоритм будет обрабатывать эти вершины именно в таком порядке. Алгоритм формирует последовательность монотонных полигонов  $P = P_1, P_2, \dots, P_n = 0$ . Полигон  $P_i$ , как результат обработки вершины  $v_i$ , получается путем отсечения нуля или нескольких треугольников от предыдущего полигона  $P_{i-1}$ . Алгоритм заканчивает свою работу после выхода с  $P_n$  пустым полигоном, а коллекция треугольников, накопленная в процессе обработки, представляет собой триангуляцию исходного полигона  $P$ .

### Уровень детализации (LOD)

*Уровень детализации* (Level Of Detail, LOD) – это метод снижения сложности рендеринга кадра, уменьшения общего количества полигонов, текстур и иных ресурсов в сцене, общее снижение ее сложности в зависимости от расстояния до объекта. Простой пример: основная модель персонажа состоит из тысячи полигонов. В тех случаях, когда в обрабатываемой сцене он расположен близко к камере, важно, чтобы использовались все полигоны, но на очень большом расстоянии от наблюдателя в итоговом изображении персонаж будет занимать лишь несколько пикселов, и смысла в обработке всей тысячи полигонов нет никакого.

Метод LOD обычно применяется при моделировании и визуализации трехмерных сцен с использованием нескольких уровней сложности для объектов пропорционально расстоянию от них до наблюдателя. Метод часто используется для снижения количества полигонов в сцене и для улучшения производительности. Изменение сложности, в частности, количества треугольников в модели может происходить автоматически на основе одной 3D-модели максимальной сложности, а может – на основе нескольких заранее подготовленных моделей с разным уровнем детализации (рис. 9.11).

Метод особенно эффективен, если количество объектов в сцене велико и они расположены на разных расстояниях от камеры.

Кроме расстояния от наблюдателя до объекта, для LOD могут иметь значение и другие факторы: общее количество объектов на экране (если в кадре один-два персонажа, используются сложные модели, а если 10–20, то они переключают-

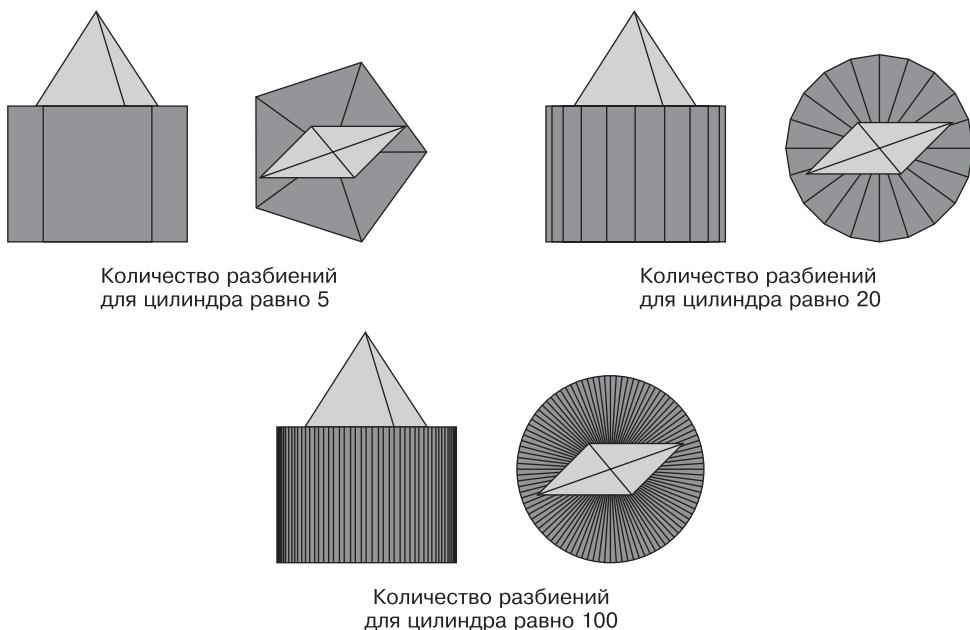


Рис. 9.11. Варианты аппроксимации

ся на более простые) или количество кадров в секунду. Другими возможными факторами, влияющими на уровень детализации, являются скорость перемещения объекта (машины в движении вы рассмотреть вряд ли успеете, а вот улитку — запросто) и важность объекта (рис. 9.12). Главное — не переусердствовать, частые и заметные изменения уровня детализации мешают.

В данном примере видно, что ближайшая к наблюдателю модель автомобиля максимально детализована, а все дальние имеют видимые упрощения. Самая дальняя машина вообще изображена лишь небольшим пятном. Это и есть алгоритм Level Of Detail в действии.



Рис. 9.12. Пример изображения автомобилей

# **Глава 10**

## **Получение реалистичных изображений**

---

Получение реалистичных изображений — одна из самых важных задач компьютерной графики на сегодняшний день. Изобразить апельсин не только в виде оранжевого круга, но и передать его фактуру в виде пористой кожуры, показать блик от источника света и отбрасываемую на стол тень — все это становится возможным в связи с развитием технических средств. И в то же время требуются очень экономичные алгоритмы, позволяющие за короткий промежуток времени просчитать сложные сцены.

### **10.1. Методы создания реалистичных изображений**

Главная трудность на пути получения изображения объекта состоит в том, что все устройства вывода являются двумерными. Трехмерные объекты приходится проецировать на плоскость, что приводит к существенным потерям информации, а иногда и к неопределенности изображения. Существуют разные методы, которые используются для восстановления информации, теряющейся при проецировании (см. пример с кубом на рис. 6.3).

С середины 1960-х годов ведутся поиски способов и средств создания реалистичных изображений, чтобы наблюдатель принимал изображение за реальный объект, а не за синтетический, существующий только в памяти компьютера. Это особенно важно при моделировании, проектировании и организации досуга.

Конструкторам автомобилей, самолетов, машин и т. д. хотелось бы заранее знать, как будет выглядеть их проект. Формирование реалистичных изображений, генерированных на компьютере, во многих случаях представляет собой более легкий, дешевый и эффективный способ просмотра предварительных результатов, чем изготовление моделей и опытных образцов. Кроме того, оно позволяет рассмотреть большее количество вариантов проекта.

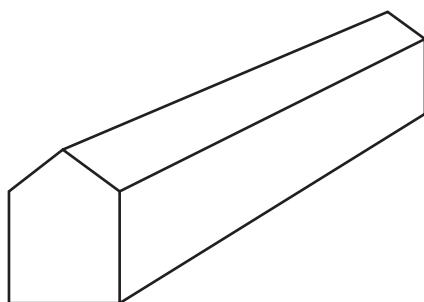
Основной трудностью при создании реалистичных изображений является сложность реальных визуальных образов. Наше окружение очень разнообразно. В нем существуют многочисленные фактуры поверхности, плавные переходы цветов, тени, отражения и мелкие неровности (царапины на полу, чешуйки краски, выступы на стенах). Сочетаясь в нашем сознании, они образуют реальный визуальный опыт.

Рассмотрим методы, которые используются для восстановления информации, чтобы присущие человеку механизмы восприятия глубины могли соответствующим образом ликвидировать неясности.

#### **Перспективные проекции**

Размеры объектов обратно пропорциональны их расстоянию от наблюдателя. Чем дальше расположен объект, тем меньше он в изображении. Но перспектива

подходит не всегда. Особенно эффективно ее использование для объектов с большим количеством параллельных линий, так как в изображении они будут соединяться в точке схода (рис. 10.1). Фактически это схождение линий хорошо передает глубину, потому что согласуется с опытом человека. Для сложных объектов (молекулярные структуры), где отсутствуют параллельные линии, перспективные изображения мало пригодны.



**Рис. 10.1.** Перспективное изображение

### Передача глубины яркостью

Глубину объекта (расстояние до него) можно представить путем изменения уровня яркости. Объекты, находящиеся ближе к наблюдателю, выводятся с увеличенной яркостью (аналогия с реальным зрением). Чем дальше расположен объект, тем менее ярким он будет в изображении.

Для реализации данного принципа необходима информация о глубине (координате  $z$ ). Но человеческий глаз различает яркости гораздо хуже, чем положение по глубине, поэтому передать небольшие различия в расстояниях с помощью яркости сложно.

### Отсечение по глубине

Выводимый объект пересекается плоскостью, отсекающей его удаленную часть. Чем дальше расположен объект, тем позже он будет в изображении. Удобно динамически изменять положение отсекающей плоскости от ближней части объекта до дальней.

### Динамические проекции

Если серию проекций объекта выводить быстро с разных точек зрения, расположенных недалеко друг от друга, то создается впечатление вращения объекта.

### Удаление скрытых линий и поверхностей

Убирая с изображения невидимые линии, легче понять геометрию сложного объекта и его расположение. Ранее эти алгоритмы требовали больших затрат времени и нечасто использовались в компьютерной графике. Сейчас же ситуация заметно изменилась и удаление невидимых линий стало естественным процессом.

### Стереоскопия

Если посмотреть на объект поочередно одним и другим глазом, то два вида будут при этом различаться (бинокулярный эффект). Наш мозг сливает два раздельных образа в один трехмерный, получая на базе разницы изображений информацию об объеме. Два изображения можно объединить в один трехмерный образ, если разглядывать эту пару так, чтобы каждый глаз видел только одно изображение. Разработчик интерактивной системы должен предъявить каждому глазу вид, который отличается от другого.

Для достижения этого эффекта используются специальные технические средства, например шлем с двумя ЭЛТ. При движении головы может изменяться и изображение (датчики, фиксирующие движение).

## 10.2. Перспективные изображения

В IV веке до н. э. в Древней Греции существовала Саксонская школа рисунка, на дверях которой было написано: «Сюда не допускаются люди, не знающие геометрии». И это не случайно: знание геометрии особенно необходимо для отображения трехмерного мира на плоскости.

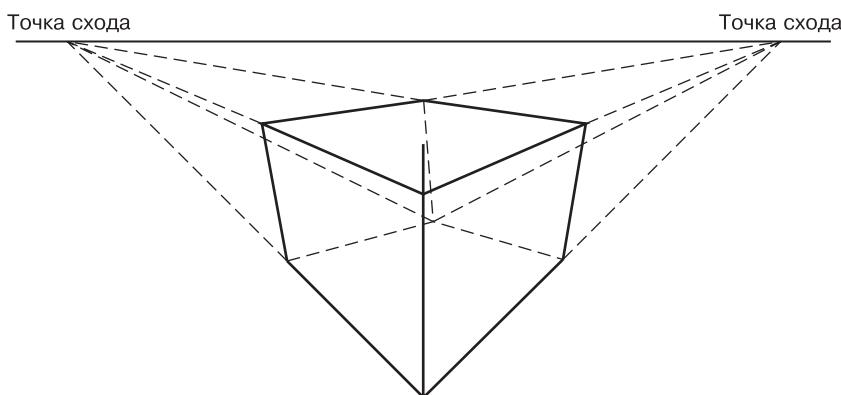
Долгий путь, измеряемый многими тысячелетиями, прошло развитие рисунка, прежде чем были открыты основные законы реалистичного отображения окружающей нас действительности — *законы перспективы и светотени*. До открытия этих законов живопись носила плоскостной характер.

Так, живописцы Древнего Египта на протяжении тысячелетий пользовались одними и теми же приемами. Пространство они условно расчленяли на пояса: нижний пояс относился к ближнему плану, самый верхний — к дальнему.

Первые правила перспективы были сформулированы древнегреческим математиком Эвклидом (III век до н. э.). В совершенстве овладел искусством перспективы и светотени Леонардо да Винчи. В своем «Трактате о живописи» он писал: «Перспектива есть не что иное, как вид из окна, на совершенно прозрачном стекле которого изображены предметы, находящиеся за окном».

Изобретение фотографии предложило новый способ формирования перспективных изображений. Существует строгая аналогия между камерой и человеческим глазом. Фотокамера является простейшей имитацией человеческого глаза.

Основное свойство перспективного изображения — более удаленные предметы изображаются в меньших масштабах. Параллельные прямые в общем случае в изображении не параллельны (рис. 10.2).



**Рис. 10.2.** Перспективная проекция куба

Проекции параллельных ребер объекта на такой проекции не являются параллельными.

# Глава 11

## Проектирование

Для отображения трехмерной модели объекта на плоском экране монитора требуется предварительное проецирование на плоскость. Существуют разные виды проекций. Некоторые широко использовались ранее, когда чертежи выполнялись вручную и, следовательно, выполнять сложные расчеты проецирования было невозможно. Некоторые стали популярны лишь с появлением компьютерной графики.

### 11.1. Основные виды проекций

В общем случае проекции преобразуют точки в системе координат размерностью  $n$  в точки в системе координат размерностью  $m$ , где  $m < n$ . На практике применяется преобразование трехмерного пространства в двумерное.

Проектирование трехмерного объекта осуществляется с помощью прямых проецирующих лучей, которые называются *проекторами*. Они выходят из центра проекции, проходят через каждую точку объекта и, пересекая картинную плоскость, образуют проекцию. Так как проекция отрезка сама является отрезком, то достаточно спроектировать лишь конечные точки. Проекции делятся на центральные и параллельные (рис. 11.1).

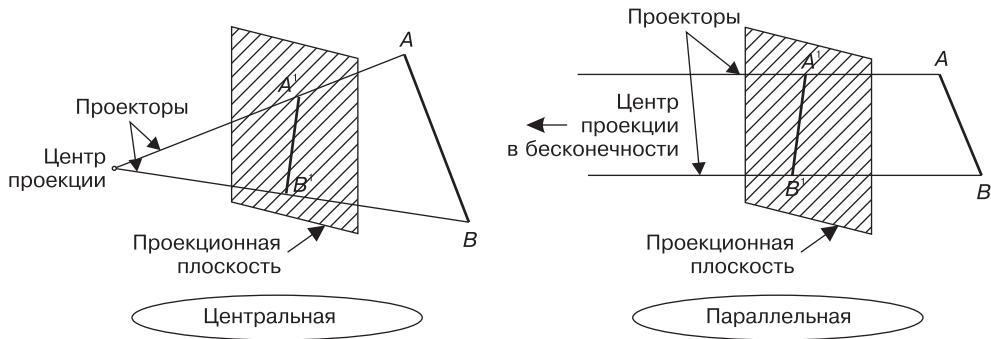
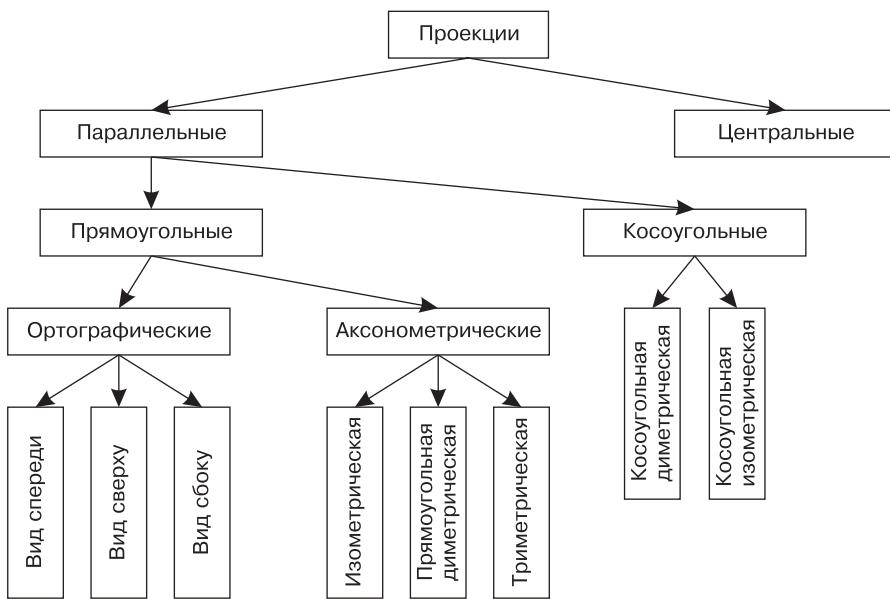


Рис. 11.1. Виды проецирования

Определенные таким образом проекции являются плоскими геометрическими проекциями (проецирование на плоскость прямыми линиями). Основные виды проекций приведены на рис. 11.2.

Если расстояние между центром проекции и ее плоскостью конечно, то проекция называется *центральной*, если же бесконечно, проекция называется *параллельной*.



**Рис. 11.2.** Основные виды проекций<sup>1</sup>

При описании центральной проекции задается центр проекции, а при описании параллельной проекции — направление проецирования.

### Параллельные проекции

Параллельные проекции делятся на два типа в зависимости от соотношения между направлением проецирования ( $\vec{l}$ ) и нормалью к проецируемой плоскости ( $\vec{n}$ ).

В *прямоугольных* проекциях эти направления совпадают ( $\vec{n} = \vec{l}$ ), а в *косоугольных* — нет ( $\vec{n} \neq \vec{l}$ ). Наиболее широко используются *ортографические* проекции: вид спереди, сверху и сбоку, в которых картинная плоскость перпендикулярна главным координатным осям, совпадающим с направлением проецирования  $\vec{l}$  (рис. 11.3).

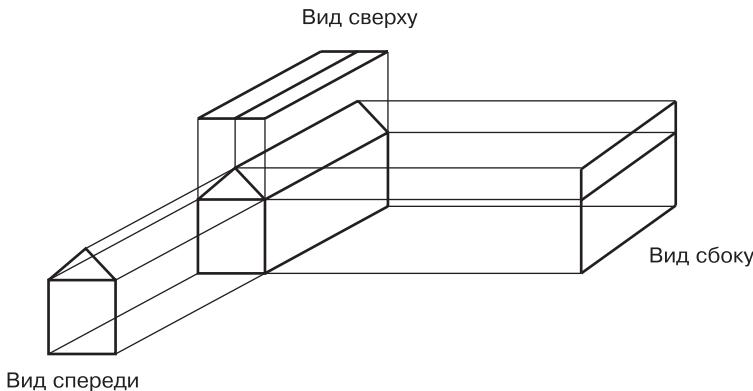
В *аксонометрических* проекциях используется проекционная плоскость, не перпендикулярная главным координатным осям, поэтому на них изображается сразу несколько сторон объекта. Сохраняется параллельность прямых, а углы изменяются.

Широко используется *изометрия*. В этом случае нормаль к проекционной плоскости составляет равные углы с каждой из главных координатных осей. Если нормаль к проекционной плоскости имеет координаты  $(a, b, c)$ , то  $|a| = |b| = |c|$ .

Существуют четыре различные изометрические проекции:  $(a, a, a)$ ;  $(-a, a, a)$ ;  $(a, -a, a)$ ;  $(a, a, -a)$ . Свойство изометрии: все три главные координатные оси одинаково укорачиваются.

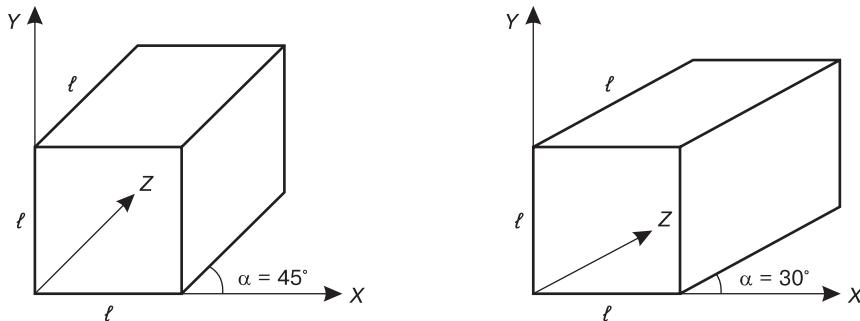
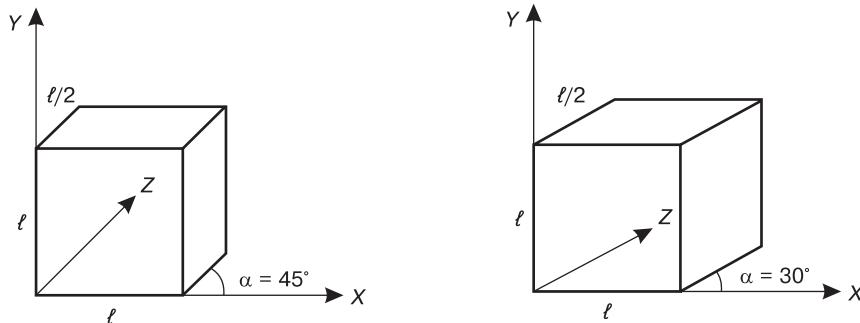
В *диметрии* нормаль к проекционной плоскости составляет равные углы с двумя координатными осями.

<sup>1</sup> Котов Ю. В. Как рисует машина. — М.: Наука, 1988.

**Рис. 11.3.** Ортографические проекции

В *трииметрии* нормаль к проекционной плоскости образует с координатными осями различные углы.

*Косоугольные* проекции ( $\vec{n} \neq \vec{l}$ ) сочетают в себе свойства ортографических проекций со свойствами аксонометрии. Проекционная плоскость перпендикулярна главной координатной оси, поэтому сторона объекта, параллельная этой плоскости, проецируется так, что углы и расстояние не искажаются.

**Рис. 11.4.** Косоугольная изометрия**Рис. 11.5.** Косоугольная диметрия

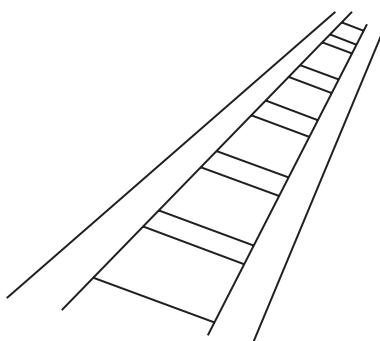
В *косоугольной изометрии* направление проецирования составляет с проекционной плоскостью угол  $45^\circ$ . В результате проекция отрезка, перпендикулярного проекционной плоскости, имеет ту же длину, что и сам отрезок, то есть укорачивания нет (рис. 11.4).

В *косоугольной диметрии* направление проецирования составляет с проекционной плоскостью угол  $\text{arctg}(2)$ . Отрезки, перпендикулярные проекционной плоскости, после проецирования составляют  $1/2$  их действительной длины (рис. 11.5).

Эта проекция более реалистична, так как укорачивание с  $k = 1/2$  больше согласуется с нашим визуальным опытом.

### Центральные проекции

Основное свойство центральных проекций — более удаленные предметы изображаются в меньших масштабах (рис. 11.6).



**Рис. 11.6.** Центральная проекция

Параллельные прямые в общем случае в изображении не параллельны.

Если совокупность прямых параллельна одной из главных координатных осей, то их точка схода называется *главной точкой схода*. Имеются три такие точки.

Если проекционная плоскость перпендикулярна оси  $Z$ , то лишь на этой оси будет лежать главная точка схода.

Центральные проекции классифицируются в зависимости от количества главных точек схода, которыми они обладают (рис. 11.7).

Двухточечная центральная проекция широко применяется в архитектурном, инженерном проектировании и в рекламных изображениях, в которых вертикальные прямые проецируются как параллельные.

Трехточечные центральные проекции почти совсем не используются, так как их трудно конструировать и они добавляют мало нового с точки зрения реалистичности.

Перспективное изображение зависит от положения глаза. Эффект перспективы обратно пропорционален расстоянию между глазом и объектом. Если глаз находится близко от объекта, то получается сильный эффект перспективы (хорошо видны точки схода, линии явно не параллельны). Если глаз расположен далеко, то параллельные линии объекта будут казаться параллельными и на картинке.

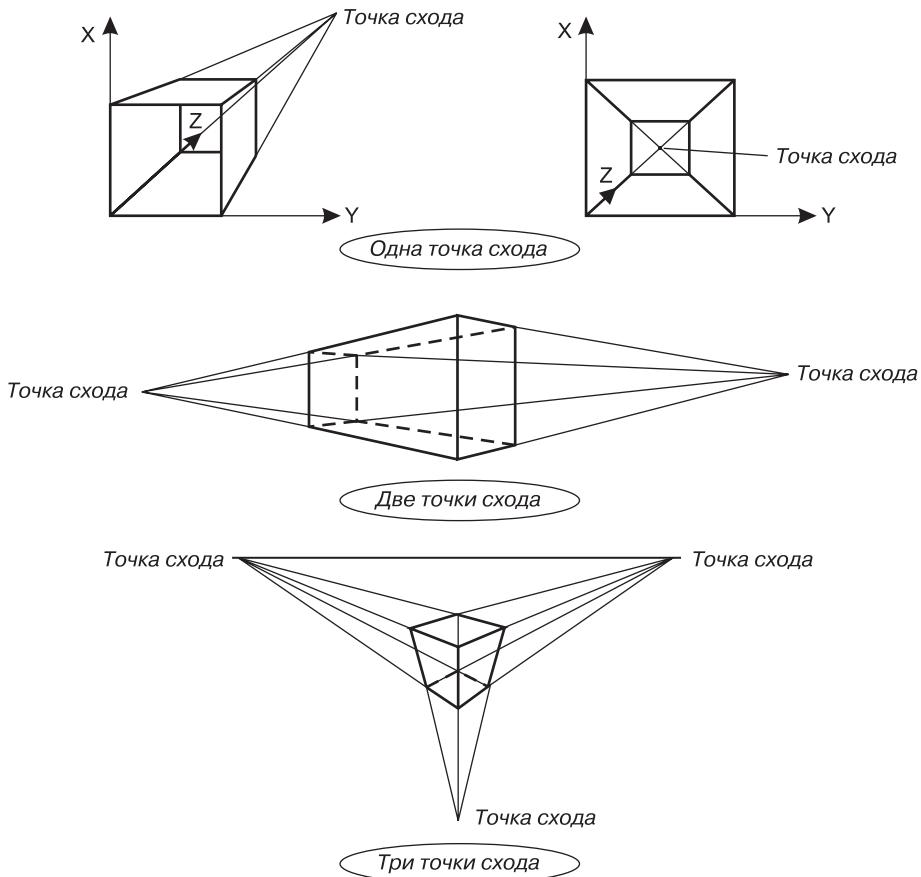


Рис. 11.7. Центральные проекции с разным количеством точек схода<sup>1</sup>

## 11.2. Математическое описание прямоугольных проекций

Ортографические проекции получаются достаточно просто. Если проекционная плоскость —  $Z = 0$ , направление проецирования совпадает с осью  $Z$ , то точка  $P$  имеет координаты:

$$X_p = X, \quad Y_p = Y, \quad Z_p = 0.$$

Матрица проецирования:

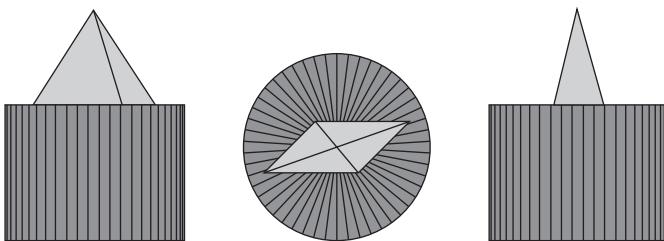
$$M_{\text{опр}_Z} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \text{фронтальная проекция.}$$

<sup>1</sup> Котов Ю. В. Как рисует машина. — М.: Наука, 1988.

Аналогично для двух других проекций (рис. 11.8):

$$M_{\text{опт}_X} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \text{профильная проекция};$$

$$M_{\text{опт}_Y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \text{горизонтальная проекция}.$$



**Рис. 11.8.** Ортографические проекции

**Аксонометрическая проекция** получается при повороте на угол  $\psi$  относительно оси  $Y$  и на угол  $\phi$  относительно оси  $X$  с последующим проецированием вдоль оси  $Z$  (рис. 11.9).

Матрица проецирования:

$$M_{\text{акс}} = \begin{bmatrix} \cos\psi & \sin\phi\sin\psi & 0 & 0 \\ 0 & \cos\phi & 0 & 0 \\ \sin\psi & -\sin\phi\cos\psi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\psi & 0 & -\sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \\ \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & \sin\phi & 0 \\ 0 & -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Покажем, как при этом преобразуются единичные орты координатных осей  $X, Y, Z$ :

$$(1 \ 0 \ 0 \ 1) \cdot M = (\cos\psi \ \sin\phi\sin\psi \ 0 \ 1);$$

$$(0 \ 1 \ 0 \ 1) \cdot M = (0 \ \cos\phi \ 0 \ 1);$$

$$(0 \ 0 \ 1 \ 1) \cdot M = (\sin\psi \ -\sin\phi\cos\psi \ 0 \ 1).$$

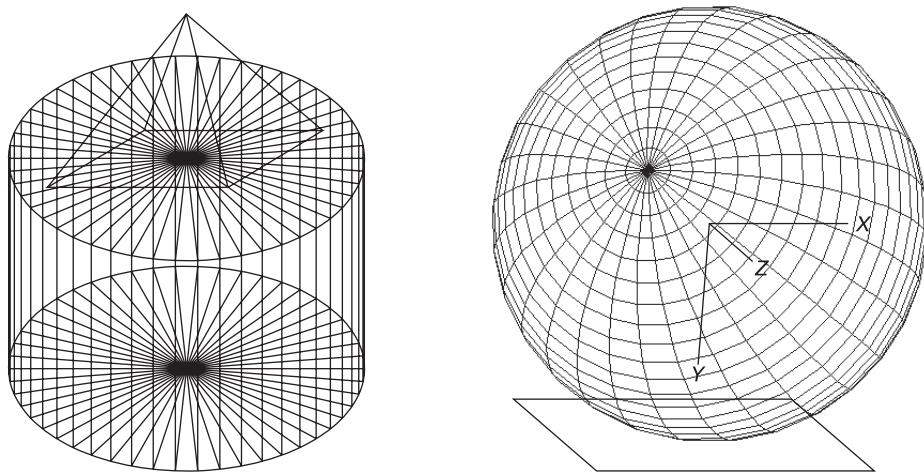


Рис. 11.9. Аксонометрические проекции

Для изометрии:

$$\begin{cases} \cos^2 \psi + \sin^2 \phi \sin^2 \psi = \cos^2 \phi \\ \sin^2 \psi + \sin^2 \phi \cos^2 \psi = \cos^2 \phi \end{cases} \Rightarrow \sin^2 \phi = \frac{1}{3}, \quad \sin^2 \psi = \frac{1}{2}.$$

Для диметрии:

$$\cos^2 \psi + \sin^2 \phi \sin^2 \psi = \cos^2 \phi \Rightarrow \sin^2 \psi = \tan^2 \phi.$$

Изометрическая проекция:

$$\begin{bmatrix} X_{a'} \\ Y_{a'} \\ Z_{a'} \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix} + m \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} X_a \\ Y_a \\ Z_a \end{bmatrix}.$$

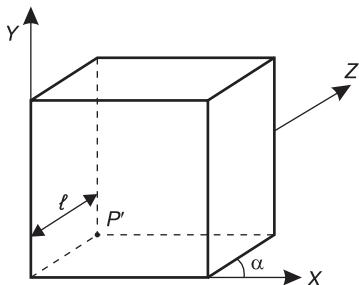
Диметрическая проекция:

$$\begin{bmatrix} X_{a'} \\ Y_{a'} \\ Z_{a'} \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix} + m \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\frac{\sqrt{2}}{4} \\ 0 & 1 & \frac{\sqrt{2}}{4} \\ \frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{3} \end{bmatrix} \begin{bmatrix} X_a \\ Y_a \\ Z_a \end{bmatrix},$$

где  $X_a, Y_a, Z_a$  — координаты точки объекта;  $X_{a'}, Y_{a'}, Z_{a'}$  — координаты точки изображения.

### 11.3. Математическое описание косоугольных проекций

При описании косоугольных проекций матрица проецирования может быть записана исходя из значений  $\alpha$  и  $\ell$ . Для единичного куба, спроецированного на плоскость  $XY$ , точка  $P$  принадлежит объекту, а точка  $P'$  – изображению точки  $P$  на проекции (рис. 11.10).



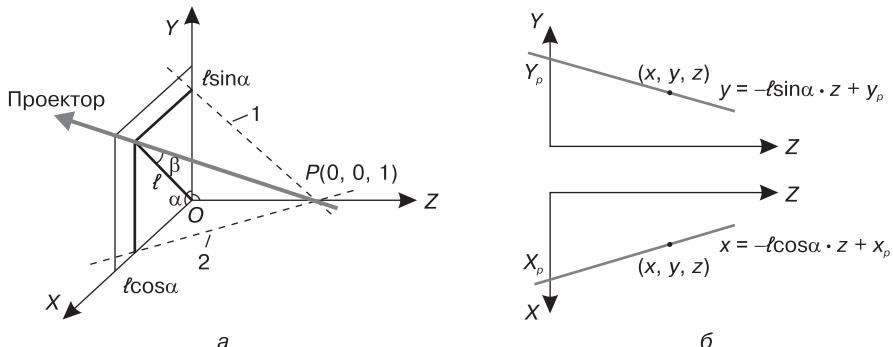
**Рис. 11.10.** Косоугольная проекция

Проекцией точки  $P$ , координаты которой  $(0, 0, 1)$ , является точка  $P'$  с координатами  $(\ell \cos \alpha, \ell \sin \alpha, 0)$ , принадлежащая плоскости  $XY$ . По определению это означает, что направление проецирования совпадает с отрезком  $\overline{PP'}$ . Данное направление есть:

$$\overrightarrow{P' - P} = (\ell \cos \alpha, \ell \sin \alpha, -1).$$

Направление проецирования составляет угол  $\beta$  с плоскостью  $XY$ .

Рассмотрим произвольную точку  $(x, y, z)$  и определим ее косоугольную проекцию  $(x_p, y_p)$  на плоскость  $XY$  (рис. 11.11).



**Рис. 11.11.** Получение косоугольной проекции<sup>1</sup>

Уравнение для координат  $x$  и  $y$  проектора как функций  $z$  имеет вид:

$$y = mz + b.$$

$$y = -\ell \sin \alpha \cdot z + \ell \sin \alpha \quad (\text{рис. 11.11, } a \text{ прямая 1}).$$

$$x = -\ell \cos \alpha \cdot z + \ell \cos \alpha \quad (\text{рис. 11.11, } a \text{ прямая 2}).$$

На рис. 11.11, *б* показаны изображения точки и проектора. Уравнения для координат  $x$  и  $y$  проектора:

$$\begin{cases} y = -\ell \sin \alpha \cdot z + y_p; \\ x = -\ell \cos \alpha \cdot z + x_p. \end{cases}$$

Найдем  $x_p, y_p$ :

<sup>1</sup> Комов Ю. В. Как рисует машина. – М.: Наука, 1988.

$$\begin{cases} x_p = x + z \cdot \ell \cos \alpha; \\ y_p = y + z \cdot \ell \sin \alpha. \end{cases}$$

Матрица, которая выполняет эти действия, а следовательно, описывает косоугольную проекцию:

$$M_{\text{кос}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \ell \cos \alpha & \ell \sin \alpha & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Применение матрицы  $M_{\text{кос}}$  приводит к сдвигу и последующему проецированию объекта. Плоскости с постоянной координатой  $z = z_1$  переносятся в направлении  $x$  на  $z_1 \ell \cos \alpha$ , в направлении  $y$  — на  $z_1 \ell \sin \alpha$  и затем проецируются на плоскость  $z = 0$ . Сдвиг сохраняет параллельность прямых, а также углы и расстояния в плоскостях, параллельных оси  $z$ .

Для изометрической косоугольной проекции  $\ell = 1$  (см. рис. 11.11):

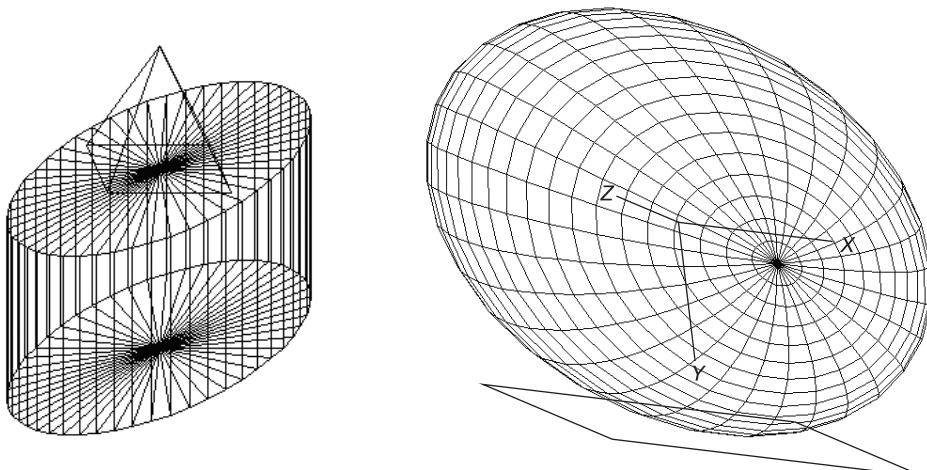
$$\operatorname{tg} \beta = \frac{OP}{\ell} = \frac{1}{1} = 1 \Rightarrow \beta = 45^\circ.$$

Для диметрической косоугольной проекции  $\ell = \frac{1}{2}$ :

$$\beta = \operatorname{arctg} \frac{OP}{\ell} = \operatorname{arctg} \frac{1}{1/2} = \operatorname{arctg} 2 \Rightarrow \beta = 63,4^\circ.$$

Для ортографической косоугольной проекции  $\ell = 0$ ,  $\beta = 90^\circ$ .

Пример косоугольной проекции приведен на рис. 11.12.



**Рис. 11.12.** Пример косоугольных проекций

Параллельные проекции реализуются через аффинные преобразования, которые являются комбинацией линейных преобразований, сопровождаемых переносом изображений. Для аффинных преобразований последний столбец в результирующей матрице преобразования размера  $4 \times 4$  должен быть равен

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

## 11.4. Математическое описание перспективной проекции

Перспективное преобразование имеет место в том случае, если последний столбец результирующей матрицы преобразования  $4 \times 4$  не нулевой.

Для простоты примем, что плоскость проецирования перпендикулярна оси  $Z$  и совпадает с плоскостью  $z = d$  (рис. 11.13).

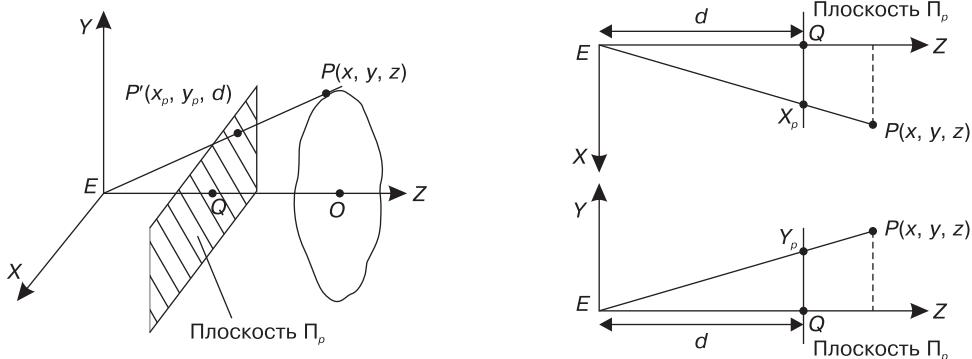


Рис. 11.13. Схема перспективной проекции

Для каждой точки  $P$  объекта точка изображения  $P'$  определяется как точка пересечения прямой линии  $OP$  и экрана.

Рассмотрим подобные треугольники:

$$\frac{x_p}{d} = \frac{x}{z}, \frac{y_p}{d} = \frac{y}{z} \Rightarrow x_p = \frac{d \cdot x}{z} = x \cdot \frac{d}{z}, y_p = \frac{d \cdot y}{z} = y \cdot \frac{d}{z}.$$

Расстояние  $d$  является в данном случае масштабным коэффициентом. Фактором, приводящим к тому, что удаленные объекты выглядят мельче, является деление на  $z$ . Допустимы все значения  $z$ , кроме  $z = 0$  (рис. 11.14).

В матричном виде перспективное преобразование выразится:

$$M_{\text{персп}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/d \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

$$\begin{aligned} P' &= P \cdot M_{\text{персп}} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot M_{\text{персп}} = \\ &= \begin{bmatrix} x & y & z & z/d \end{bmatrix} = \begin{bmatrix} x & y & d & 1 \end{bmatrix}. \end{aligned}$$

**Замечание 1.** Так как точка, принадлежащая оси  $Z$ , находится примерно в центре объекта, а ось  $Z$  совпадает с  $[OE]$ , то точка  $Q$  тоже будет находиться приблизительно в центре изображения объекта. Если расположить начало координат экрана (плоскости проецирования) в левом нижнем углу, а размеры экрана:  $a$  — по горизонтали,  $b$  — по вертикали, то

$$x_p = x \cdot \frac{d}{z} + \frac{a}{2}; \quad y_p = y \cdot \frac{d}{z} + \frac{b}{2}.$$

**Замечание 2.** Необходимо определить расстояние между точкой наблюдения  $E$  и экраном —  $d$  (рис. 11.15).

$$\frac{\text{Размер картинки}}{d} = \frac{\text{Размер объекта}}{\rho};$$

$$d = \rho \frac{\text{Размер картинки}}{\text{Размер объекта}}.$$

Это выражение равно применимо для горизонтальных и вертикальных размеров. Его следует использовать лишь для приблизительной оценки  $d$ , так как трехмерный объект может иметь сложную форму и не всегда ясно, какие размеры нужно включать в данное уравнение.

**Замечание 3.** Особенность нашего глаза такова, что мы можем видеть только точки, расположенные внутри определенного конуса, ось которого совпадает с направлением взгляда  $OE$ . Очень важный параметр этого конуса — угол  $\alpha_{\max}$  (см. рис. 11.15).

Глаз, как и камера, допускает только такие значения угла  $\alpha$ , которые не больше  $\alpha_{\max}$ . При выборе  $\alpha$  рекомендуется пользоваться формулой:

$$\operatorname{tg} \alpha = \frac{0,5 \cdot p - \text{Размер объекта}}{\rho}.$$

Выбор слишком малого  $\rho$  может привести к трудностям. Если же  $\rho$  будет слишком большим, то  $\alpha$  будет мал, эффект перспективы уменьшится.

Оптимальный вариант —  $\alpha = 40 \div 60^\circ$ .

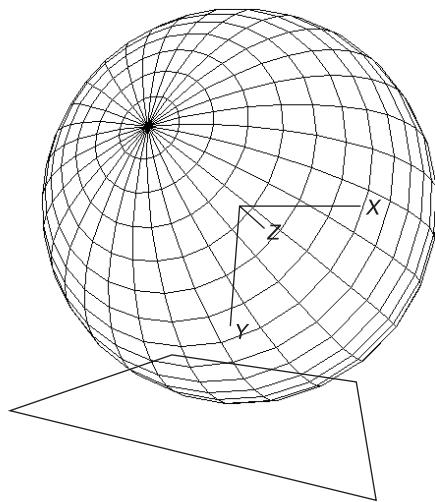


Рис. 11.14. Пример перспективной проекции

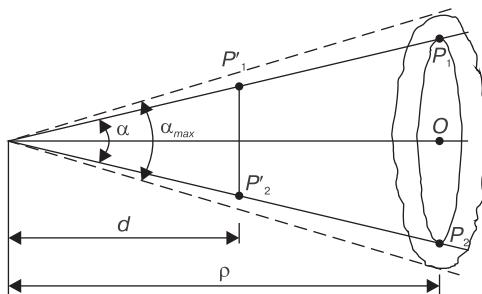


Рис. 11.15. Схема перспективного проецирования

**Замечание 4.** Рассмотрим случай, когда объект слишком длинный в направлении оси  $X$  (балка  $200 \times 2 \times 2$ ). Где лучше выбрать точку  $O$ ? До сих пор мы выбирали ее в середине объекта. Всегда ли она будет в центре объекта (рис. 11.16)?

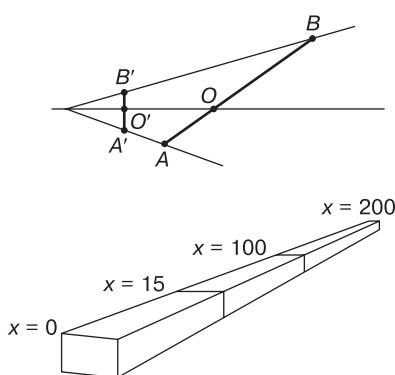


Рис. 11.16. Перспективное изображение балки

Точка  $O'$ , которая находится в центре изображения, не лежит на середине объекта.

Очевидно, что точку  $O$  нужно выбирать не в середине балки, а ближе к глазу.

Рассмотренные формулы описывают перспективную проекцию с одной точкой схода. Результирующая матрица такого преобразования имеет ненулевой элемент на третьей строке в четвертом столбце.

Двухточечная перспективная проекция реализуется с помощью матрицы, в которой два ненулевых элемента в четвертом столбце.

Если же три элемента четвертого столбца матрицы преобразования ненулевые, то получается трехточечная (косая) перспективная проекция.

## 11.5. Задание произвольных проекций. Видовое преобразование

В рассмотренных алгоритмах получения проекций были допущены ограничения на расположение картинной плоскости, центра проекции, что часто свойственно многим графическим пакетам.

Рассмотрим алгоритм получения проекций, когда картинная плоскость может располагаться произвольным образом относительно объекта. По сути, задача сводится к преобразованию систем координат (рис. 11.17).

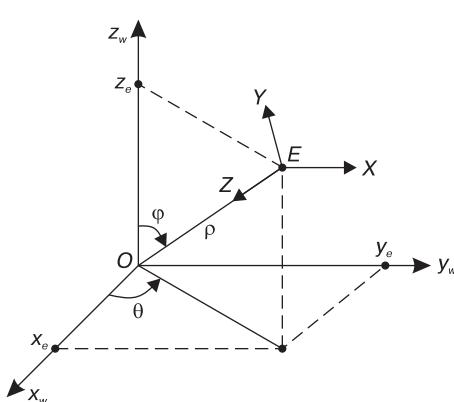


Рис. 11.17. Схема видового преобразования<sup>1</sup>

Пусть объект задан в мировой системе координат, причем ее начало находится приблизительно в центре объекта (точка  $O$ ). Объект наблюдается от точки  $E$  к точке  $O$ . В точке  $E$  расположим вторую систему координат — видовую. Вектор  $EO$  определяет направление наблюдения. Пусть точка  $E$  задана в сферических координатах  $\rho, \theta, \varphi$  по отношению к мировым. То есть ортогональные координаты  $x_e, y_e, z_e$  могут быть вычислены:

$$\begin{cases} x_e = \rho \sin \varphi \cos \theta; \\ y_e = \rho \sin \varphi \sin \theta; \\ z_e = \rho \cos \varphi. \end{cases}$$

<sup>1</sup> Котов Ю. В. Как рисует машина. — М.: Наука, 1988.

Ось  $Z$  видовой системы координат расположена по линии наблюдения,  $X$  — вправо,  $Y$  — вверх.

Мировая система координат — правосторонняя, видовая система координат — левосторонняя (обычно так выбирается) (рис. 11.18).

Видовое преобразование:

$$\begin{bmatrix} x_e & y_e & z_e & 1 \end{bmatrix} = \begin{bmatrix} x_w & y_w & z_w & 1 \end{bmatrix} \cdot V,$$

где  $V$  — матрица видового преобразования ( $4 \times 4$ ).

$V$  представляет собой четыре элементарных преобразования и получается путем их перемножения.

### Перенос

Перенос начала координат из точки  $O$  в точку  $E$ .

Точка  $E$  становится новым началом координат:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_e & -y_e & -z_e & 1 \end{bmatrix}.$$

### Поворот 1

Поворот системы координат вокруг оси  $Z$  на угол  $(\frac{\pi}{2} - \theta)$  в отрицательном направлении (по часовой стрелке). В результате ось  $Y$  совпадает по направлению с горизонтальной составляющей отрезка  $OE$ . Матрица такого изменения системы координат будет совпадать с матрицей для поворота точки на такой же угол в положительном направлении:

$$R_z = \begin{bmatrix} \cos(\frac{\pi}{2} - \theta) & \sin(\frac{\pi}{2} - \theta) & 0 & 0 \\ -\sin(\frac{\pi}{2} - \theta) & \cos(\frac{\pi}{2} - \theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

### Поворот 2

Поворот вокруг оси  $X$  на угол  $(\pi - \varphi)$  в положительном направлении, чтобы ось  $Z$  совпадала по направлению с  $OE$ . Этот поворот соответствует повороту точки на угол  $-(\pi - \varphi) = \varphi - \pi$ :

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\varphi - \pi) & \sin(\varphi - \pi) & 0 \\ 0 & -\sin(\varphi - \pi) & \cos(\varphi - \pi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

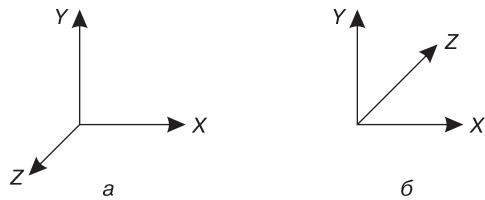


Рис. 11.18. Правосторонняя (а) и левосторонняя (б) системы координат

### Изменение направления оси X

После трех преобразований оси  $Y$  и  $Z$  имеют правильную ориентацию, а ось  $Z$  должна быть направлена в противоположную сторону:

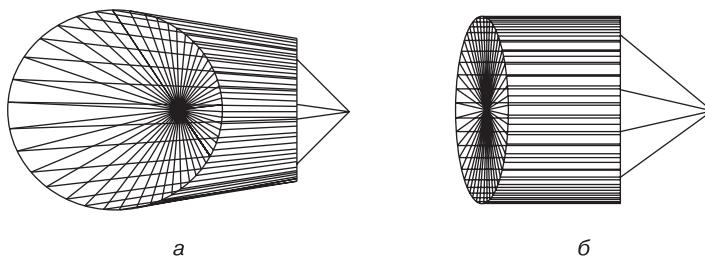
$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

### Результирующая матрица:

$$V = T \cdot R_z \cdot R_x \cdot S = \begin{bmatrix} -\sin\theta & -\cos\varphi\cos\theta & -\sin\varphi\cos\theta & 0 \\ \cos\theta & -\cos\varphi\sin\theta & -\sin\varphi\sin\theta & 0 \\ 0 & \sin\varphi & -\cos\varphi & 0 \\ 0 & 0 & \rho & 1 \end{bmatrix}.$$

Теперь к полученным видовым координатам можно применить любое из элементарных проекционных преобразований (например, ортографическое, косоугольное или перспективное) для получения окончательного результата.

Проанализировав влияние параметров на получаемый результат, можно отметить, что расстояние до картинной плоскости работает как масштабный коэффициент, делая проекцию больше или меньше. Расстояние же до объекта влияет на сходимость линий (рис. 11.19).



**Рис. 11.19.** Примеры перспективных проекций с разными значениями расстояния до объекта

Чем ближе точка зрения к объекту, тем больше угол зрения и искажение параллельности линий (см. рис. 11.19, *a*). И наоборот — чем дальше точка зрения от объекта, тем больше параллельные линии объекта стремятся к параллельности на проекции, а сама проекция стремится к параллельной (см. рис. 11.19, *b*).

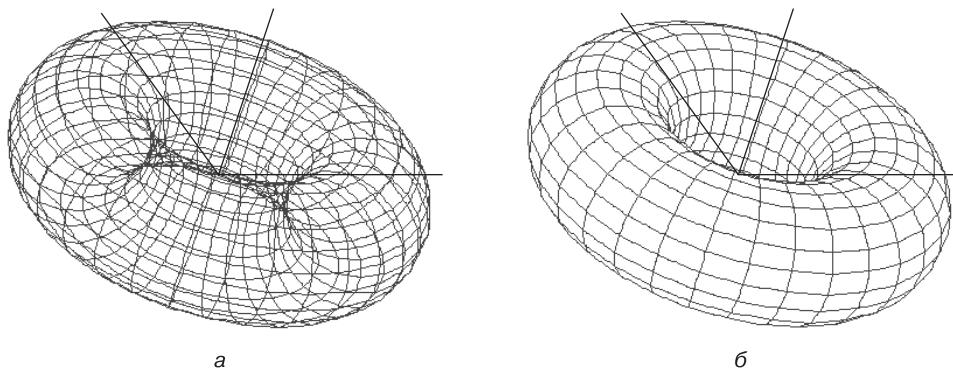
## Глава 12

# Алгоритмы удаления скрытых линий и поверхностей

Удаление невидимых поверхностей является классической задачей компьютерной графики. Начиная с 1970-х годов было предложено большое количество алгоритмов ее решения. Удаление невидимых поверхностей основывается на том факте, что если пользователь не видит некоторый объект, то нет необходимости этот объект визуализировать, то есть визуализировать нужно только полностью или частично видимые объекты.

### 12.1. Общие сведения об удалении скрытых линий и поверхностей

Алгоритмы удаления невидимых поверхностей определяют невидимые для пользователя части сцены и не используют их для отображения (рис. 12.1).



**Рис. 12.1.** Модель тора без удаления (а) и с удалением (б) невидимых линий

Многие алгоритмы ориентированы на специализированное применение. При выборе алгоритма учитываются количество объектов в сцене, их вид и способ расположения. Поэтому наилучшего решения задачи удаления скрытых линий и поверхностей не существует. Почти все алгоритмы удаления скрытых линий и поверхностей включают в себя сортировку по расстоянию от тела до точки наблюдения. Основная идея сортировки — чем дальше объект, тем больше вероятность, что он будет заслонен другим объектом.

Все алгоритмы удаления скрытых линий и поверхностей можно разделить на два типа.

1. Алгоритмы, работающие в объектном пространстве (ОП).

- Основной принцип этих алгоритмов — каждая из  $n$  граней сравнивается с оставшимися  $(n - 1)$  гранями.
- Они имеют дело с физической системой координат, в которой описаны эти объекты.
- Данные алгоритмы весьма точны (полученные изображения можно легко увеличить в несколько раз без потери качества).
- Объем вычислений теоретически —  $n^2$ , где  $n$  — количество объектов.

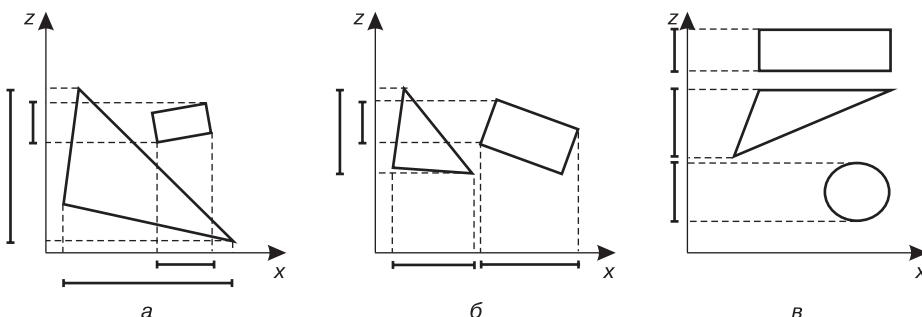
2. Алгоритмы, работающие в пространстве изображения (ПИ).

- Основной принцип этих алгоритмов — нужно определить, какая из  $n$  граней видна в каждой точке экрана, то есть каждый объект сравнивается с каждым пикселом экрана.
- Они имеют дело с системой координат экрана.
- Точность ограничена разрешающей способностью экрана (если полученные результаты потом увеличивать во много раз, они не будут соответствовать исходному изображению).
- Объем вычислений теоретически —  $N \cdot n$ , где  $N$  — количество пикселов на экране.

## 12.2. Алгоритм сортировки по глубине

Алгоритм сортировки по глубине относится к алгоритмам, работающим в объектном пространстве.

Основной его принцип — все объекты сортируются по глубине и выводятся на экран в обратном порядке, и, таким образом, более близкие объекты затирают более дальние. Данный алгоритм называют еще алгоритмом художника, так как, рисуя картины, обычно сначала изображают дальний план, а потом ближайшие объекты.



**Рис. 12.2.** Разрешение неопределенностей

Шаги алгоритма следующие.

1. Упорядочение всех объектов в соответствии с увеличением их  $z$ -координат.
2. Разрешение всех неопределенностей, которые возникают при перекрытии  $z$ -оболочек (дополнительно исследуется  $x$ -оболочка) (рис. 12.2).
3. Преобразование каждого из объектов в растровую форму в порядке уменьшения  $z$ -координаты.

На рис. 12.2, *в* неопределенностей нет, так как  $z$ -оболочки не перекрываются. На рис. 12.2, *а* и *б* происходит перекрытие  $z$ -оболочек, поэтому дополнительно исследуются  $x$ -оболочки. На рис. 12.2, *б* они не перекрываются, а на рис. 12.2, *а* перекрываются. В этом случае исследуются  $y$ -оболочки.

### 12.3. Алгоритм, использующий Z-буфер

Алгоритм, использующий  $Z$ -буфер, относится к алгоритмам, работающим в пространстве изображения.

Основной его принцип в том, что используются два буфера: регенерации, в котором хранятся значения пикселов, и  $Z$ -буфер, который хранит  $z$ -координаты.

Буфер регенерации заполняется значениями при параллельном анализе  $z$ -координаты со значениями  $Z$ -буфера.

Шаги алгоритма следующие.

1. В  $Z$ -буфер заносятся максимально возможные значения  $z$ .
2. Буфер регенерации заполняется значениями фона.
3. Каждый объект раскладывается в растр. Если координата  $z$  точки  $(x, y)$  меньше либо равна значению  $Z$ -буфера в элементе  $(x, y)$ , то:
  - 1)  $z(x, y)$  заносится в элемент  $(x, y)$   $Z$ -буфера;
  - 2) значение пикселя помещается в элемент  $(x, y)$  буфера регенерации.

Достоинствами данного алгоритма являются простота реализации и отсутствие сортировки.

Недостатки этого алгоритма такие:

- нужен большой объем памяти для хранения  $Z$ -буфера. Информация о значении пикселя занимает 24 бита, информация о глубине — около 20 бит;
- большая стоимость устранения лестничного эффекта.

Расчет координаты  $z$  производится из уравнения плоскости:

$$Ax + By + Cz + D = 0, \quad z = \frac{-D - Ax - By}{C}.$$

Если в точке  $(x_i, y_i) \rightarrow z_i$ , то в точке  $(x_{i+1}, y_i)$ , где  $x_{i+1} = x_i + \Delta x \rightarrow z_{i+1} = z_i - \frac{A}{C}(\Delta x)$ , если же  $\Delta x = 1$ , то  $z_{i+1} = z_i - \frac{A}{C}$ .

Пример работы алгоритма приведен на рис. 12.3. В буфере регенерации (БР) хранятся номера цветов, в примере вместо них для упрощения понимания стоят буквы (б — белый, к — красный, с — синий, з — зеленый). Сначала в Z-буфере хранится максимальное  $z$ , равное 5, а в БР — белый цвет. Потом происходит последовательное разложение в растр объектов, начиная с красного.

Z-буфер I	БР	Z-буфер II	БР
5 5 5 5 5 5	á á á á á á	5 5 5 5 5 5	á á á á á á
5 5 5 5 5 5	á á á á á á	5 5 5 5 5 5	á á á á á á
5 5 5 5 5 5	á á á á á á	5 5 5 5 5 5	á á á á á á
5 5 5 5 5 5	á á á á á á	5 5 5 5 5 5	á á á á á á
5 5 5 5 5 5	á á á á á á	1 1 5 5 5 5	ê ê á á á á
5 5 5 5 5 5	á á á á á á	1 1 5 5 5 5	ê ê á á á á

Z-буфер III	БР	Z-буфер IV	БР
5 5 5 5 5 5	б б б б б б	5 5 5 5 5 5	б б с с с с
5 5 5 5 5 5	б б б б б б	5 5 5 5 5 5	б з з з з с
5 3 3 3 3 5	б з з з з б	5 3 3 3 3 5	б з з з з с
5 3 3 3 3 5	б з з з з б	5 3 3 3 3 5	к к з з з б
1 1 3 3 3 5	к к з з з б	1 1 3 3 3 5	к к б б б б
1 1 5 5 5 5	к к б б б б	1 1 5 5 5 5	

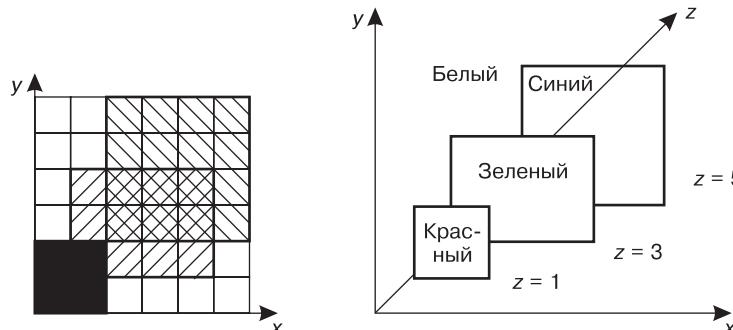


Рис. 12.3. Реализация алгоритма, использующего Z-буфер

## 12.4. Алгоритм построчного сканирования

Алгоритм построчного сканирования относится к алгоритмам, работающим в пространстве изображения.

Он является расширением алгоритма преобразования многоугольника в раstroвую форму. Разница в том, что работа происходит не с одним многоугольником, а со всеми сразу.

Шаги алгоритма следующие.

- Создается таблица ребер (ТР). Она содержит все ребра многоугольников, отсортированные по меньшей  $y$ -координате.

Описание ребра содержит:

$y_{\max}$	$x(y_{\min})$	$\frac{1}{m}$	Указатель на многоугольник
------------	---------------	---------------	----------------------------

- Создается таблица многоугольников (ТМ).

Описание многоугольников содержит:

Коэффициенты уравнения плоскости ( $A, B, C, D$ )	Цвет	$\Phi\text{лаг} = \begin{cases} 0 & - \text{вне многоугольника;} \\ 1 & - \text{внутри многоугольника} \end{cases}$
---	------	---

- Создается таблица активных ребер (ТАР).

Содержит все активные ребра на текущей сканирующей строке. Ребра упорядочены по возрастанию  $x$ -координаты.

$y_{\max}$	$x_i$	$\frac{1}{m}$	Указание на многоугольник
------------	-------	---------------	---------------------------

Пример работы алгоритма показан на рис. 12.4. Имеются два треугольника. Сканирующая строка  $\alpha$  пересекает четыре ребра.

Ниже рассмотрены четыре интервала, которые при этом образуются. Происходит анализ флагов пересекаемых многоугольников. Если все флаги равны нулю, цвет участка совпадает с цветом фона. Если есть только один флаг, равный нулю, то участок закрашивается цветом пересекаемого многоугольника. Если имеется более одного флага, равного нулю, то необходимо рассчитать, какой многоугольник из пересекаемых находится ближе, и участок закрашивается его цветом.

□ Для отрезка  $[LM]$ :

- флаг 1 = 1;
- флаг 2 = 0;
- цвет = цвету 1.

Флаг 1 равен 1, флаг 2 равен 0, значит, извлекаем цвет 1.

□ Для отрезка  $[MN]$ :

- флаг 2 = 0;
- флаг 1 = 0;
- цвет = фону.

□ Для отрезка  $[NO]$ :

- флаг 2 = 1;
- флаг 1 = 0;
- цвет = цвету 2.

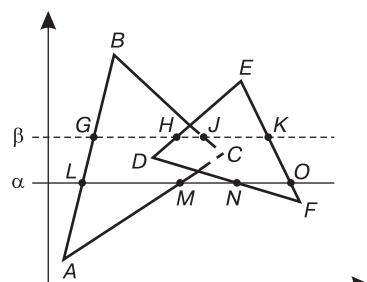


Рис. 12.4. Пример работы алгоритма построчного сканирования

□ Для отрезка [ $O$  – граница]:

- флаг 2 = 0;
- флаг 1 = 0;
- цвет = фону.

Строка  $\beta$  – четыре пересекаемых ребра в ТАР.

□ Для отрезка [ $GH$ ]:

- флаг 1 = 1;
- флаг 2 = 0;
- цвет = цвету 1.

□ Для отрезка [ $HJ$ ]:

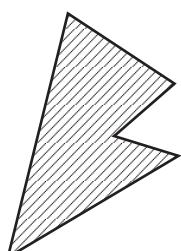
- флаг 1 = 1;
- флаг 2 = 1;
- вычисляются  $z_1$  и  $z_2$  из уравнений плоскости, цвет = цвету 2.

□ Для отрезка [ $JK$ ]:

- флаг 1 = 0;
- флаг 2 = 1;
- цвет = цвету 2.

□ Для отрезка [ $K$  – граница]:

- флаг 1 = 0;
- флаг 2 = 0;
- цвет = фону.



Алгоритм немного усложняется, если многоугольники проникают друг в друга. Тогда находят линию пересечения, и многоугольник разбивается на несколько многоугольников (рис. 12.5).

Удобно использовать *принцип когерентности по глубине*: если при переходе к следующей сканирующей строке ребра остаются те же и расположены в том же порядке, то соотношения глубин остаются теми же и их не нужно вычислять.

**Рис. 12.5.** Пример расположения многоугольников с проникновением

## 12.5. Алгоритм разбиения области

Алгоритм разбиения области относится к алгоритмам, работающим в пространстве изображения.

В его основе лежит гипотеза о способе обработки информации глазом и мозгом. Области, более насыщенные информацией, дольше прокручиваются к себе взгляд.

Широко используется когерентность (однородность смежных областей), которая позволяет снизить трудоемкость алгоритмов, работающих в пространстве изображения.

Основной принцип алгоритма следующий. Область разбивается на части, и в каждой части решается вопрос о том, достаточно ли она проста для визуа-

лизации. Если это не так, то часть разбивается дальше до тех пор, пока не станет простой или ее величина не достигнет размера пикселя.

Обычно достаточно около 10 разбиений.

Конкретная реализация алгоритма зависит от метода разбиения и критерия определения простоты изображения в части.

**Вариант 1.** Область разбивается последовательно на четыре равные прямоугольные части (рис. 12.6). Критерий простоты — объекты не попадают в область.

Первоначальное разбиение делит область на четыре части, не удовлетворяющие критерию простоты (в каждую часть попадают объекты). Поэтому необходимо выполнять деление дальше (на примере показано деление нижней левой четверти). Из вновь полученных новых четырех частей нижняя левая удовлетворяет критерию простоты. Далее она не рассматривается. А оставшиеся три делятся еще.

**Вариант 2.** Существуют четыре способа расположения объекта по отношению к части (рис. 12.7).

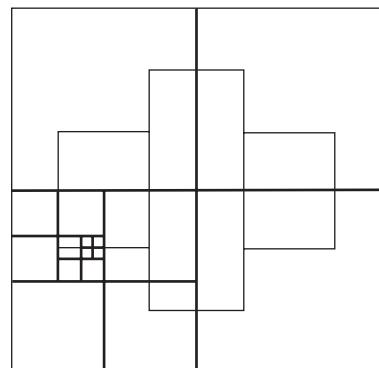


Рис. 12.6. Разбиение области

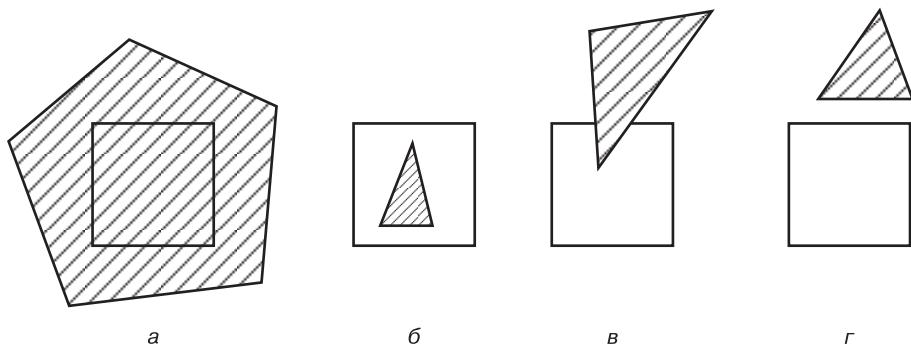


Рис. 12.7. Способы расположения объекта по отношению к части:  
а — охватывающий; б — внутренний; в — пересекающий; г — внешний

Область разбивается последовательно на четыре равные прямоугольные части. Критерий простоты — одна из четырех ситуаций.

1. Ни один многоугольник не пересекает область. Цвет части равен цвету фона.
2. Есть один внутренний или пересекающий многоугольник. Цвет части складывается из двух цветов — цвета фона и цвета многоугольника.
3. Есть один охватывающий многоугольник. Цвет части равен цвету этого многоугольника.

4. Есть несколько внутренних, пересекающих многоугольников и, как минимум, один охватывающий, расположенный ближе всех. Цвет части равен цвету охватывающего многоугольника.

Результат разбиения представлен на рис. 12.8.

Части, не требующие дальнейшего деления, помечены цифрой, соответствующей номеру ситуации критерия простоты. Разбиений становится меньше, но расчетов больше.

**Вариант 3.** Область разбивается относительно вершин многоугольника (рис. 12.9).

В этом случае разбиений становится меньше, хотя сами они реализуются медленнее.

Можно заметить, что трудоемкость алгоритмов по-разному зависит от количества примитивов. Так, алгоритм сортировки по глубине, работающий в объектном пространстве, становится медленным в сложных сценах. Его удобнее использовать в случае работы с небольшим количеством примитивов. Алгоритмы же, работающие в пространстве изображения, меньше зависят от количества объектов в сцене (их трудоемкость —  $N \cdot n$ ). Уменьшить время работы алгоритмов позволяет также использование принципа когерентности.

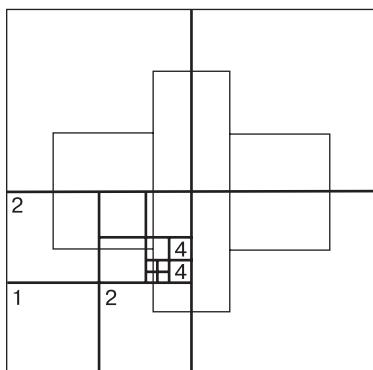


Рис. 12.8. Результат разбиения области

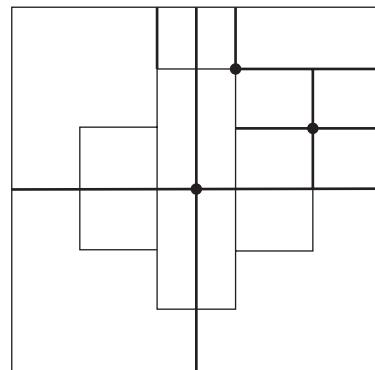


Рис. 12.9. Разбиение области относительно вершин многоугольника

## 12.6. Алгоритм плавающего горизонта

Алгоритм плавающего горизонта чаще всего используется для удаления невидимых линий трехмерных объектов, поверхности которых описаны в виде:

$$F(x, y, z) = 0.$$

Подобные функции возникают во многих приложениях в математике, технике, естественных науках.

Трехмерная задача сводится к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат  $x$ ,  $y$  или  $z$  (рис. 12.10).

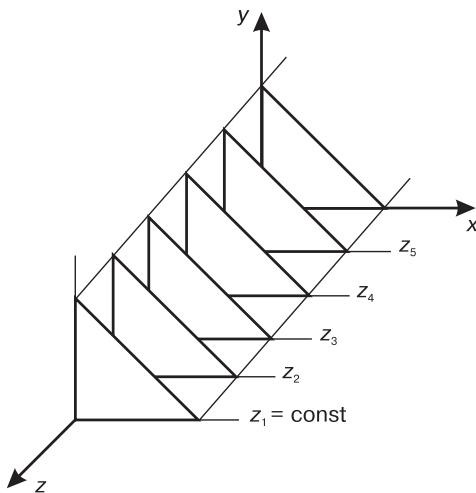


Рис. 12.10. Параллельные секущие плоскости

На рисунке показаны параллельные плоскости с  $z = \text{const}$ . Функция  $F(x, y, z) = 0$  сводится к последовательности кривых, лежащих в каждой из этих параллельных плоскостей, например к последовательности:

$$y = f(x, z), \quad x = g(y, z),$$

где  $z = \text{const}$  на каждой из заданных параллельных плоскостей.

Поверхность теперь складывается из последовательности кривых, лежащих в каждой из этих плоскостей (рис. 12.11).

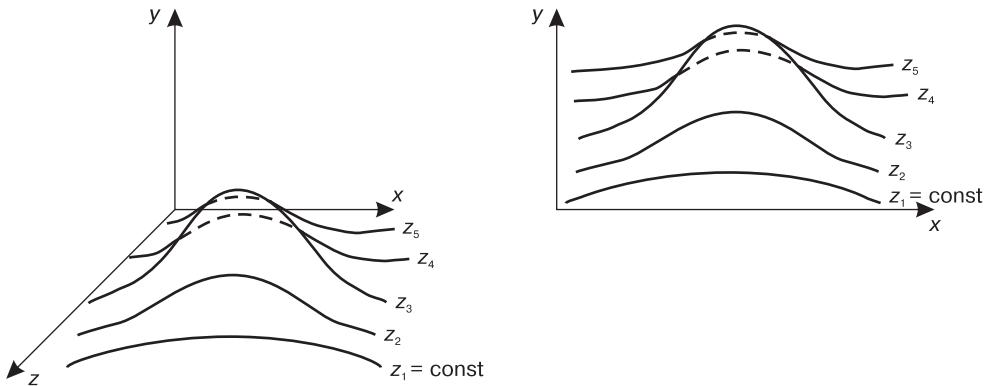


Рис. 12.11. Последовательность кривых поверхности

При проецировании кривых на плоскость  $z = 0$  (см. рис. 12.11) становится ясна идея алгоритма.

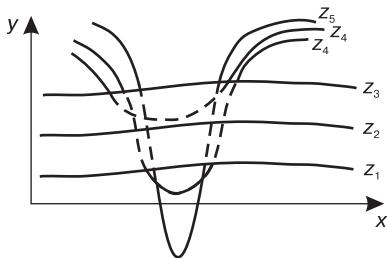
- Сначала происходит упорядочивание плоскостей  $z = \text{const}$  по возрастанию расстояния до них от точки наблюдения.

- Для каждой плоскости, начиная с ближайшей к точке наблюдения, строится кривая, лежащая на ней, то есть для каждого значения  $x$  в пространстве изображения определяется соответствующее  $y$ .
- Проводится проверка: если на текущей плоскости при некотором значении  $x$  соответствующее значение  $y$  на кривой больше максимального или меньше минимального по  $y$  для всех предыдущих кривых при этом  $x$ , то текущая кривая видима, иначе — нет (рис. 12.12).

Реализация алгоритма достаточна проста.

Для хранения максимальных и минимальных значений  $y$  при каждом значении  $x$  используются два массива, длина которых равна разрешению по  $x$ . Значения в этих массивах представляют собой текущие значения верхнего и нижнего плавающего горизонта. По мере рисования каждой очередной кривой данный горизонт «всплывает».

Фактически этот алгоритм работает каждый раз с одной линией.



**Рис. 12.12.** Реализация алгоритма плавающего горизонта

## 12.7. Алгоритм Робертса

Алгоритм Робертса относится к алгоритмам, работающим в объектном пространстве. Это математически элегантный алгоритм.

Сначала из каждого объекта удаляются ребра или грани, которые экранируются самим телом. Затем каждое из видимых ребер каждого объекта сравнивается с каждым из оставшихся объектов для определения того, какая его часть или части экранируются данными телами.

Это в сочетании с распространением растровых дисплеев привело к снижению интереса к данному алгоритму. Но математические методы, используемые в нем, просты, мощны и точны. В последующем введение предварительной сортировки по  $z$  снижает трудоемкость.

Необходимо, чтобы все изображаемые объекты были выпуклыми. Невыпуклые объекты разбиваются на выпуклые части. Выпуклый объект с плоскими гранями представляется набором пересекающихся плоскостей. Уравнение плоскости в пространстве:

$$ax + by + cz + d = 0.$$

В матричной форме:

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = 0,$$

или  $[x \ y \ z \ 1] \cdot [p]^T = 0$ ,  $[p]^T = [a \ b \ c \ d]$  — представляет собой плоскость.

Поэтому любой выпуклый объект можно выразить матрицей тела, состоящей из коэффициентов уравнений плоскостей:

$$[V] = \begin{bmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{bmatrix}.$$

Каждый столбец содержит коэффициенты одной плоскости. Любая точка пространства в однородных координатах:

$$[S] = [x \ y \ z \ 1].$$

Если точка  $S$  лежит на плоскости, то

$$[S] \cdot [P]^T = 0.$$

Если точка  $S$  не лежит на плоскости, то знак этого скалярного произведения показывает, по какую сторону от плоскости расположена точка. Если знак «+» — точка внутри тела, если «-» — вне его.

## 12.8. Алгоритм трассировки лучей

Алгоритм трассировки лучей относится к алгоритмам, работающим в объективном пространстве. Он является частью общей трассировки лучей, решающей самые разнообразные задачи на пути получения реалистичного изображения.

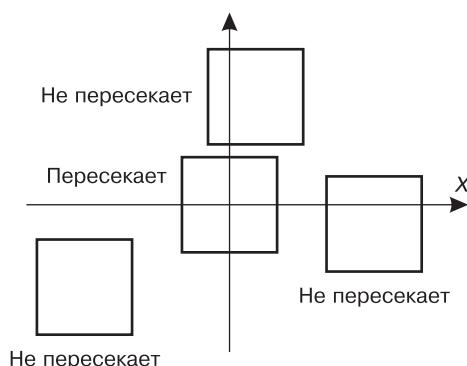
Предполагается, что наблюдатель находится на положительной полуоси  $Z$ , а экран дисплея перпендикулярен оси  $Z$  и располагается между объектом и наблюдателем.

Удаление невидимых (скрытых) поверхностей в алгоритме трассировки лучей выполняется следующим образом:

- сцена преобразуется в пространство изображения;
- из точки наблюдения в каждый пиксел экрана проводится луч и определяется, какие именно объекты сцены пересекаются с лучом;
- вычисляются и упорядочиваются по  $Z$ -координате точки пересечения объектов с лучом. В простейшем случае для непрозрачных поверхностей без отражений и преломлений видимой точкой будет точка с максимальным значением  $Z$ -координаты. Для более сложных случаев требуется сортировка точек пересечения вдоль луча.

Ясно, что наиболее важная часть алгоритма — процедура определения пересечения, которая в принципе выполняется  $Rx \times Ry \times N$  раз (здесь  $Rx, Ry$  — разрешение дисплея по  $X$  и  $Y$  соответственно, а  $N$  — количество многоугольников в сцене).

Очевидно, что повышение эффективности может достигаться сокращением времени вычисления пересечений и избавлением от ненужных вычислений.



**Рис. 12.13.** Определение пересечения луча и оболочки

Последнее обеспечивается использованием геометрически простой оболочки, в которую заключается объект, — если луч не пересекает оболочку, то не нужно вычислять пересечения с ним многоугольников, составляющих исследуемый объект.

При использовании прямоугольной оболочки определяется преобразование, совмещающее луч с осью  $Z$ . Оболочка подвергается этому преобразованию, а затем попарно сравниваются знаки  $X_{min}$  с  $X_{max}$  и  $Y_{min}$  с  $Y_{max}$ . Если они различны, то имеется пересечение луча с оболочкой (рис. 12.13).

При использовании сферической оболочки для определения пересечения луча со сферой достаточно рассчитать расстояние от луча до центра сферы. Если оно больше радиуса, то пересечения нет. Параметрическое уравнение луча, проходящего через две точки  $P1(x_1, y_1, z_1)$  и  $P2(x_2, y_2, z_2)$ , имеет вид:

$$P(t) = P1 + (P2 - P1) \cdot t.$$

Минимальное расстояние от точки центра сферы  $P0(x_0, y_0, z_0)$  до луча равно:

$$d^2 = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2.$$

Этому соответствует значение  $t$ :

$$t = -\frac{(x_2 - x_1)(x_1 - x_0) + (y_2 - y_1)(y_1 - y_0) + (z_2 - z_1)(z_1 - z_0)}{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

Если  $d^2 > R^2$ , то луч не пересекает объекты, заключенные в оболочку.

Дальнейшее сокращение расчетов пересечений основывается на использовании групп пространственно связанных объектов. Каждая такая группа окружается общей оболочкой. Получается иерархическая последовательность оболочек, вложенная в общую оболочку для всей сцены. Если луч не пересекает какую-либо оболочку, то из рассмотрения исключаются все оболочки, вложенные в нее, и, следовательно, объекты. Если же луч пересекает некоторую оболочку, то рекурсивно анализируются все вложенные в нее оболочки.

Наряду с вложенными оболочками для сокращения расчетов пересечений используется отложенное вычисление пересечений с объектами. Если обнаруживается, что объект пересекается лучом, то он заносится в специальный список пересеченных. После завершения обработки всех объектов сцены объекты, попавшие в список пересеченных, упорядочиваются по глубине. Заведомо невидимые отбрасываются, а для оставшихся выполняется расчет пересечений и отображается точка пересечения, наиболее близкая к наблюдателю.

Дополнительное сокращение объема вычислений может достигаться отбрасыванием нелицевых граней, учетов связности строк растрового разложения и т. д.

Для сокращения времени вычислений собственно пересечений предложено достаточно много алгоритмов, упрощающих вычисления для определенной формы задания поверхностей.

## 12.9. Иерархический Z-буфер

Представим себе виртуальную прогулку по городу с его растительностью, зданиями, мебелью внутри зданий со всеми ее ящиками, ножками, ручками и т. д. Традиционные алгоритмы определения текущей видимости объектов, реализуемые на существующем аппаратном обеспечении, вряд ли справятся с задачей визуализации сцен подобной сложности на скоростях, соответствующих интерактивной графике. Поэтому нужны быстрые алгоритмы, способные значительно ускорить просчет видимых частей и отброс всей невидимой части сцены.

Существуют три типа когерентных связей, присущих процессам просчета видимости объектов.

1. **Когерентность в объектном пространстве.** Во многих случаях однократное вычисление позволит определить видимость целого набора рядом расположенных объектов.
2. **Когерентность в пространстве изображения.** Очень часто однократное вычисление позволяет определить видимость объекта, покрывающего определенный набор пикселов на экране.
3. **Переходная когерентность.** Информация о видимости объектов в одном кадре зачастую может быть использована для ускорения просчетов в следующем кадре.

Иерархический Z-буфер реализует все три типа когерентных связей и позволяет на несколько порядков опередить традиционные технологии.

В основе лежат два алгоритма – Z-буфер (*Z-buffer*) и трассировка лучей (*ray tracing*). Традиционный Z-буфер достаточно эффективно использует когерентные связи в пространстве изображения. Но проблема традиционного Z-буфера состоит в том, что он совершенно не использует когерентные связи в объектном пространстве и переходную когерентность между кадрами. Каждый полигон просчитывается отдельно, и нам недоступна информация об уже произведенных расчетах в предыдущем кадре. Для сцен с высокой геометрической сложностью, как, например, модель города, данный подход очень неэффективен. Традиционный алгоритм будет тратить время на просчет каждого полигона у каждого объекта, каждого ящика у каждого письменного стола в здании, даже если все здание не будет видно, и все потому, что видимость определяется только на пиксельном уровне.

Традиционные методы трассировки лучей, наоборот, используют когерентные связи в объектном пространстве, реализуя пространственное деление. Луч из глаз наблюдателя проходит через структуру поделенного пространства, пока не коснется первой видимой поверхности. Как только луч достиг поверхности, уже нет необходимости рассматривать остальные поверхности в подпространствах, расположенных

за первой поверхностью по ходу луча. Таким образом, из дальнейшей обработки исключается большое количество геометрии. Получается значительное преимущество по сравнению с традиционным Z-буфером.

Иерархический Z-буфер объединяет в себе силы сразу двух этих алгоритмов. Для утилизации когерентных связей в объектном пространстве используется рекурсивное деление пространства на восемь подпространств. Реализация когерентности в пространстве изображения возложена на Z-буфер, усовершенствованный с помощью Z-пирамиды, которая позволяет очень эффективно отсекать невидимые части геометрии сцены. И наконец, для переходной когерентности используется уже просчитанная видимая геометрия из предыдущего кадра. Алгоритм несложен в реализации и применим для полигонных сеток любой сложности. И чем сложнее геометрия в сцене, тем заметнее будет разница в скорости просчетов по сравнению с традиционными алгоритмами.

Для начала все геометрическое пространство последовательно размещается в структуре дерева. Дерево рекурсивного деления пространства на восемь подпространств формируется следующим образом. Вся сцена помещается в минимально возможный, выровненный по осям координат куб. Этот куб будет являться базовым и соответствовать началу (корню) дерева. Дальнейшая процедура является рекурсивной и начинается с проверки содержащихся в данном кубе примитивов, и если их количество меньше определенного порогового значения, то рекурсия заканчивается. Если нет, то производится деление куба далее. В результате этого получается восемь новых дочерних кубиков с размером сторон  $1/2$  от размеров родительского куба. Эти восемь кубиков будут представлять первую ветвь дерева. Полученные кубы снова проверяются на соответствие содержащихся в них примитивов определенному пороговому количеству, и, если необходимо, каждый не удовлетворивший условию куб будет снова поделен на восемь меньших кубиков (подпространств). Этот процесс будет продолжаться до тех пор, пока каждый из кубов не будет содержать примитивов меньше, чем пороговое значение, или пока рекурсия не достигнет своего самого глубокого уровня.

После окончания формирования дерева рекурсивно выполняется следующая процедура: начиная с базового куба проверяется, попадает ли данный куб в поле зрения. Если нет, процедура на этом и заканчивается, если же да, то определяется видимость данного куба. Если куб невидим, процесс заканчивается, если видим, то визуализируется геометрия, ассоциированная с данным кубом, а затем рассматриваются его дочерние ветви (меньшие по размерам кубы) и т. д.

Алгоритм обрабатывает ту геометрию, которая содержится только в видимых кубах (видимых ветвях дерева). При этом часть просчитанных примитивов может быть полностью невидимой, но все они считаются *видимыми частично*. Частично видимыми они называются исходя из следующего: всегда найдется такая точка в пространстве, в которой данный полигон станет полностью видимым, и эта точка будет находиться не дальше, чем длина диагонали куба, содержащего данный полигон.

Алгоритм не тратит время на ненужные ветви дерева разбиений, так как он посещает только те ветви, родительские структуры которых видимы. Алгоритм никогда не посещает одни и те же ветви дважды.

Чтобы снизить стоимость процедуры определения видимости кубических пространств, используется  $Z$ -пирамида. Она позволяет очень быстро определить, видимо пространство или нет, исключая при этом попиксельные операции.

По сути,  $Z$ -пирамида очень напоминает пирамиду текстур с  $ti\!r$ -уровнями. Смысл  $Z$ -пирамиды — использование базового  $Z$ -буфера в качестве основания пирамиды. Это основание будет являться самым точным уровнем во всей пирамиде. Следующий, более грубый, уровень будет представлять собой набор значений, полученный путем выбора самого большого (наиболее удаленного) значения из четырех близлежащих значений предыдущего, более точного, уровня. И так далее. Таким образом, каждая запись в пирамиде будет представлять собой максимальное значение глубины из определенной квадратной области базового  $Z$ -буфера и соответствовать определенному фрагменту экрана. Самый верхний, наиболее грубый, уровень пирамиды (ее вершина) будет представлять единственную запись, содержащую самое большое значение координаты  $Z$  из базового  $Z$ -буфера, и соответствовать максимальной глубине всего изображения.

Поддерживать пирамиду в актуальном состоянии просто: каждый раз, когда обновляется значение базового  $Z$ -буфера, это значение последовательно продвигается по более грубым уровням до тех пор, пока не встречается та запись, значение которой находится так же далеко, как новое значение  $Z$ .

Проверка на видимость с помощью  $Z$ -пирамиды осуществляется следующим образом. Находится та запись в пирамиде, которая отображает минимально возможную квадратную площадь экрана, полностью содержащую исследуемый полигон. И если значение  $Z$  ближайшей вершины полигона будет дальше значения, содержащегося в этой записи, немедленно определяется, что полигон невидим. Этим методом пользуются для определения видимости как ветвей дерева, так и полигонов самой модели.

Если просчитывается следующий кадр анимации, то можно с большой вероятностью утверждать, что большинство кубов, видимых в предыдущем кадре, будет видимо и в следующем. Некоторые из видимых кубов станут невидимыми, а некоторые — наоборот, но когерентность между соседними кадрами в большинстве анимаций достаточно велика, и только небольшое количество кубов изменит свой статус при переходе между соседними кадрами (если, конечно, не произошла полная смена всей сцены). После создания первого кадра формируется и сохраняется перечень видимых кубов из него в виде списка. При переходе к формированию следующего, перед тем как с самого начала запустить иерархический алгоритм для нового кадра, проводится просчет геометрии, содержащейся в подпространствах списка. Кубы, геометрия из которых уже просчитана, помечаются.

Может случиться так, что при малой когерентности между кадрами или ее отсутствии такой подход вынудит впустую затратить время на предварительную рендеризацию геометрии из списка, так как все последующие циклы рекурсии все равно будут выполнены по полной программе без какого-либо выигрыша. Поэтому необходимо предусмотреть возможность отключения использования переходной когерентности в случаях резкой смены содержимого анимационной последовательности.

# **Глава 13**

## **Свет в компьютерной графике**

---

Получение геометрической модели объекта является первостепенной задачей компьютерной графики. Но не менее важно увидеть проектируемый объект с учетом особенностей тех материалов, из которых он изготовлен, изучить его в реальных световых условиях. Это актуально как в рекламных целях или на презентациях разрабатываемых объектов, когда необходимо представить на рассмотрение проект при отсутствии изготовленного образца, так и при проектировании. Например, при работе над новым образом здания архитектору желательно посмотреть его общий вид, представить, как строение впишется в существующую застройку, как будут распределяться тени. В процессе такого моделирования можно менять материалы и покрытие (текстуры) элементов проекта, проверять освещенность отдельных участков в зависимости от времени суток. Для моделирования освещенности, просчета теней, нанесения рисунка и создания неровностей на поверхности объекта необходимо изучение особенностей света.

### **13.1. Общие сведения о свете**

*Свет* — электромагнитная энергия, которая после взаимодействия с окружающей средой попадает в глаз, где в результате химических и физических реакций вырабатываются электроимпульсы, воспринимаемые мозгом.

Через опыт наш мозг учится определять и распознавать множество образов и отпечатков, которые создает свет из окружающей нас действительности. Младенец берет предмет, глядит на него мгновение, а затем тащит в рот. Его язык — это прекрасный датчик, который может определять форму и вид поверхности предмета практически так же, как глаз, а иногда и лучше. Ребенок учится ассоциировать то, что он видит, с той формой, которую описал ему язык. Со временем ребенок узнает, что один и тот же предмет может выглядеть по-разному в зависимости от того, как его держать, хотя он по-прежнему является тем же самым предметом. Это очевидно, подумаете вы, но было обнаружено, что слепым с рождения людям, которым медики вернули зрение, понять вышеизложенное очень сложно. Им также сложно усвоить смысл тени и отражения, суть которых зрячие люди познали еще от рождения. И сам факт того, что мы можем видеть, еще не означает, что мы можем понять то, что видим.

В этом и заключается разница между данными (data) и информацией (information). Данные — это световой образ, формирующийся на сетчатке глаза. Информация — это интерпретация данного образа нашим мозгом.

Создавая изображение любого вида, мы пытаемся сформировать световой образ на сетчатке глаза таким образом, чтобы он интерпретировался мозгом как предмет, который показывает это изображение. Тренированный мозг может извлечь из изображения огромное количество информации. Благодаря этому в голове мы можем получить полное трехмерное представление сцены, изображенной на двумерной картинке. Чтобы получить его, наш мозг анализирует порядок взаимодействия света со сценой (набором объектов, изображенных на картинке) и на основе такого анализа данных выдает нам конечное трехмерное представление сцены.

Разнообразие моделей освещения, применяемых в процессе формирования изображений компьютером, — это попытка увеличить количество информации, которую мозг сможет извлечь.

Человеческий мозг может извлечь и интерпретировать четыре информационных ресурса из потока видимых данных.

□ **Форма.** Это внешний вид объекта (предмета) в сцене, его видимые границы и края. Глаз человека обладает способностью улучшать четкость воспринимаемого изображения, что позволяет увереннее распознавать края предметов (многие компьютерные программы для обработки изображений используют алгоритмы повышения четкости, подобные тем, которые присущи глазу человека).

□ **Оттенки.** Блики и тени. Тон и структура поверхностей.

□ **Цвет.** Человеческим глазом могут быть обнаружены три цвета: красный, зеленый и синий.

□ **Движение.** Мозг человека особенно восприимчив к движению объектов. Прекрасно «камуфлированное» животное мгновенно будет обнаружено, если оно пошевелится. Очень часто, если потерян курсор на экране монитора, лучший способ найти его — сдвинуть мышку.

За обработку этих четырех информационных ресурсов отвечают специальные отделы головного мозга. Это было неоднократно доказано при анализе черепно-мозговых травм, полученных человеком. Как только человек получает травму и лишается отдела головного мозга, отвечающего за любой из перечисленных выше ресурсов, он сразу утрачивает способность к восприятию этой информации. Способность к восприятию принимается человеком как сама собой разумеющаяся. Принято считать, что если мы можем видеть, значит, в состоянии определить форму, оттенки, цвет и движение. Но это не всегда так.

Не менее важной является информация, которую мозг добавляет или удаляет во время анализа. Когда мы созерцаем, то имеем дело с гигантскими объемами информации. Было бы просто невозможно проанализировать и запомнить все сведения до мельчайших деталей. Да это и не нужно. Большая часть данных, поступающих нам через зрение, не обладает какой-либо ценностью. Мозг автоматически производит фильтрацию этого «мусора», позволяя нам сконцентрироваться на более значимой информации. Что еще более важно, мозг также добавляет недостающие сведения. Человеческое зрение имеет «мертвые зоны», но, тем не менее, мы этого не

замечаем, потому что пробелы всегда будут заполнены подходящей информацией. Наш мозг многое прощает.

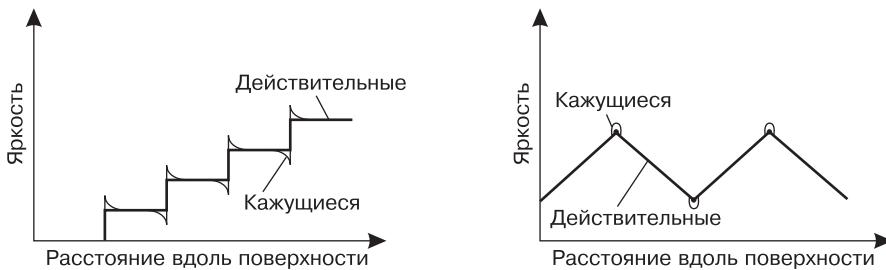
Для программиста это означает то, что ему совсем не нужно прорисовывать изображение с точностью до мельчайших деталей. Большинство из этих деталей будет просто проигнорировано и заполнено чем-то другим. Картина может быть значительно упрощена.

Конечное изображение может быть еще более упрощено, если сцена находится в движении. Если нажать паузу на видеомагнитофоне и посмотреть на неподвижную картинку, то будет заметно, что она выглядит никуда не годной, но это становится незаметным во время движения изображения.

Поэтому главное — грамотно разграничить, что и по каким алгоритмам выполняет программист, а что будет доделывать мозг. Цель программиста фотoreалистичной графики — попытаться смоделировать взаимодействие света с объектами сцены настолько аккуратно, чтобы оно могло выдержать скрупулезную проверку человеческим мозгом.

Кроме того, необходимо учитывать две особенности глаза.

- Глаз приспосабливается к средней яркости сцены, поэтому область с постоянной яркостью на темном фоне кажется ярче или светлее, чем на светлом.
- Границы областей постоянной яркости кажутся более яркими (рис. 13.1).



**Рис. 13.1.** Полосы Маха

Данный эффект является причиной слишком резкого перепада яркости на граничных ребрах, где происходит изменение яркости между соседними плоскостями. Это явление называется *эффектом полос Маха*.

На рис. 13.1 показаны действительные и кажущиеся изменения яркости вдоль поверхности, вызванныеliteralным торможением рецепторов глаза. Рецепторы глаза при реакции на свет подвергаются воздействию соседних рецепторов. Рецепторы, расположенные на границе перепада яркостей с более яркой ее стороны, подвергаются более сильному раздражению, чем те, которые находятся дальше от границы. Это объясняется тем, что они меньше затормаживаются своими соседями с более темной стороны. И наоборот, рецепторы, расположенные на границе с более темной стороной, подвергаются меньшему воздействию, чем находящиеся дальше от границы. Причина в том, что они больше затормаживаются своими соседями с яркой стороны границы.

Эффект полос Маха мешает глазу создавать сглаженное изображение сцены. Увеличивая количество полигональных граней, его можно ослабить, но полностью устраниТЬ нельзя.

## 13.2. Модель освещения

Световая энергия, падающая на поверхность, может быть:

- поглощена (превращена в тепло);
- отражена;
- пропущена.

Объект можно увидеть, если он отражает или пропускает свет. Если объект поглощает весь падающий свет, то он невидим и называется *абсолютно черным телом*. Количество поглощенной, отраженной или пропущенной энергии зависит от длины волны света. Если поглощаются лишь определенные длины волн, то у света, исходящего от объекта, изменяется распределение энергии и объект выглядит цветным. Так, зеленая трава отражает зеленый свет, а остальные поглощает.

Свойства отраженного света зависят:

- от вида источника света;
- его ориентации;
- свойств поверхности.

### Свойства объектов

В основу классификации объектов по характеру отражения падающего света положено пространственное распределение отраженного света. Определяющее влияние на характер распределения оказывает структура поверхности объекта.

Существуют четыре типа поверхностей.

- Ортотропные поверхности.** Отражают падающий свет равномерно (диффузно) по всем направлениям (рис. 13.2). Их называют *диффузными* (ламбертовскими). Эти поверхности доминируют среди естественных и искусственных объектов: пески, рыхлый снег, сухой асфальт, грунт. Их отличительная особенность — независимость яркости от положения наблюдателя.
- Зеркальные поверхности.** Отражают падающий свет преимущественно под углом, равным углу падения (рис. 13.3). К ним относятся чистые стеклянные поверхности, пластики, металлические поверхности, лед, сухие камни, поверхности

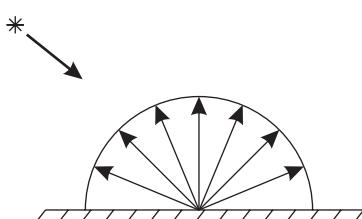


Рис. 13.2. Ортотропные поверхности

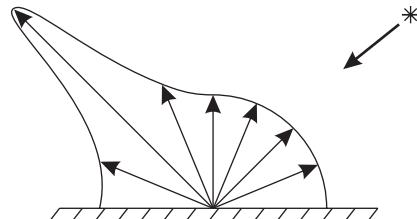
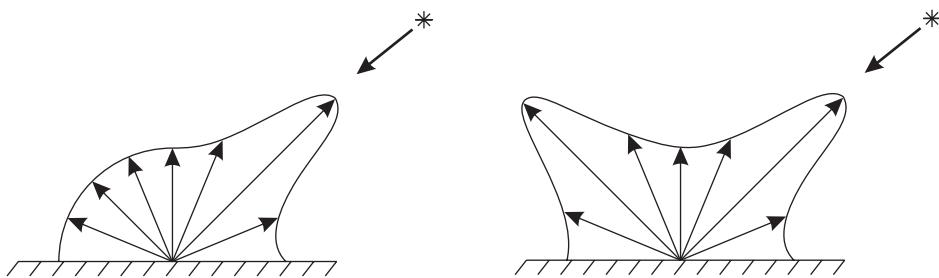


Рис. 13.3. Зеркальные поверхности

водных бассейнов. Применительно к реальным объектам термин «зеркальная поверхность» указывает на направленный характер отражения падающего света, но не означает, что отражение происходит в полном соответствии с законами геометрической оптики. Для идеальных зеркальных отражений угол отражения равен углу падения (идеально отражающая поверхность — зеркало). При этом падающий свет рассеивается в некотором телесном угле.

- **Обратно отражающие поверхности.** Отражают свет преимущественно по направлению к источнику (рис. 13.4). Их называют изрытыми, антизеркальными, световозвращающими. Такое отражение характерно для сельскохозяйственных культур, лугов и другой растительности.
- **Поверхности со смешанным отражением.** Для таких поверхностей характерно наличие двух или трех типов отражения (рис. 13.5). В общем случае можно выделить диффузную, зеркальную и обратную составляющие, а индикатриса имеет два максимума. Такое отражение наблюдается у рисовых полей, лугов, покрытых росой, и других аналогичных объектов.



**Рис. 13.4.** Обратно отражающие поверхности

**Рис. 13.5.** Поверхности со смешанным отражением

С увеличением высоты шероховатостей зеркальная компонента уменьшается и отражение стремится к диффузному. Иногда диффузное отражение преобладает и для объектов с гладкими поверхностями (молочное стекло). В таком случае большая часть падающего света проникает в приповерхностный слой и рассеивается массой мелких неоднородностей (диффузное излучение из внутренних областей объекта).

### Диффузное отражение

Отражение от объекта может быть диффузным и зеркальным. При диффузном отражении свет как бы проникает под поверхность объекта, поглощается и вновь испускается. Положение наблюдателя не имеет значения, так как диффузно отраженный свет рассеивается равномерно по всем направлениям. Зеркальное же отражение происходит от внешней поверхности объекта. При диффузном отражении поверхности имеют одинаковую яркость независимо от угла обзора.

Свет точечного источника отражается от поверхности по закону Ламберта:

$$I_d = I_t k_d \cos\theta,$$

где  $I_d$  — интенсивность отраженного света;

$I_L$  — интенсивность точечного источника;

$k_d$  — коэффициент диффузного отражения ( $0 \leq k_d \leq 1$ );

$\theta$  — угол между направлением света и нормалью к поверхности (рис. 13.6).

Поверхность будет освещена больше, если свет падает на нее перпендикулярно ( $\theta = 0$ ), и меньше, если свет падает под любым другим углом, поскольку в этом случае увеличивается освещаемая площадь. Диффузно рассеянный свет является главным источником визуальной информации о геометрии трехмерных объектов.

Предметы, освещенные одним точечным источником света, выглядят контрастными (предмет в темной комнате при фотоспышке). В реальной ситуации на объекты падает еще и свет, отраженный от окружающей обстановки, например от стен комнаты и других предметов. Он называется *рассеянным*. Рассеянный свет — это окружающее объект освещение от удаленных источников, чье положение и характеристики неизвестны. Необходимость его учета обусловлена тем, что вклад данного света может быть достаточно велик — до 50 % от общей освещенности. Рассеянный свет задает цвет (и его интенсивность) объекта в отсутствии явных источников света или в тени. Он не несет никакой информации об объекте, кроме значения простого цвета, равномерно заливающего контур объекта.

Интенсивность такого освещения постоянна и равномерно распределена во всем пространстве, расчет его отражения поверхностью выполняется по формуле:

$$I = I_a k_a,$$

где  $I_a$  — интенсивность отраженного света;

$k_a$  — коэффициент диффузного отражения рассеянного света ( $0 \leq k_a \leq 1$ ).

И тогда интенсивность с учетом рассеянного света и диффузного отражения:

$$I = I_a k_a + I_L k_d \cos\theta.$$

Если есть два объекта, одинаково ориентированных относительно источника, но расположенных на разном расстоянии, то их интенсивность ( $I$ ) по данной формуле будет одинакова. А ведь интенсивность должна быть обратно пропорциональна расстоянию до объекта.

С учетом расстояния до объекта модель освещения примет вид:

$$I = I_a k_a + \frac{I_L k_d \cos\theta}{d + K},$$

где  $d$  — расстояние до объекта от точечного источника;

$K$  — произвольная константа.

Если предполагается, что точка наблюдения находится в бесконечности, то  $d$  определяется положением объекта, ближайшего к точке наблюдения.

Для цветных поверхностей модель освещения применяется к каждому из трех основных цветов.

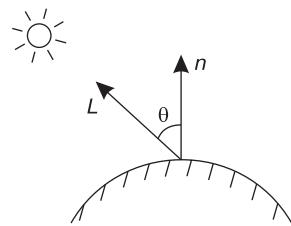


Рис. 13.6. Схема диффузного отражения

## Зеркальное отражение

Что означает термин «идеальное зеркало»? У такого зеркала идеально ровная отполированная поверхность, поэтому одному отраженному лучу соответствует только один падающий луч. Зеркало может быть затемненным, то есть поглощать часть световой энергии, но все равно сохраняется правило: один луч падает — один отражается. Можно рассматривать также «неидеальное зеркало». Это означает, что поверхность неровная. Один падающий луч порождает несколько отраженных лучей, образующих некоторый конус, возможно несимметричный, с осью вдоль линии падающего луча идеального зеркала. Конус соответствует некоторому закону распределения интенсивностей, простейший из которых описывается моделью Фонга — косинус угла, возведенный в некоторую степень.

Зеркальное отражение получается от любой гладкой поверхности. Если осветить ярким светом яблоко, возникнет световой блик, который получится в результате зеркального отражения, а свет, отраженный от остальной части яблока, будет диффузным. В месте светового блика яблоко кажется не красным, а белым, то есть окрашенным в цвет падающего света. Так как зеркально отраженный свет сфокусирован вдоль вектора отражения, блики при движении наблюдателя тоже смещаются.

Учитывать зеркальное отражение в модели освещения впервые предложил Фонг. Эти блики существенно увеличивают реалистичность изображения, ведь редкие реальные поверхности не отражают свет, поэтому данная составляющая очень важна. Особенно в движении, потому что по бликам сразу видно изменение положения камеры или самого объекта.

Зеркальное отражение света является направленным. Угол отражения от идеальной отражающей поверхности (зеркала) равен углу падения; в любом другом положении наблюдатель не видит зеркально отраженный свет ( $\alpha = 0$ ) (рис. 13.7).

Для неидеально отражающих поверхностей (яблоко) интенсивность отраженного света резко падает с увеличением  $\alpha$ . У гладких поверхностей распределение узкое, сфокусированное, у шероховатых — более широкое.

Эмпирическая модель Фонга:

$$\dot{I}_s = I_L \cdot \omega(\theta, \lambda) \cos^n \alpha,$$

где  $\omega(\theta, \lambda)$  — кривая отражения, представляющая собой отношение зеркально отраженного света к падающему как функцию угла падения  $\theta$  и длины волны  $\lambda$ ;

$\alpha$  — угол между отраженным лучом и направлением к наблюдателю.

Большие значения  $n$  дают сфокусированные распределения характеристик металлов и других блестящих поверхностей, а малые — более широкие распределения для мало блестящих поверхностей (рис. 13.8).

Коэффициент отражения  $n$  для металлов обычно больше 80 %, а для неметаллов — всего 4 %.

Функция  $\omega(\theta, \lambda)$  очень сложна, поэтому ее обычно заменяют коэффициентом  $k_s$ , который выбирается из эстетических соображений либо определяется экспериментально.

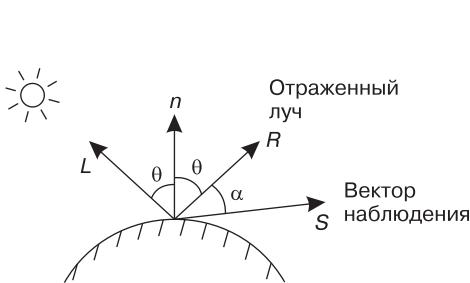
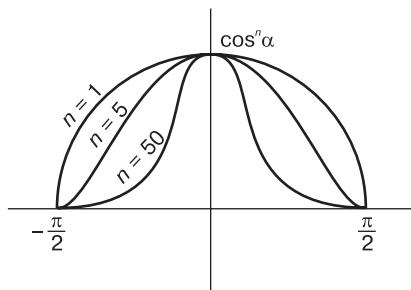


Рис. 13.7. Схема зеркального отражения

Рис. 13.8. Влияние показателя  $n$  на размер блика

Коэффициент  $k_s$  обычно одинаков для всех трех основных цветов.

Модель освещения с учетом рассеянного света, диффузного и зеркального отражений:

$$I = I_a k_a + \frac{I_L}{d+K} (k_d \cos \theta + k_s \cos^n \alpha_j).$$

Если есть несколько  $m$  источников света, то их эффекты суммируются:

$$I = I_a k_a + \sum_{j=1}^m \frac{I_{Lj}}{d_j+K} (k_d \cos \theta_j + k_s \cos^n \alpha_j).$$

### Пропускание света (прозрачность)

Поверхности могут пропускать свет направленно и диффузно. *Направленное* пропускание света происходит сквозь прозрачные вещества (стекло). Через них хорошо видны предметы, несмотря на то, что лучи света, как правило, преломляются, то есть отклоняются от первоначального направления. *Диффузное* пропускание света происходит сквозь просвечивающиеся материалы (замерзшее стекло), в которых поверхностные неоднородности приводят к беспорядочному перемешиванию световых лучей. Поэтому очертания предмета, рассмотренного через такие материалы, размыты.

При переходе из одной среды в другую световой луч преломляется (торчащая из воды палка кажется согнутой) (рис. 13.9). Преломление рассчитывается по закону Снеллиуса: падающий и преломляющийся лучи лежат в одной плоскости, а углы падения и преломления определяются по формуле:

$$\eta_1 \sin \theta = \eta_2 \sin \theta',$$

где  $\eta_1, \eta_2$  — показатели преломления двух сред.

Моделирование пропускания света осуществляется несколькими способами. В простейшем случае преломление

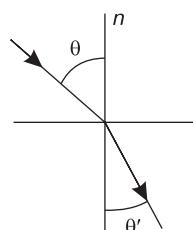
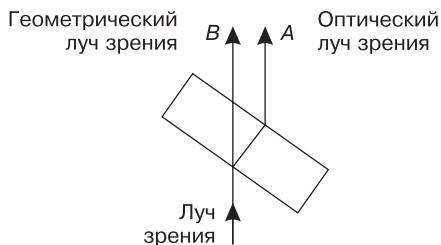


Рис. 13.9. Преломление лучей



**Рис. 13.10.** Пропускание света с преломлением и без преломления

не учитывается совсем, и световые лучи пересекают поверхность без изменения направления. Таким образом, все, что видимо на луче зрения при его прохождении через прозрачную поверхность, геометрически также принадлежит этому лучу. При наличии преломления геометрический и оптический лучи зрения не совпадают (рис. 13.10). Без учета преломления виден предмет *B*, с преломлением — предмет *A*. На первый взгляд, достаточно знать угловые соотно-

шения в точках пересечения луча с объектом. Но это не так, так как длина пути луча в объекте тоже меняется, поэтому не совпадают точки выхода луча из объекта и меняется количество поглощенного объектом света. Вследствие этого исходящий луч имеет другую интенсивность.

Простое пропускание света можно встроить в любой алгоритм удаления невидимых поверхностей, кроме Z-буфера, так как поверхности в нем обрабатываются в произвольном порядке. Если используется алгоритм построчного сканирования и передний многоугольник оказывается прозрачным, определяется ближайший из других многоугольников, внутри которых находится сканирующая строка. Уровень закраски определяется как взвешенная сумма уровней, вычисленных для каждого из двух многоугольников:

$$I = kI_1 + (1 - k)I_2,$$

где  $I_1$  — интенсивность видимой поверхности;

$I_2$  — интенсивность поверхности, расположенной за видимой;

$k$  — коэффициент прозрачности первой поверхности (если  $k = 0$ , первая поверхность полностью прозрачна, если  $k = 1$ , поверхность полностью непрозрачна).

Если вторая поверхность тоже прозрачна, то алгоритм применяется рекуррентно, пока не встретится непрозрачная поверхность или фон.

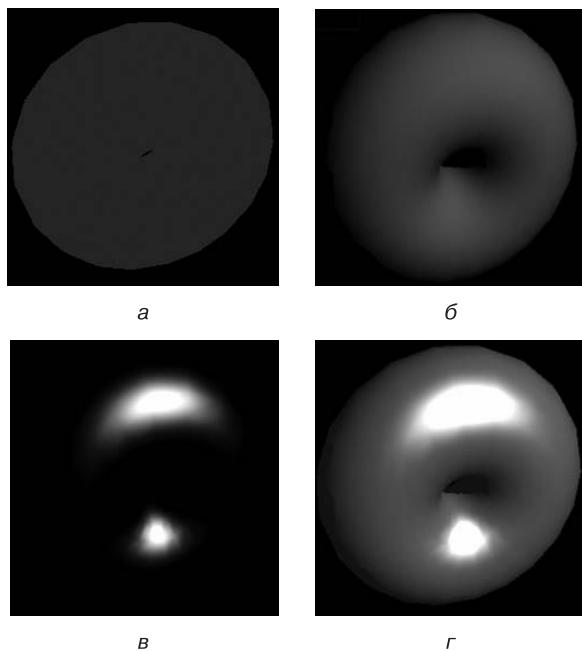
При расчете общей интенсивности обычно используется направленный пропущенный свет, поскольку учет диффузного пропускания вызывает много сложностей. Поэтому моделируются только прозрачные вещества.

Общий вид модели освещения:

$$I = k_a I_a + k_d I_d + k_s I_s + k_t I_t,$$

где  $a$  — рассеянный свет,  $d$  — диффузно отраженный свет,  $s$  — зеркально отраженный свет,  $t$  — пропущенный свет.

На рис. 13.11 показаны результаты освещения. На первом рисунке (*а*) тор закрашен с учетом рассеянного света, на втором (*б*) — диффузного отражения и рассеянного света, на третьем (*в*) показано зеркальное отражение, а на четвертом (*г*) — все три составляющие вместе.



**Рис. 13.11.** Результаты освещения с разными составляющими отражения

### Специальные модели

Для исследования общих закономерностей отражения поверхностей сложной структуры (природные объекты) используют специальные модели.

1. **Модель Торренса – Спэрроу (фацетная модель).** Поверхность представляется в виде совокупности случайно ориентированных микроскопических зеркальных граней. Отражение от каждой микрограмми определяется по формуле, затем методами геометрической оптики учитываются затенение микрограмм соседними и маскирование части зеркально отраженного света соседними микрограммами. Эта модель позволяет в аналитической форме учесть длину волны и угол падения лучей.
2. **Слоистая модель.** Используется для растительности, покрытой листвой. Каждый слой образован отдельными, в общем случае не перекрывающимися площадками определенных форм и размеров, обладающими ортотропным отражением. Отражение определяется затенением отражающих площадок нижних слоев вышележащими. Получить аналитическое решение данной модели сложно, обычно используют метод Монте-Карло. Результаты моделирования показывают, что поверхности такой структуры обладают обратным отражением.

Модели, основанные на статистическом описании структуры отражающих поверхностей, сложны. Это очень ограничивает их применение в компьютерной графике. Обычно используют приближенные модели.

### 13.3. Закраска полигональных сеток

Объект, представленный или аппроксимированный гранями, с учетом освещения точечным источником света может быть закрашен разными способами. Существуют три способа закраски объектов, заданных полигональными сетками:

- однотонная закраска;
- интерполяция интенсивностей (метод Гуро);
- интерполяция векторов нормали (метод Фонга).

#### Однотонная закраска

Данный вариант закраски самый простой. Вычисляется один уровень интенсивности, который используется для закраски всего полигонального многоугольника. Это чаще всего не соответствует реальной картине, так как освещенность в рамках грани меняется. Если из некоторой точки направить один луч к источнику света, а второй — перпендикулярно поверхности, то угол между этими лучами не будет постоянен в рамках полигональной грани. Исключение составляет случай, когда источник света находится в бесконечности. Чем ближе источник света расположен к поверхности, тем больше будет различие в освещенности разных точек грани.

При однотонной закраске предполагается, что:

- источник света расположен в бесконечности ( $\cos\theta = \text{const}$  на всей полигональной грани);
- наблюдатель находится в бесконечности ( $\cos\alpha = \text{const}$  на всей полигональной грани);
- многоугольник представляет собой реальную моделируемую поверхность, а не является аппроксимацией криволинейной поверхности.



**Рис. 13.12.** Пример однотонной закраски

Если первое или второе условие неприменимо, можно использовать усредненные значения  $\cos\theta$ ,  $\cos\alpha$ , вычисленные в центре многоугольника.

Третье предположение тоже часто не выполняется, но оно оказывает большое влияние на результат: каждая из видимых граней аппроксимированной поверхности хорошо отличима от других, так как интенсивность этих граней отличается от интенсивности соседних граней, то есть наблюдается эффект полос Маха (рис. 13.12).

#### Интерполяция интенсивностей (метод Гуро)

Метод Гуро позволяет получать слаженный объект на этапе визуализации без внесения изменений в геометрическую модель (полигональные сетки). Полосы Маха значительно уменьшаются. Происходит сглаживание ребер. Поэтому данный метод используется для объектов, аппроксимированных гранями.

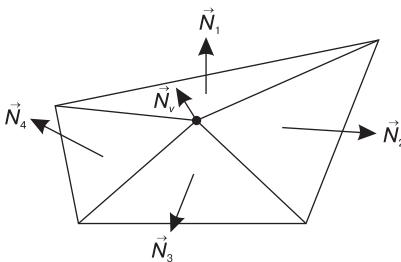
Процесс закраски осуществляется в четыре этапа.

1. Вычисляются нормали к поверхностям ( $\vec{N}_1, \vec{N}_2, \vec{N}_3, \vec{N}_4$ ).
  2. Определяются нормали в вершинах путем усреднения нормалей по всем граням, которым принадлежит вершина (рис. 13.13):
- $$\vec{N}_v = (\vec{N}_1 + \vec{N}_2 + \vec{N}_3 + \vec{N}_4)/4.$$
3. Используя нормали в вершинах и применяя произвольный метод закраски, вычисляются значения интенсивности в вершинах.
  4. Каждый многоугольник закрашивается путем линейной интерполяции значений интенсивности в вершинах сначала вдоль каждого ребра, а затем между ребрами вдоль каждой сканирующей строки (рис. 13.14):

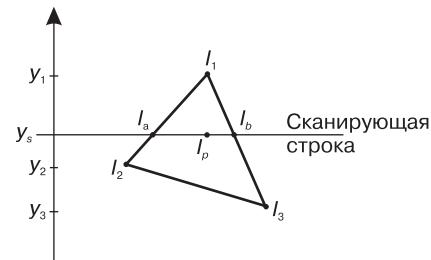
$$I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2};$$

$$I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3};$$

$$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}.$$



**Рис. 13.13.** Определение нормалей



**Рис. 13.14.** Определение значений интенсивности путем линейной интерполяции

Интерполяция вдоль ребер легко объединяется с алгоритмом удаления скрытых поверхностей, построенным на принципе построчного сканирования. Для всех ребер запоминается начальное значение интенсивности, а также изменение интенсивности при каждом единичном шаге по координате  $y$ . Заполнение видимого интервала на сканирующей строке производится путем интерполяции между значениями интенсивности на двух ребрах, ограничивающих интервал. Для цветных объектов отдельно интерполируется каждая из компонент цвета (рис. 13.15).



**Рис. 13.15.** Пример закраски

По методу Гуро точные значения интенсивности находятся лишь в вершинах полигональных сеток. Значения во всех остальных точках получаются на базе значений в вершинах. Поэтому данный тип закраски не подходит для зеркального отражения. Блик, если он не попадает на вершину, будет просто проигнорирован. Если же блик захватывает вершину, то он будет неестественно размазан вдоль грани, которым принадлежит эта вершина.

### Интерполяция векторов нормали (метод Фонга)

Закраска Фонга требует больших вычислительных затрат, но она позволяет разрешить многие проблемы метода Гуро. При закраске Гуро вдоль сканирующей строки интерполируется значение интенсивности, а при закраске Фонга — вектор нормали. Затем он используется в модели освещения для вычисления интенсивности пикселя. При этом достигается лучшая локальная аппроксимация кривизны поверхности, и получается более реалистичное изображение. Особенно правдоподобно выглядят зеркальные блики, которые в методе Гуро сильно размываются.



**Рис. 13.16.** Пример закраски методом Фонга

Этапы закраски следующие.

1. Вычисляются нормали к поверхностям.
2. Определяются нормали в вершинах путем усреднения нормалей по всем граням, которым принадлежит вершина.
3. Для каждой точки сканирующей строки определяется вектор нормали путем линейной интерполяции значений  $N$  (сначала в вершинах, затем между ребрами).
4. Для каждой точки сканирующей строки вычисляется значение интенсивности  $I$ .

Метод Фонга приводит к более качественным результатам, так как аппроксимация нормали осуществляется в каждой точке. Полосы Маха практически исчезают (рис. 13.16).

## 13.4. Тени

Изображение с построенными тенями выглядит гораздо реалистичнее, чем без них. Кроме того, тени очень важны для моделирования. Например, особо интересующий нас участок может оказаться невидимым из-за того, что он попадает в тень. А в строительстве и при разработке космических аппаратов тени влияют на расчет падающей солнечной энергии, обогрев и кондиционирование воздуха. Если положения наблюдателя и источника света совпадают, то теней не видно, но они появляются, когда наблюдатель перемещается в любую другую точку.

Тень состоит из двух частей: полной тени и полутиени. Полная тень — это центральная, темная, резко очерченная часть, а полутиень — окружающая ее более светлая часть. Распределенные источники света создают как полную тень, так

и полутень: в полной тени свет вообще отсутствует, а полутень освещается частью распределенного источника.

Точечные источники создают только полную тень. Из-за больших вычислительных затрат рассматривается только полная тень, образуемая точечным источником света.

Легче всего, когда источник находится в бесконечности, тогда тени определяются с помощью ортогонального проецирования. Если источник расположен на конечном расстоянии, то используется перспективная проекция.

Существуют два варианта образования тени (рис. 13.17):

- собственно тень на объекте;
- проекционная тень.

Собственно тень получается тогда, когда сам объект препятствует попаданию света на некоторые его грани. Алгоритм затенения в этом случае идентичен алгоритму удаления скрытых поверхностей. В последнем определяются поверхности, которые можно увидеть из точки зрения, а в алгоритме затенения — поверхности, которые можно «увидеть» из источника света. Поверхности, видимые из источника света и из точки зрения, не лежат в тени. Поверхности, видимые из точки зрения, но невидимые из источника света, находятся в тени. Поэтому удобно использовать один алгоритм, последовательно применяя его к точке зрения и к каждому из точечных источников света.

Если один объект препятствует попаданию света на другой, то получается проекционная тень. Чтобы найти такие тени, необходимо построить проекции граней на сцену. Центр проекции располагается в источнике света. Таким образом находятся теневые многоугольники для всех граней, которые заносятся в структуру данных. Чтобы не вносить в нее слишком много многоугольников, можно проецировать контур каждого объекта, а не отдельные грани.

Для создания различных видов из разных точек зрения не нужно вычислять тени заново, так как они зависят только от положения источника света и не зависят от положения наблюдателя.

### Источник света в бесконечности

Ниже приведена схема вычисления тени, когда источник света находится в бесконечности (рис. 13.18).

В случае бесконечно удаленного источника света предполагается, что лучи света, приходящие к объекту, полностью параллельны.

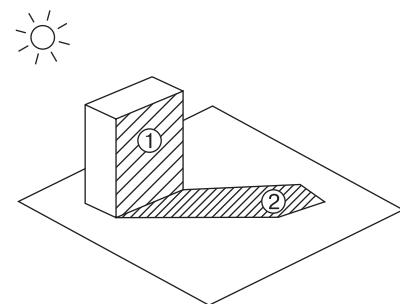


Рис. 13.17. Два варианта образования тени: 1 — тень на объекте; 2 — проекционная тень

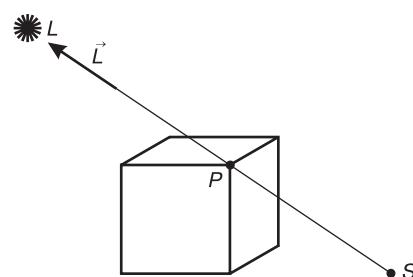


Рис. 13.18. Источник света в бесконечности

Это позволяет решить уравнение проекции только раз и применить полученное решение ко всем вершинам объекта.

Имея точку источника света  $(x_b, y_b, z_l)$  и вершину объекта  $(x_p, y_p, z_p)$ , необходимо получить проекцию вершины объекта на плоскость  $z = 0$ , то есть точку тени  $(x_s, y_s, z_s)$ .

Из подобных треугольников получаем:

$$\frac{x_p - x_s}{z_p - z_s} = \frac{x_l - x_p}{z_l - z_p}. \quad (13.1)$$

Решая это уравнение относительно  $x_s$ , получаем:

$$x_s = x_p - (z_p - z_s) \left( \frac{x_l - x_p}{z_l - z_p} \right). \quad (13.2)$$

Если принять, что  $L$  — это вектор из точки  $P$  к источнику света, то точку  $S$  можно выразить как:

$$S = P - \alpha \vec{L}. \quad (13.3)$$

Так как проекция производится на плоскость  $z = 0$ , то уравнение (13.3) можно переписать в следующем виде:

$$0 = z_p - \alpha z_l, \quad (13.4)$$

или

$$\alpha = \frac{z_p}{z_l}. \quad (13.5)$$

Решая (13.3) относительно  $x_s$  и  $y_s$ , получаем:

$$x_s = x_p - \frac{z_p}{z_l} x_l; y_s = y_p - \frac{z_p}{z_l} y_l, \quad (13.6)$$

или в матричной форме:

$$M_r = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{x_l}{z_l} & -\frac{y_l}{z_l} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (13.7)$$

Теперь, имея координаты точки  $P$  в мировом координатном пространстве, можно получить ее проекцию на плоскость  $z = 0$  просто путем умножения на матрицу  $M_r$ :

$$S = P \cdot M_r. \quad (13.8)$$

### Локальный источник

Уравнение (13.6) для бесконечно удаленного источника света может быть обобщено для случая, когда источник света находится на конечном расстоянии от

объекта. При этом понадобятся дополнительные вычисления на каждую вершину, так как каждая вершина имеет в общем случае свое собственное направление на источник света. Тем не менее, в такой ситуации мы тоже можем перенести большую часть вычислений в матрицу  $M_T$ .

Если  $L$  – это точка расположения источника света, то (13.3) принимает вид:

$$S = P + \alpha(P - L). \quad (13.9)$$

После проецирования на плоскость  $z = 0$ :

$$\alpha = \frac{-z_p}{z_p - z_l} \quad (13.10)$$

и

$$x_s = \frac{x_l z_p - x_p z_l}{z_p - z_l}; y_s = \frac{y_l z_p - y_p z_l}{z_p - z_l}. \quad (13.11)$$

Если использовать гомогенизацию после преобразования, то (13.11) можно записать в виде матрицы:

$$M_T = \begin{bmatrix} -z_l & 0 & 0 & 0 \\ 0 & -z_l & 0 & 0 \\ x_l & y_l & 0 & 1 \\ 0 & 0 & 0 & -z_l \end{bmatrix}. \quad (13.12)$$

Имея координаты точки  $P$  в мировом координатном пространстве, можно записать:

$$S = P \cdot M_T. \quad (13.13)$$

### 13.5. Фактура. Нанесение узора

В компьютерной графике *фактурой* называется детализация строения поверхности. Существуют два вида детализации.

- Нанесение заданного узора на гладкую поверхность (регулярная и стохастическая текстуры).
- Создание неровностей на поверхности.

#### Нанесение узора на поверхность. Регулярная текстура

Речь идет о рисунке, который чаще всего можно описать аналитически: клеточка, полосочка и т. д. Характерные точки узора из пространства текстуры (фактурное пространство (ФП)) переносятся в объектное пространство (ОП), затем в пространство изображения (ПИ) и определенным образом соединяются отрезками. Главным при этом является отображение, вследствие чего задача сводится к преобразованию систем координат.

Пусть рисунок узора задан в прямоугольной системе координат  $(u, w)$  ( $\Phi\Pi$ ), а поверхность — в другой прямоугольной системе координат  $(x, y)$  ( $O\Pi$ ). Для нанесения узора на поверхность нужно найти или задать функцию отображения одного пространства на другое:

$$x = f(u, w), y = g(u, w)$$

или

$$u = h(x, y), w = e(x, y).$$

Обычно предполагается, что функция отображения линейна:

$$x = Au + B;$$

$$y = Cw + D,$$

где коэффициенты  $A, B, C, D$  выводятся из соотношения между двумя известными точками в системах координат.

**Пример.** Узор в виде клеточки (рис. 13.19, *a*) нужно отобразить на часть поверхности, заданную десятой частью сферы (рис. 13.19, *б* и *в*).

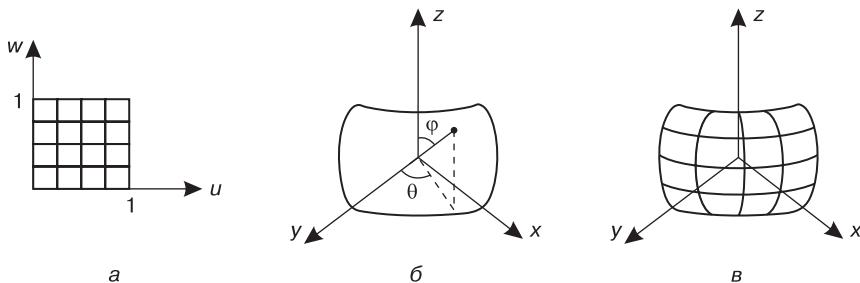


Рис. 13.19. Нанесение узора на поверхность<sup>1</sup>

Параметрическое представление куска сферы:

$$\begin{aligned} x &= \sin\theta \sin\varphi; \quad 0 \leq \theta \leq \frac{\pi}{2}; \\ y &= \cos\theta \sin\varphi; \\ z &= \cos\varphi. \quad \frac{\pi}{4} \leq \varphi \leq \frac{\pi}{2}. \end{aligned}$$

Пусть функция отображения линейна и имеет вид:

$$\theta = Au + B, \quad \varphi = Cw + D.$$

Углы узора переходят в углы куска следующим образом:

$$u = 0, w = 0 \text{ при } \theta = 0, \varphi = \frac{\pi}{2};$$

$$u = 1, w = 0 \text{ при } \theta = \frac{\pi}{2}, \varphi = \frac{\pi}{2};$$

<sup>1</sup> Фолли Дж., А Вэн Дэм. Основы интерактивной машинной графики. — М.: Мир, 1989.

$$u = 0, w = 1 \text{ при } \theta = 0, \varphi = \frac{\pi}{4};$$

$$u = 1, w = 1 \text{ при } \theta = \frac{\pi}{2}, \varphi = \frac{\pi}{4}.$$

$$\text{Отсюда } A = \frac{\pi}{2}, B = 0, C = -\frac{\pi}{4}, D = \frac{\pi}{2}.$$

$$\text{Функция отображения: } \theta = \frac{\pi}{2}u, \varphi = \frac{\pi}{2} - \frac{\pi}{4}w,$$

$$\text{или обратное преобразование: } u = \frac{\theta}{\pi/2}, w = \frac{\pi/2 - \varphi}{\pi/4}.$$

В табл. 13.1 приведено отображение одной линии узора из фактурного пространства ( $u - w$ ) в объектное пространство ( $\theta - \varphi$ ), а затем в декартовы координаты ( $x, y, z$ ).

Таблица 13.1. Значения параметров

$u$	$w$	$\theta$	$\varphi$	$x$	$y$	$z$
1/4	0	$\pi/2$	$\pi/2$	0,38	0,92	0
	1/4		$7\pi/16$	0,38	0,90	0,20
	1/2		$3\pi/8$	0,35	0,85	0,38
	3/4		$5\pi/16$	0,32	0,77	0,56
	1		$\pi/4$	0,27	0,65	0,71

### Нанесение узора на поверхность. Стохастическая текстура

Рассмотренный регулярный узор был задан математически, но узор может быть также нарисован от руки или получен путем сканирования фотографий, то есть иметь нерегулярный вид. В этом случае используется метод обратной трассировки лучей. Центр каждого пикселя изображения проецируется на поверхность объекта, и по координатам точки на поверхности определяется соответствующая ей точка в фактурном пространстве. Далее используются процедуры сглаживания для устранения дискретизации.

Для нанесения рисунка на поверхность необходимо:

- отображение объектного пространства (ОП) в пространство изображения (ПИ);
- преобразование из фактурного пространства (ФП) в ОП.

Изложенный принцип реализует алгоритм разбиения Кэтмула. Его основные шаги следующие.

1. Кусок поверхности разбивается на фрагменты до тех пор, пока фрагмент не будет покрывать центр только одного пикселя.
2. Производится отображение параметрических значений центра фрагмента или пикселя в ФП.
3. Находится интенсивность пикселя по узору.

**Пример.** (Исходные данные те же.) Узор задан на растре  $64 \times 64 \bar{p}$  (рис. 13.20).

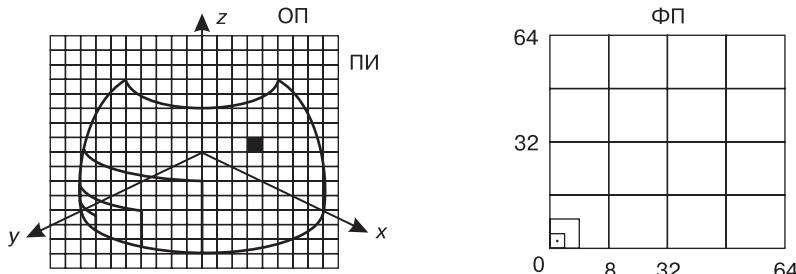


Рис. 13.20. Схема нанесения стохастического узора на поверхность<sup>1</sup>

Кусок поверхности разбивается на фрагменты. Для того чтобы фрагмент покрывал центр только одного пикселя, необходимо четыре разбиения. В ПИ этот фрагмент имеет прямоугольную форму. Пределы изменения  $\theta$  и  $\varphi$  в ОП:

$$0 \leq \theta \leq \frac{\pi}{32}, \frac{31\pi}{64} \leq \varphi \leq \frac{\pi}{2}.$$

С помощью функции обратного отображения из ОП ( $\theta - \varphi$ ) в ФП ( $u - w$ ):

$$u = \frac{\theta}{\pi/2}, w = \frac{\pi/2 - \varphi}{\pi/4}$$

получим координаты углов фрагмента в ФП:

$$\theta = 0, \varphi = \frac{\pi}{2} \rightarrow u = 0, w = 0;$$

$$\theta = 0, \varphi = \frac{31\pi}{64} \rightarrow u = 0, w = \frac{1}{16};$$

$$\theta = \frac{\pi}{32}, \varphi = \frac{31\pi}{64} \rightarrow u = \frac{1}{16}, w = \frac{1}{16};$$

$$\theta = \frac{\pi}{32}, \varphi = \frac{\pi}{2} \rightarrow u = \frac{1}{16}, w = 0.$$

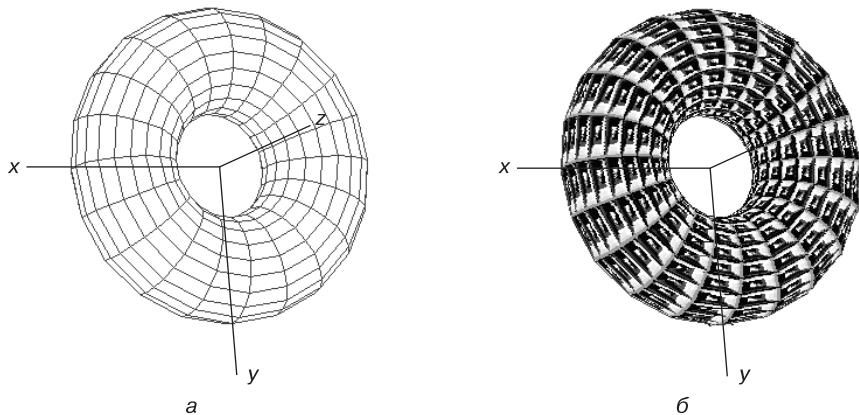
В ФП это квадрат. На растре  $64 \times 64$  часть  $1/16$  соответствует 4 пикселям. Интенсивность пикселя в ПИ определяется путем усреднения интенсивностей пикселов в соответствующей части ФП. Кусок фрагмента  $4 \times 4$  пикселя содержит 7 черных пикселов, поэтому в ПИ интенсивность равна  $7/16$ .

Недостатком метода является поточечная выборка, которая приводит к сильно-му лестничному эффекту.

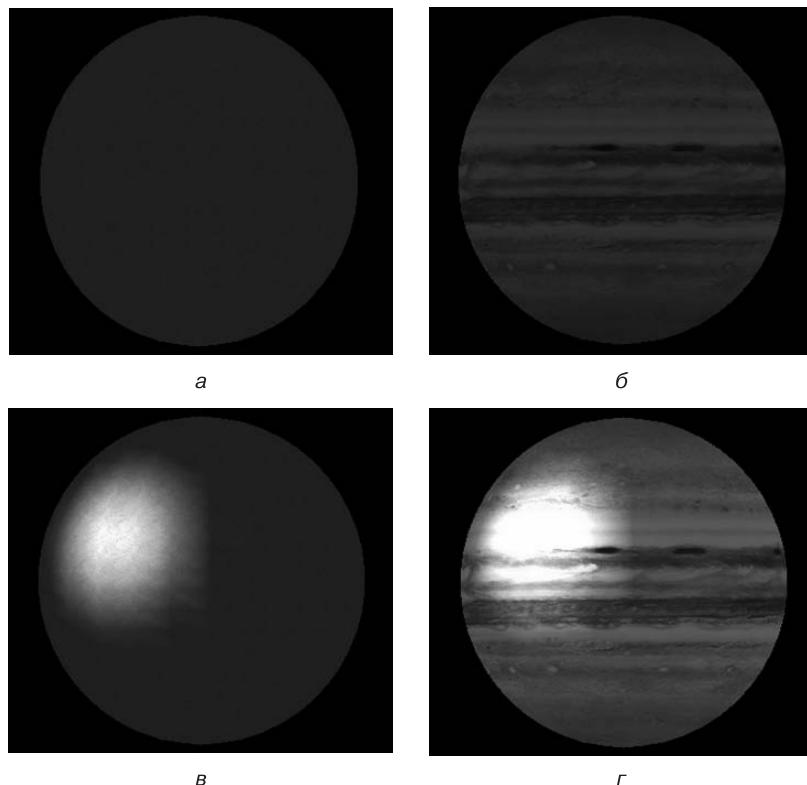
Нанесение стохастического узора на аппроксимированную поверхность тора (рис. 13.21, а и б) может быть выполнено по-разному. Рисунок, заданный на рас-

<sup>1</sup> Фолли Дж., А Вэн Дэм. Основы интерактивной машинной графики. – М.: Мир, 1989.

тре, может полностью наноситься на каждую грань без учета ее размеров, а может определяться часть узора для некоторого полигонального многоугольника.



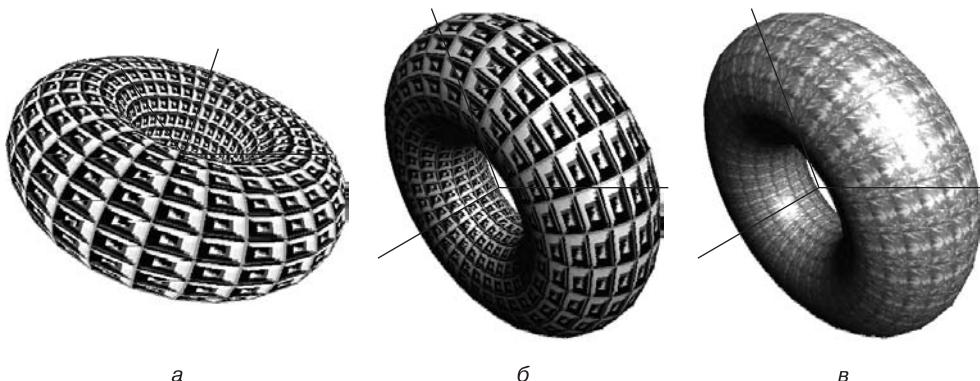
**Рис. 13.21.** Пример нанесения стохастического узора на поверхность



**Рис. 13.22.** Пример закраски: *а* — с рассеянным светом; *б* — с рассеянным светом и текстурированием; *в* — с рассеянным светом, диффузным и зеркальным отражением; *г* — с рассеянным светом, текстурированием, диффузным и зеркальным отражением

Если же добавить освещение объекта по рассмотренной ранее модели расчета интенсивности света, то можно получить достаточно реалистичные изображения (рис. 13.22). На первом рисунке (*а*) показан шар, освещенный рассеянным светом, на втором (*б*) — с рассеянным светом и наложенным стохастическим рисунком, на третьем (*в*) — с рассеянным светом, зеркальным и диффузным отражением, на четвертом (*г*) — с рассеянным светом, зеркальным, диффузным отражением и наложенным стохастическим рисунком.

В качестве еще одного примера закраски рассмотрим тор (рис. 13.23). На первом рисунке (*а*) на поверхность тора нанесен рисунок, на втором (*б*) добавлено диффузное отражение, на третьем (*в*) — зеркальное отражение.



**Рис. 13.23.** Пример закраски тора:  
*а* — с рассеянным светом; *б* — с рассеянным светом и диффузным отражением;  
*в* — с рассеянным светом, диффузным и зеркальным отражением<sup>1</sup>

## 13.6. Создание неровностей на поверхности

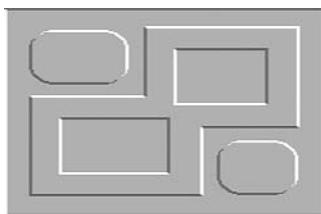
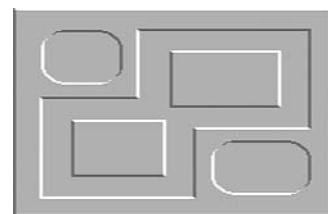
Как сделать рисунок объемным? Проанализировать изменения яркости отдельных участков изображения. Наш мозг делает это автоматически, незаметно для нас. В результате четко определяется, что будет выпуклостью, а что впадиной (рис. 13.24).

Данный подход очень напоминает выдавливание (чеканку). Главное — это правильно наложить яркие и темные участки. Остальное сделает наш мозг.

Но как определить, какие части изображения делать яркими, а какие темными? Очень просто. В большинстве случаев свет падает сверху, поэтому получается, что поверхности ярко освещены сверху, а снизу, наоборот, находятся в тени. Таким образом, если глаз воспринимает светлые и темные области на объекте, то человек видит их как рельеф (рис. 13.25).

Посмотрев на тот же рисунок, только развернутый на 180° (см. рис. 13.24), можно заметить, что он стал полной противоположностью предыдущему. То, что раньше казалось выпуклым, стало вогнутым, и наоборот. А ведь это то же самое

<sup>1</sup> <http://www.ixbt.com>.

Рис. 13.24. Пример с выпуклым рисунком<sup>1</sup>Рис. 13.25. Пример с вогнутым рисунком<sup>2</sup>

изображение. Если же заставить мозг воспринимать свет как идущий снизу, мозг воспроизведет информацию так же, как на первом рисунке.

Для того чтобы поверхность казалась шероховатой, можно попробовать нарисовать на ней рисунок. Но при этом будет казаться, что неровности нарисованы на гладкой поверхности, и получить эффект шероховатости не удастся.

### Рельефное текстурирование

Первый метод основан на использовании рельефных карт (Bump Mapping).

*Рельефная карта* — это обычная текстура, только в отличие от основной, несущей информацию о цвете определенных участков, она несет информацию о неровностях. Самый распространенный способ представить неровности — применить карту высот. *Карта высот* — это текстура в оттенках серого, где яркость каждого пикселя представляет, насколько он выдается над базовой поверхностью (рис. 13.26).

Используя карту высот, можно имитировать неровности практически любой поверхности: дерева, камня, чеканки и т. д. Конечно, у всего есть свои пределы. Используя рельефное текстурирование, нельзя передать крупные впадины и возвышенностей, но вот для имитации неровностей и шероховатостей на поверхности данный метод подходит идеально.

По сути, это логическое продолжение техники просчета по Фонгу, где интерполируется нормаль к поверхности по всему полигону и данный вектор используется для дальнейшего определения яркости соответствующего пикселя. Реализуя его, немного меняется направление вектора нормали, основываясь на информации, содержащейся в карте высот. При изменении положения вектора нормали в конкретной точке полигона меняется яркость текущего пикселя (закон косинуса из теории света).

Для достижения этого существует несколько путей. Сначала необходимо преобразовать информацию о высоте неровностей на карте высот в информацию о величине подстройки вектора нормали, для чего нужно преобразовать высоты с карты в маленькие векторы. Более светлые квадраты соответствуют более выпуклым

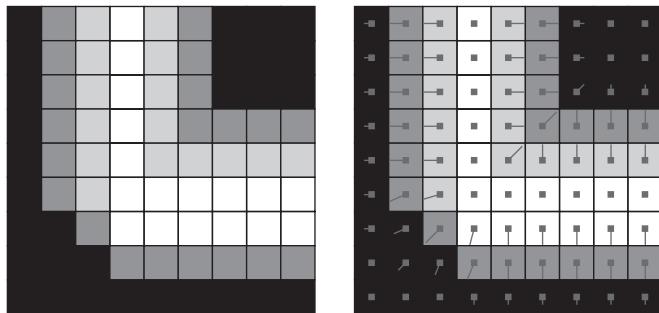
Рис. 13.26. Карта высот<sup>3</sup>

<sup>1</sup> <http://www.ixbt.com>.

<sup>2</sup> Там же.

<sup>3</sup> Там же.

участкам. После этого для каждого пикселя рассчитывается вектор, указывающий направление уклона поверхности (рис. 13.27). Маленькие векторы указывают на уменьшение высоты.



**Рис. 13.27.** Расчет векторов, указывающих направление уклона поверхности

### Метод возмущения нормали

В векторе нормали к настоящей шероховатой поверхности и, следовательно, в направлении отражения есть небольшая случайная составляющая. На этой основе Блинн разработал метод возмущения нормали (Normal Mapping) для построения неровных поверхностей. Строится новая поверхность, которая выглядит шероховатой, путем внесения в направление нормали функции возмущения  $P(x, y)$ :

$$Q'(x, y) = Q(x, y) + P(x, y) \cdot \frac{n}{|n|},$$

где  $Q$  — поверхность,  $Q'$  — новая поверхность,  $n$  — нормаль в точке  $(x, y)$ .

В качестве  $P(x, y)$  можно использовать почти любую функцию, у которой существуют частные производные. Если узор не определяется аналитически, то функция возмущения записывается как двумерная таблица цветов с индексами  $x$  и  $y$ . Промежуточные значения вычисляются билинейной интерполяцией табличных величин, а производные — методом конечных разностей.

В то время как рельефное текстурирование всего лишь изменяет существующую нормаль для точек поверхности, метод возмущения нормали полностью заменяет нормали с помощью выборки их значений из специально подготовленной *карты нормалей* (normal map). Эти карты обычно являются текстурами с сохраненными в них заранее просчитанными значениями нормалей, представленными в виде компонент цвета RGB, в отличие от восьмибитных черно-белых карт высот в бэмп-маппинге.

Данный метод добавления шероховатости к модели, как и предыдущий, сравнительно низкой сложности, без использования большего количества реальной геометрии. Одно из наиболее интересных применений метода — существенное увеличение детализации низкополигональных моделей с помощью карт нормалей, полученных обработкой такой же модели высокой геометрической сложности.

Карты нормалей содержат более подробное описание поверхности по сравнению с рельефным текстурированием и позволяют представить более сложные формы.

Карты нормалей предоставляют более эффективный способ для хранения подробных данных о поверхностях по сравнению с простым использованием большого количества полигонов. Единственное серьезное ограничение карт нормалей в том, что они не очень хорошо подходят для крупных деталей, ведь метод возмущения нормали на самом деле не добавляет полигонов и не изменяет форму объекта, а только создает видимость этого. То есть это всего лишь симуляция деталей на основе расчета освещения на пиксельном уровне, что очень хорошо заметно на крайних полигонах объекта и больших углах наклона поверхности. Поэтому наиболее разумный способ применения метода возмущения нормали состоит в том, чтобы сделать низкополигональную модель достаточно детализированной для того, чтобы сохранялась основная форма объекта, и использовать карты нормалей для добавления более мелких деталей.

Карты нормалей обычно создаются на основе двух версий модели — низко- и высокополигональной. Низкополигональная модель состоит из минимума геометрии, основных форм объекта, а высокополигональная содержит все необходимое для максимальной детализации. Затем они сравниваются друг с другом, разница рассчитывается и сохраняется в текстуре, называемой картой нормалей. При ее создании дополнительно можно использовать и рельефное текстурирование для очень мелких деталей, которые нельзя смоделировать даже в высокополигональной модели (поры кожи, другие мелкие углубления).

Карты нормалей изначально были представлены в виде обычных RGB-текстур, где компоненты цвета R, G и B (от 0 до 1) интерпретируются как координаты X, Y и Z. Каждый тексел в карте нормалей представлен как нормаль точки поверхности.

Метод имеет интересные особенности: эффект шероховатости зависит от масштаба изображаемого объекта. Если размер объекта возрастает в два раза, то вектор нормали увеличивается в четыре раза, а его возмущение — только в два раза. Это приведет к тому, что увеличенный объект будет казаться более гладким. Но масштаб фактуры в перспективном изображении не зависит от перемещения объекта в пространстве по направлению к точке зрения или от нее.

В изображении может появиться лестничный эффект, но можно воспользоваться способом усреднения по площади фактуры или методами устранения лестничного эффекта, основанными на предварительной фильтрации, что приведет к сглаживанию фактуры. Нужно рассчитывать изображение с разрешением, большим, чем у дисплея, а затем отфильтровать или усреднить его и вывести с более низким разрешением экрана.

### **Использование фрактальных поверхностей**

Третий метод построения шероховатости основан на фрактальных поверхностях. Фрактальная поверхность состоит из случайно заданных полигональных поверхностей. С помощью этого метода изображаются природные объекты: камни, деревья, облака.

Для получения полигональной фрактальной поверхности исходный многоугольник рекурсивно разбивается на фрагменты (см. рис. 9.1). Для этого можно, например, случайным образом сместить центр и середины сторон многоугольника, причем исходный, и полученный многоугольники не обязательно должны быть плоскими.

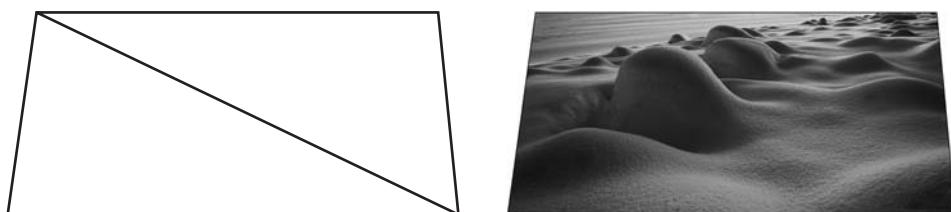
Одно из преимуществ фрактальных поверхностей в том, что их можно разбивать бесконечно и получать любой уровень детализации. Он может зависеть от положения наблюдателя: чем ближе точка зрения, тем с большей степенью детализации изображается поверхность. Затем фрактальная поверхность изображается с помощью любого подходящего алгоритма удаления невидимых поверхностей и любой модели освещения. Но число разбиений растет со скоростью выше линейной, поэтому между количеством разбиений и уровнем детализации должен быть некоторый компромисс. Иначе потребуется слишком много вычислений.

### Использование карт смещения

Наложение карт смещения (Displacement Mapping) является методом добавления деталей к трехмерным объектам. В отличие от рельефного текстурирования, когда картами высот моделируется лишь освещенность точки, но не изменяется ее положение в пространстве, что дает лишь иллюзию увеличения сложности поверхности, карты смещения позволяют получить настоящие сложные 3D-объекты из вершин и полигонов путем изменения их геометрии. Этот метод изменяет положение вершин треугольников, сдвигая их по нормали на величину, которая берется из значений в картах смещения.

*Карта смещения* — обычно черно-белая текстура, значения в которой используются для определения высоты каждой точки поверхности объекта (значения могут храниться как 8- или 16-битные числа). Часто карты смещения используются для создания земной поверхности с холмами и впадинами (в этом случае они называются и картами высот). Так как рельеф местности описывается двумерной картой смещения, его при необходимости относительно легко деформировать, так как это потребует всего лишь модификации карты смещения и визуализации на ее основе поверхности в следующем кадре.

Например, при создании ландшафта с наложением карт смещения исходными были четыре вершины и два полигона. В итоге получился полноценный кусок ландшафта (рис. 13.28).



**Рис. 13.28.** Создание ландшафта с помощью наложения карт смещения

Большим преимуществом наложения карт смещения является не просто возможность добавления деталей к поверхности, а практически полное создание объекта. Берется низкополигональный объект, разбивается на большее количество вершин и полигонов. Вершины, полученные в результате разбиения, затем смещаются по нормали, исходя из значения, прочитанного в карте смещения. В итоге из простого получается сложный трехмерный объект (рис. 13.29).



Рис. 13.29. Пример получения сложного объекта из более простого

Количество треугольников должно быть достаточно большим для того, чтобы передать все детали, задаваемые картой смещений. Наложение карт смещения можно, по существу, считать методом сжатия геометрии — использование карт смещения снижает объем памяти, требуемый для определенной детализации сложной трехмерной модели. Громоздкие геометрические данные замещаются простыми двумерными текстурами смещения, обычно 8- или 16-битными. Это снижает требования к объему памяти и пропускной способности. Еще одно преимущество заключается в том, что применение карт смещения превращает сложные полигональные трехмерные сетки в несколько двумерных текстур, которые проще поддаются обработке.

Но у карт смещения есть и некоторые ограничения, они не могут быть применены во всех ситуациях. Например, гладкие объекты, не содержащие большого количества тонких деталей, будут лучше представлены в виде стандартных полигональных сеток или иных поверхностей более высокого уровня, вроде кривых Безье. С другой стороны, очень сложные модели, такие как деревья или растения, также нелегко представить картами смещения. Многие проблемы и ограничения, присущие картам смещения, совпадают с таковыми у наложения карт нормалей, поскольку эти два метода, по сути, — два разных представления похожей идеи.

### 13.7. Фильтрация текстур

Текстурирование является важнейшим элементом сегодняшних трехмерных приложений, без него многие модели теряют значительную часть своей визуальной привлекательности. Однако процесс нанесения текстур на поверхности

не обходится без артефактов, и необходимы соответствующие методы их подавления.

Частным случаем эффекта ступенчатости, рассмотренного ранее, является эффект ступенчатости текстурированных поверхностей, который, к сожалению, нельзя устраниить методами, описанными выше.

Представьте себе кирпичную стену большого, практически бесконечного размера (рис. 13.30). Допустим, эта стена рисуется на экране при взгляде на нее под небольшим углом. Для достаточно удаленных участков стены размеры кирпичей неизбежно начнут уменьшаться до размера одного пикселя и меньше. Это так называемое оптическое уменьшение текстуры. Между пикселями текстуры начнется «борьба» за обладание пикселями экрана, что приводит к неприятному мельтешению и является одной из разновидностей эффекта ступенчатости. Увеличение экранного разрешения (реального или эффективного) помогает только немного, потому что для достаточно удаленных объектов детали текстур все равно становятся меньше пикселов.

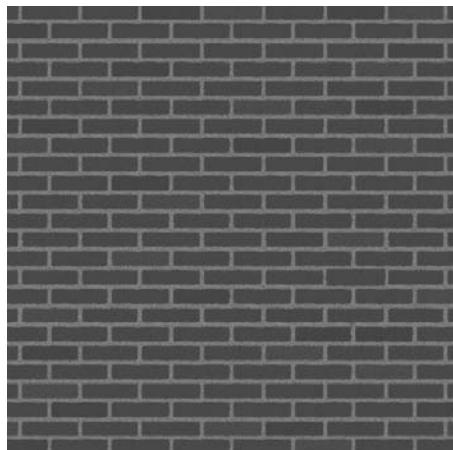
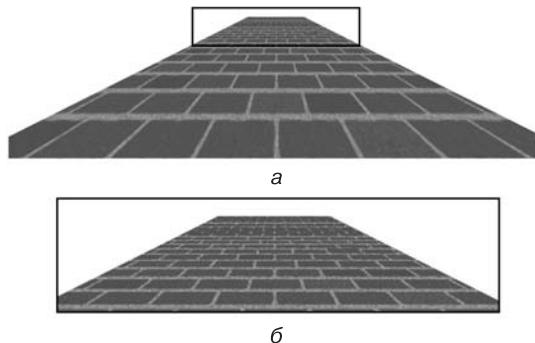


Рис. 13.30. Накладываемая текстура стены

С другой стороны, наиболее близкие к нам части стены занимают большую экранную площадь, и можно наблюдать огромные тексели текстуры. Это называется *оптическим увеличением текстуры*. Хотя данная проблема стоит не так остро, для уменьшения негативного эффекта с ней тоже необходимо бороться.

Для решения проблем текстурирования применяется так называемая *фильтрация текстур*. Если разобраться в процессе рисования трехмерного объекта с наложенной текстурой, можно увидеть, что вычисление цвета пикселя идет как бы наоборот: сначала находится пиксель экрана, куда будет спроектирована некоторая точка объекта, а затем для этой точки находятся все тексели текстуры, попадающие в нее. Выбор текселов текстуры и их комбинация (усреднение) для получения финального цвета пикселя экрана и называется фильтрацией текстуры.

Простейший метод наложения текстур называется *поточечной выборкой* (Single-Point Sampling). Суть его в том, что для каждого пикселя, составляющего полигон, из текстурного изображения выбирается один тексел, ближе всех расположенный к центру отображаемой части фактурного пространства. Данный подход самый неточный, так как цвет пикселя определяют несколько текселов, а выбран был только один. Результатом его применения является появление неровностей. Всякий раз, когда пиксели больше по размеру, чем текселя, наблюдается эффект мерцания. Он имеет место, если часть полигона достаточно удалена от точки наблюдения так, что сразу много текселов накладывается на пространство, занимаемое одним пикселом. Если полигон расположен очень близко к точке наблюдения и текселя больше пикселов по размеру, наблюдается другой тип ухудшения качества изображения — изображение выглядит блочным. В качестве примера рассмотрим наложение текстуры кирпичной стены (см. рис. 13.30) на объект в виде прямоугольной грани, изображенной в перспективной проекции. На проекции такая грань выглядит в виде трапеции (рис. 13.31).



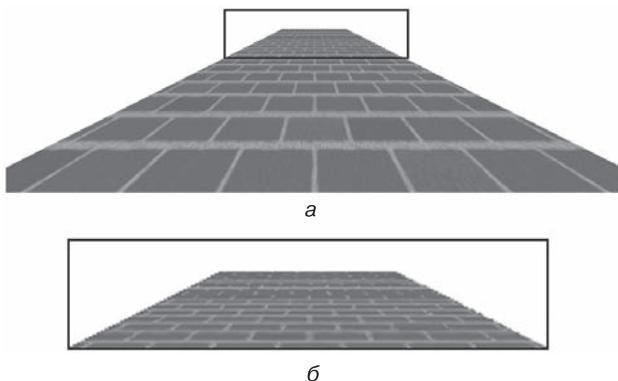
**Рис. 13.31.** Закраска объекта методом поточечной выборки

Нанесенный рисунок (рис. 13.31, *а*) имеет заметные артефакты. Область, выделенная рамкой, приведена в увеличенном виде на рис. 13.31, *б*.

Второй метод — *билинейная фильтрация* (Bilinear Filtering) — состоит в использовании интерполяционной техники. Билинейная фильтрация позволяет устранить блочность текстур при их увеличении и мерцание, возникающее при медленном вращении или движении объекта (приближение/удаление). Для устранения этих эффектов для определения цвета каждого пикселя берется среднее значение цвета четырех смежных текселов, и в результате определяется цвет накладываемой текстуры. Для этого необходимо провести три операции смещивания: сначала смещиваются цвета двух пар текселов, а потом — два полученных цвета.

Главный недостаток билинейной фильтрации в том, что аппроксимация выполняется корректно только для полигонов, которые расположены параллельно экрану. Если полигон развернут под углом (а это 99 % случаев), используется неправильная аппроксимация, так как должен аппроксимироваться эллипс, то есть прямоугольная форма части узора, а не квадратная.

Наложив рисунок на объект с учетом билинейной фильтрации (рис. 13.32), можно тоже заметить ошибки, выражаяющиеся в том, что несколько квадратов сливаются в один. Ошибки, называемые depth aliasing, возникают в результате того, что более отдаленные от точки наблюдения объекты выглядят более маленькими на экране, и если объект двигается и удаляется от точки наблюдения, текстурное изображение, наложенное на уменьшившийся в размерах объект, становится все более и более сжатым. В конечном счете оно становится настолько сжатым, что появляются ошибки визуализации, которые особенно нежелательны в анимации, где такие артефакты во время движения вызывают мерцание и эффект медленного движения в той части изображения, которая должна быть неподвижной и стабильной.



**Рис. 13.32.** Закраска объекта методом билинейной фильтрации:  
а — нанесенный рисунок; б — увеличенная область

Для устранения таких ошибок используется техника, известная как **mip-mapping**. Это наложение текстур, имеющих разный уровень детализации, когда в зависимости от расстояния до точки наблюдения выбирается текстура с необходимой детализацией.

Mip-текстура (mip-map) состоит из набора заранее отфильтрованных и масштабированных изображений. В изображении, связанном с уровнем mip-тар, пиксель представляется в виде среднего четырех пикселов из предыдущего уровня с более высоким разрешением. Отсюда изображение, связанное с каждым уровнем mip-текстуры, в четыре раза меньше по размеру предыдущего mip-тар-уровня (рис. 13.33).

Слева направо расположены mip-тар-уровни 0, 1, 2 и т. д. Чем меньше становится изображение, тем больше теряется деталей, вплоть до приближения к концу, когда не видно ничего, кроме расплывающегося пятна из коричневых пикселов.

Степень, или уровень, детализации (Level of Detail, LOD) используется для определения, какой mip-тар-уровень (или какую степень детализации) следует выбрать для наложения текстуры на объект. LOD должен соответствовать коли-

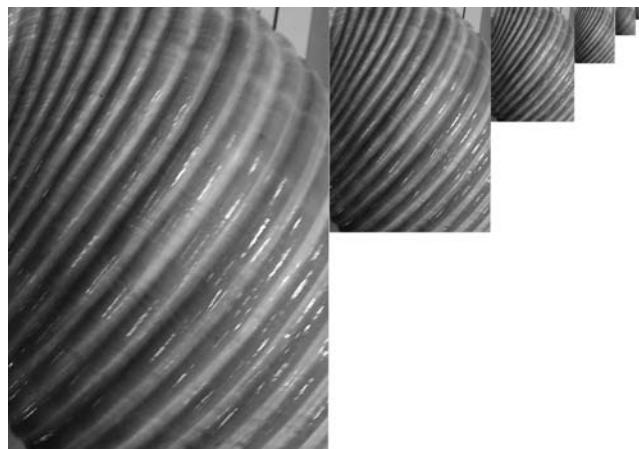


Рис. 13.33. Изображения, связанные с каждым міп-мар-уровнем текстуры ракушки

честву текстелов, накладываемых на пиксель. Например, если текстурирование происходит с соотношением, близким к 1:1, то LOD будет равен нулю и будет использоваться міп-мар-уровень с самым высоким разрешением. Если четыре текстела накладываются на один пиксель, то LOD будет равен единице и будет использоваться следующий міп-мар-уровень с меньшим разрешением. При удалении от точки наблюдения объект имеет более высокое значение LOD.

В то время как міп-текстурирование решает проблему ошибок depth aliasing, его использование может стать причиной появления других артефактов (рис. 13.34, *а*). При удалении объекта все дальше от точки наблюдения происходит переход от низкого міп-мар-уровня к высокому. В момент нахождения объекта в переходном состоянии от одного міп-мар-уровня к другому появляется особый тип ошибок визуализации, известных под названием mip-banding (полосатость, или слоенность), то есть явно различимые границы перехода от одного уровня к другому (рис. 13.34, *б*).

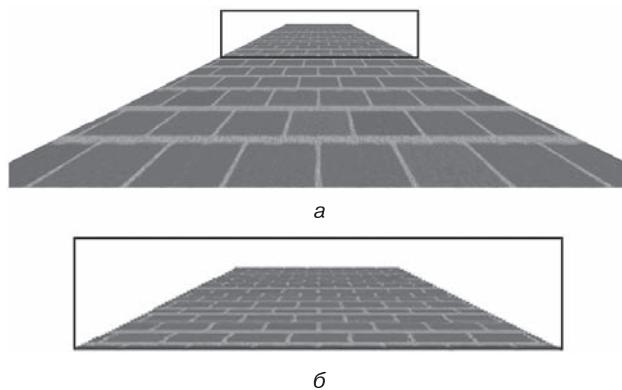
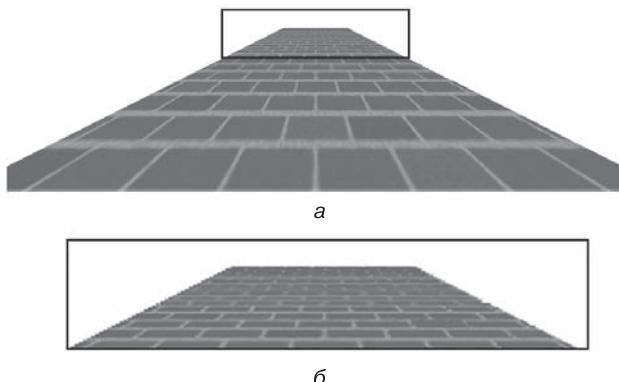


Рис. 13.34. Закраска объекта с использованием міп-текстурирования

Особенно остро данная проблема стоит в анимации из-за того, что человеческий глаз очень чувствителен к смещениям и может легко заметить место резкого перехода между уровнями фильтрации при движении вокруг объекта.

*Трилинейная фильтрация* (Trilinear Filtering) представляет собой третий метод, который удаляет артефакты mipmaping, возникающие при использовании mipmap-текстурирования. При трилинейной фильтрации для определения цвета пикселя берется среднее значение цвета восьми текстелов, по четыре из двух соседних уровней текстур, и в результате семи операций смешивания определяется цвет пикселя. При использовании трилинейной фильтрации возможен вывод на экран текстурированного объекта с плавно выполненными переходами от одного mipmap-уровня к следующему, что достигается за счет определения LOD путем интерполяции двух соседних mipmap-уровней.

На примере все того же прямоугольника уже заметны плавные переходы от одного mipmap-уровня к другому за счет использования трилинейной фильтрации (рис. 13.35).



**Рис. 13.35.** Закраска объекта с использованием mipmap-текстурирования и трилинейной фильтрации: а — нанесенный рисунок; б — увеличенная область

Существует несколько способов генерации mipmap-текстур. Один из них — просто подготовить их заранее, используя графические пакеты типа Adobe Photoshop. Другой способ — генерация «на лету», то есть в процессе выполнения программы. Заранее подготовленные mipmap-текстуры занимают дополнительно 30 % дискового пространства.

Метод mipmap-текстурирования лучше всего работает для полигонов, расположенных параллельно картинной плоскости. Но реально полигоны могут быть ориентированы по-разному, и тогда пиксель будет покрываться не квадратной частью из текстелов, а прямоугольной. Метод mipmap-текстурирования не принимает это во внимание, и в результате наблюдается эффект слишком сильного размытия текстурного изображения. Для решения данной проблемы нужно делать выборку из большего количества текстелов, составляющих текстуру, и выбирать их следует, принимая во внимание «отображенную» форму пикселя в текстурном пространстве. Такой метод называется *анизотропной фильтрацией* (Anisotropic Filtering).

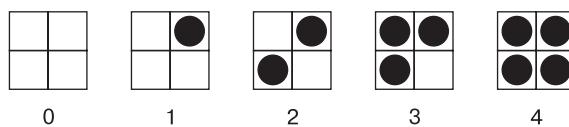
Обычное тир-текстурирование называется isotropic (изотропное, или однородное), потому что всегда фильтруются квадратные области, состоящие из текстелов. Анизотропная фильтрация означает, что используемая форма области из текстелов меняется в зависимости от обстоятельств.

### 13.8. Полутоновые изображения

Аппроксимация полутонами — это метод, в котором используется минимальное количество уровней интенсивности (обычно белый и черный) для улучшения визуального изображения, то есть для получения нескольких оттенков серого или уровней интенсивности.

Этот метод известен давно. Первоначально он использовался при изготовлении шелковых картин и других текстильных изделий. В 1880 году Стефан Хаген изобрел полутоновую печать, с помощью которой можно получать большое количество фотографических полутонов серого, используя чисто двухуровневую среду — черную краску на белой бумаге. Полутоновая печать является клеточным процессом. Для газетных фотографий из-за низкого качества бумаги применяются клетки 50–90 точек на дюйм. Бумага более высокого качества для книг и журналов позволяет использовать клетки 100–300 точек на дюйм. Успех метода полутонов зависит от свойства человеческого глаза быть интегратором, то есть объединять или сглаживать дискретную информацию.

Визуальное представление машинно-генерированных изображений можно улучшить методом конфигурирования. В противоположность полутоновой печати, в которой используются переменные размеры клеток, в данном методе размеры клеток фиксированы. Несколько пикселов объединяются в конфигурации. Происходит ухудшение пространственного разрешения за счет улучшения визуального (рис. 13.36).



**Рис. 13.36.** Конфигурации для четырех клеток

Например, для каждой клетки используются четыре пикселя. Таким образом получается пять уровней серого. В общем случае:

$$n_j = n_{\bar{p}} + 1,$$

где  $n_j$  — количество уровней интенсивности ( $\hat{I}$ ),  $n_{\bar{p}}$  — количество пикселов в клетке.

При выборе конфигураций необходимо проявлять осторожность, так как могут возникнуть нежелательные мелкомасштабные структуры. Например, не нужно применять линейчатые конфигурации (рис. 13.37), так как это приведет к появлению нежелательных горизонтальных и вертикальных линий.

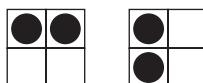


Рис. 13.37. Варианты линейчатых конфигураций

Пример конфигураций для девяти клеток приведен на рис. 13.38.

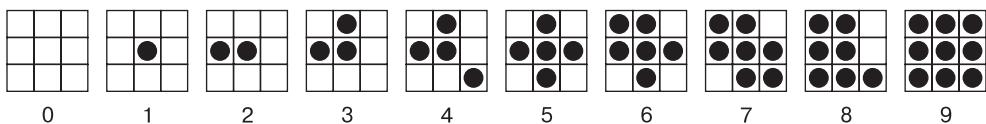


Рис. 13.38. Конфигурация для девяти клеток

Количество уровней интенсивности может возрастать с помощью увеличения размера клетки. Клетки конфигурации необязательно должны быть квадратными. Так как использование конфигурации ведет к потере пространственного разрешения, то это приемлемо в случае, когда разрешение изображения меньше разрешения дисплея. Разработаны методы улучшения визуального изображения при сохранении пространственного разрешения. Например, метод порогового значения для каждого пикселя. Если интенсивность больше некоторой пороговой величины, то пиксель горит, иначе не горит.

Пороговая величина ( $T$ ) обычно определяется по формуле:

$$T = \frac{1}{2} I_{\max}.$$

Если  $\hat{I}(x, y) > T$ , то пиксель горит, иначе не горит.

При простом пороговом методе наблюдается низкое качество. Будут теряться мелкие детали, а изображение освещенного шара будет иметь внутри белый круг. Улучшить изображение помогает метод порога с переносом, где ошибка не отбрасывается, а распределяется на следующий пиксель.

Если  $\hat{I}(x_i, y_i) < T$ , то пиксель не горит, ошибка равна  $\hat{I}(x, y)$  и

$$\hat{I}(x_{i+1}, y_{i+1}) = \hat{I}(x_{i+1}, y_{i+1}) + \text{ош.}$$

Если  $\hat{I}(x_i, y_i) \geq T$ , то пиксель горит, ошибка равна  $\hat{I}(x, y) - 2T$  и

$$\hat{I}(x_{i+1}, y_{i+1}) = \hat{I}(x_{i+1}, y_{i+1}) + \text{ош.}$$

**Пример.** Пусть интенсивность задана в интервале от 0 до 29 (рис. 13.39).  $T = 14,5$ .

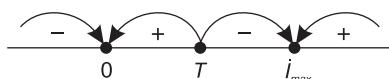


Рис. 13.39. Схема переноса ошибки

Ряд интенсивностей представлен в табл. 13.2.

**Таблица 13.2. Пример закраски простым и пороговым методами**

Значение $i$	11	13	15	17	19	21	23	23	21	18	14	10	6	5	5
Ошибки		(11)	(-5)	(10)	(-2)	(-12)	(9)	(3)	(-3)	(-11)	(7)	(-8)	(2)	(8)	(13)
Простое ограничение	-	-	+	+	+	+	+	+	+	+	-	-	-	-	-
Метод порога с переносом	-	+	-	+	+	-	+	+	+	-	+	-	-	-	+

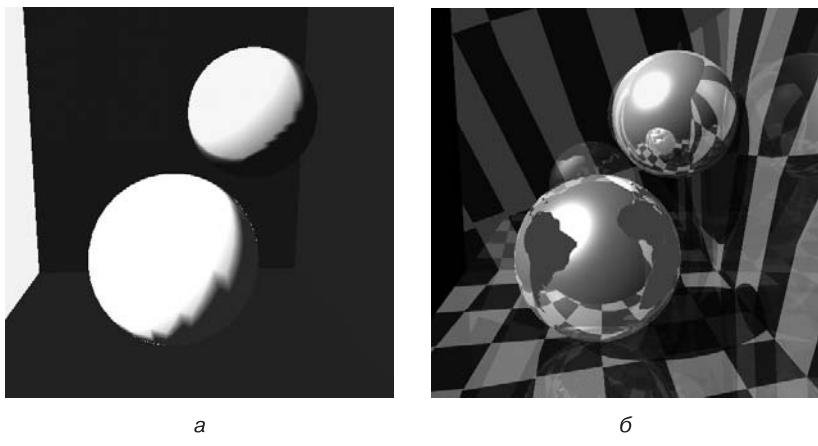
Изображение получается вполне реалистичным и широко используется там, где есть ограничения по количеству цветов.

## Глава 14

# Трассировка лучей

Ранее были рассмотрены проекционные методы получения реалистичного изображения сцены, начиная с описания объектов, получения геометрической модели, изменения объектов и заканчивая построением проекций, освещением объекта и его закраской. Проекционные методы — это методы, в которых синтез изображения выполняется с помощью аффинных преобразований и преобразований проекций. Трехмерная сцена как набор примитивов визуализации (обычно многоугольников, точек, линий и т. п.) трансформируется в двумерный массив, который и отображается на экране монитора. Но чтобы получить хотя бы тень, отбрасываемую объектом на самого себя, алгоритм придется весьма сильно усложнить, причем результат получится не совсем совпадающим с реальной картиной освещения. Когда же дело касается зеркального отражения, прозрачности либо преломления света, эти алгоритмы оказываются слабы.

Однако существует и другой подход, позволяющий получить реалистичное изображение сцены и решить все перечисленные проблемы простым и красивым математическим методом. Он называется *трассировкой лучей*. Данный метод решает также задачи удаления невидимых граней и наложения текстуры. Методы трассировки лучей работают на уровне пикселов выходного изображения, рассчитывая их цвет на основе данных о геометрии сцены и положения виртуальной камеры. На рис. 14.1, *а* показаны два шара, освещенные источником света, а на рис. 14.1, *б* — те же шары после работы метода трассировки лучей. Видны зеркальные блики, зеркальное отражение, нанесение рисунка и тени.



**Рис. 14.1.** Получение реалистичного изображения методом трассировки лучей

Классический метод трассировки лучей (Ray Tracing) был предложен Артуром Аппелем еще в 1968 году и дополнен алгоритмом общей рекурсии, разработанным в 1980-х годах на базе работ Уиттеда и Кея. Понадобилось почти 12 лет эволюции вычислительных систем, прежде чем этот алгоритм стал доступен для широкого применения в практических приложениях. Суть метода заключается в отслеживании траекторий лучей и расчете взаимодействий с лежащими на траекториях объектами от момента испускания лучей источником света до момента попадания в камеру. Трассировка лучей — первый метод расчета глобального освещения, рассматривающий освещение, затенение (расчет тени), многократные отражения и преломления.

## 14.1. Метод прямой трассировки

Предполагается, что из точек поверхности излучающих объектов исходят лучи. Такие лучи будут первичными — они освещают все остальное. Считается, что луч света распространяется прямолинейно до тех пор, пока не встретится точка отражающей поверхности или граница среды преломления.

Затем проверяется ориентация каждой точки относительно источника. Если она лежит на стороне объекта, обращенной в противоположную от источника сторону, точка исключается из расчетов освещенности. Для всех остальных точек вычисляется освещенность с помощью уже рассмотренной ранее модели освещения. Если объект не является отражающим или прозрачным, то есть поверхность объекта только диффузно рассеивает свет, траектория луча на этой точке заканчивается. Если же поверхность объекта обладает свойством зеркального отражения и/или преломления, из точки строятся новые лучи, направления которых совершенно точно определяются законами отражения и преломления.

От источников излучения в различных направлениях исходит бесчисленное множество первичных лучей. Одни лучи уходят в свободное пространство, а другие (их также бесчисленное множество) попадают на различные объекты. Если луч попадает на прозрачный объект, то, преломляясь, он идет дальше, при этом некоторая часть световой энергии поглощается. Если же на пути луча встречается зеркальная отражающая поверхность, он также изменяет направление, а часть световой энергии поглощается. Если объект зеркальный и одновременно прозрачный (например, обычное стекло), то будут уже два луча — в этом случае говорят, что луч расщепляется.

В результате действия первичных лучей на объекты возникают вторичные лучи. Бесчисленное множество вторичных лучей уходит в свободное пространство, а некоторые из них попадают на другие объекты. Так, многократно отражаясь и преломляясь, отдельные световые лучи приходят в точку наблюдения — глаз человека. Очевидно, что в точку наблюдения может попасть и часть первичных лучей непосредственно от источников излучения. Таким образом, изображение сцены формируется некоторым множеством световых лучей.

Цвет отдельных точек изображения определяется цветом и интенсивностью первичных лучей источников, а также интенсивностью и изменением световой энергии в объектах, встретившихся на пути этих лучей.

Но реализация такой лучевой модели формирования изображения является трудоемкой. Необходимо предусмотреть перебор всех первичных лучей и определить те из них, которые попадают в объекты и в камеру. Затем следует выполнить перебор всех вторичных лучей и также учесть только те, которые попадают в объекты и в камеру. И так далее. Таким образом работает прямая трассировка лучей. Практическая ценность данного метода вызывает сомнения, так как почти невозможно учесть бесконечное множество лучей, идущих во все стороны. Даже если каким-то образом свести это к конечному числу операций (например, разделить всю сферу представлений на угловые секторы и оперировать уже не бесконечно тонкими линиями, а секторами), все равно остается главный недостаток метода — много лишних операций, связанных с расчетом лучей, которые затем не используются.

## 14.2. Метод обратной трассировки

Согласно данному методу отслеживание лучей производится не от источников света, а в обратном направлении — от точки наблюдения. Поэтому учитываются только те лучи, которые вносят вклад в формирование изображения. Это очень изящный способ решить массу проблем, возникающих при прямой трассировке, а сам метод отличается простотой и понятностью.

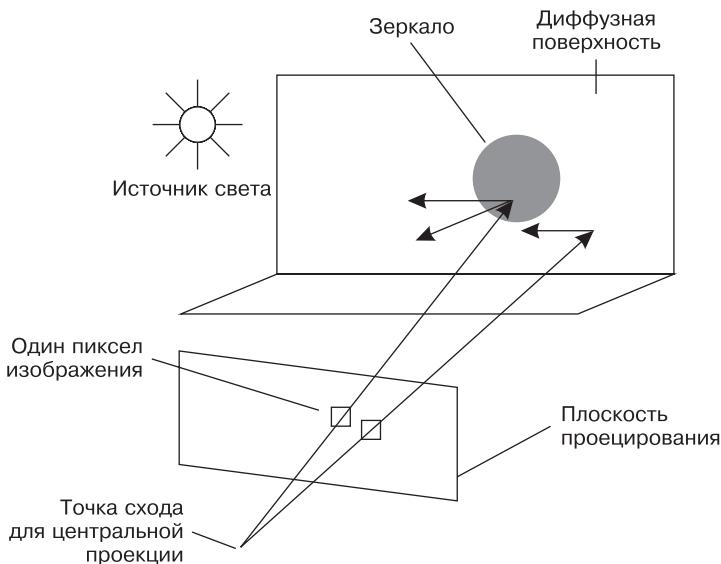
Представим, что плоскость проецирования разбита на множество пикселов. Рассмотрим центральную проекцию с точкой схода на некотором расстоянии от плоскости проецирования. При проведении прямой линии из точки схода через середину пикселя плоскости проецирования получается первичный луч обратной трассировки. Если этот луч попадает в один или несколько объектов сцены, то выбирается ближайшая точка пересечения (рис. 14.2). Для определения цвета пикселя изображения учитываются свойства объекта и источника света.

Если пересечений с объектами нет, а есть пересечение только с плоскостью отсечения, значит, луч выходит за пределы видимой части сцены, и соответствующему пикселу видового окна присваивается цвет фона.

В точке пересечения луча с объектом строятся три вторичных луча: один в направлении отражения, второй в направлении источника света, третий в направлении преломления прозрачной поверхности.

Если объект зеркальный (хотя бы частично), то вторичным лучом является луч падения, при этом лучом отражения считается предыдущий, первичный трассируемый луч. Выше было рассмотрено зеркальное отражение и получены формулы для вектора отраженного луча по заданным векторам нормали и луча падения. Но сейчас нужно решить обратную задачу — по вектору отраженного луча найти вектор падающего луча. Для этого используется та же формула зеркального отражения, но необходимый вектор луча падения определяется как отраженный луч. То есть отражение наоборот.

Для идеального зеркала достаточно затем проследить очередную точку пересечения вторичного луча с некоторым объектом. Неидеальное зеркало усложняет трассировку — нужно проследить не один, а множество падающих лучей, учитывая вклад излучения от других видимых из данной точки объектов.



**Рис. 14.2.** Схема обратной трассировки лучей

Если объект прозрачный, то необходимо построить новый луч, который при преломлении давал бы предыдущий трассируемый луч. Здесь также можно воспользоваться обратимостью, справедливой и для преломления.

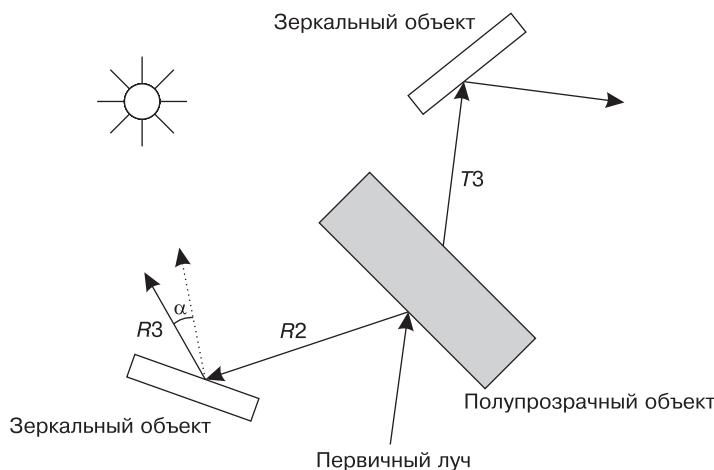
Если объект обладает свойствами диффузного отражения и преломления, то в общем случае, как и для неидеального зеркала, необходимо трассировать лучи, приходящие от всех имеющихся объектов. Для диффузного отражения интенсивность отраженного света, как известно, пропорциональна косинусу угла между вектором луча от источника света и нормалью. Здесь источником света может выступать любой видимый из данной точки объект, способный передавать световую энергию.

Если текущий луч обратной трассировки не пересекает какой-либо объект, а уходит в свободное пространство, то его рассмотрение на этом заканчивается (рис. 14.3).

Обратная трассировка лучей хоть и сокращает перебор, но не позволяет избавиться от бесконечного количества анализируемых лучей. В самом деле, данный метод позволяет сразу получить для каждой точки изображения единственный первичный луч обратной трассировки. Однако вторичных лучей отражения уже может быть бесконечно много. Так, например, если объект может отражать свет от любого другого объекта и если эти другие объекты имеют достаточно большие размеры, то нужно учитывать все точки излучающих объектов для построения соответствующих лучей.

Рассмотрим основные шаги метода трассировки лучей.

1. Испускается воображаемый луч из глаза наблюдателя через некоторый пиксель экрана, и отслеживается его путь, пока он не пересечет объект.



**Рис. 14.3.** Пример обратной трассировки лучей

2. Из первой точки пересечения луча с объектом или его сферической оболочкой испускается отраженный луч. Если поверхность непрозрачная, преломленные лучи не рисуются. Обозначается луч тени от точки пересечения к источнику света. Так как этот луч не пересекает другую непрозрачную поверхность, источник света непосредственно влияет на интенсивность освещения в данной точке.
3. Если отраженный луч пересекает другой объект, например полупрозрачную сферу, которая отражает и пропускает свет, то испускаются отраженный и преломленный лучи вместе с теневым лучом, идущим к источнику света. Пропущенный луч меняет свое направление до и после вхождения в сферу в соответствии с эффектом преломления.
4. Если точка, в которой луч пересекает сферу, не будет прямо освещена источником, потому что путь теневого луча преграждается непрозрачной поверхностью, то учитываются другие источники света, и теневые лучи пускаются в каждый из них.
5. Влияние всех лучей суммируется, и результат определяет RGB-значение данной точки.

При практической реализации метода обратной трассировки вводят ограничения. Одни из них необходимы, чтобы можно было в принципе решить задачу синтеза изображения, а другие позволяют значительно повысить быстродействие трассировки. Они состоят в следующем.

1. Выделяются объекты, называемые источниками света. Источники света могут только излучать свет, но не могут его отражать или преломлять. Рассматриваются только точечные источники света.
2. Свойства отражающих поверхностей описываются суммой двух компонент — диффузной и зеркальной.

3. Зеркальность тоже описывается двумя составляющими. Первая учитывает отражения от других объектов, не являющихся источниками света. Строится только один зеркальный луч для дальнейшей трассировки. Вторая компонента означает световые блики от источников света. Для этого направляются лучи на все источники света и определяются углы, образуемые этими лучами с зеркально отраженным лучом обратной трассировки. При зеркальном отражении цвет точки поверхности определяется цветом того, что отражается. В простейшем случае зеркало не имеет собственного цвета поверхности.
4. При диффузном отражении учитываются только лучи от источников света. Лучи от зеркально отражающих поверхностей игнорируются. Если луч, направленный на данный источник света, закрывается другим объектом, значит, данная точка объекта находится в тени. При диффузном отражении цвет освещенной точки поверхности определяется собственным цветом поверхности и цветом источников света.
5. Для прозрачных объектов обычно не учитывается зависимость коэффициента преломления от длины волны. Иногда прозрачность вообще моделируют без преломления, то есть преломленный и падающий лучи совпадают.
6. Для учета освещенности рассеиваемым светом вводится фоновая составляющая.
7. Для завершения трассировки вводят некоторое пороговое значение освещенности, которое уже не должно вносить вклад в результирующий цвет, либо ограничивают количество итераций.

Согласно модели закраски цвет некоторой точки объекта определяется суммарной интенсивностью:

$$I(\lambda) = Ka(\lambda)Ia(\lambda) + Kd Id(\lambda)C(\lambda) + Ks Is(\lambda) + Kt It(\lambda),$$

где  $\lambda$  — длина волны;

$C(\lambda)$  — заданный исходный цвет точки объекта;

$Ka, Kd, Ks, Kt$  — коэффициенты рассеянного света, диффузного и зеркального отражения, прозрачности;

$Ia$  — интенсивность рассеянного света;

$Id$  — интенсивность диффузного отражения;

$Is$  — интенсивность зеркального отражения;

$It$  — интенсивность излучения, проходящего по преломленному лучу.

Достоинства обратной трассировки лучей такие.

- ❑ Универсальность метода, его применимость для синтеза изображений достаточно сложных пространственных схем. Он воплощает многие законы геометрической оптики. Просто реализуются разнообразные проекции.
- ❑ Позволяет получить достаточно реалистичные изображения, даже если ограничиться только начальными лучами. Так, учет первичных лучей дает удаление невидимых линий. Трассировка уже одного-двух вторичных лучей дает тени, зеркальность и прозрачность.

- Все преобразования координат линейны, поэтому достаточно просто работать с текстурами.
- Для одного пикселя растрового изображения можно трассировать несколько близко расположенных лучей, а потом усреднить их цвет для устранения эффекта ступенчатости.
- Поскольку расчет отдельной точки изображения выполняется независимо от других точек, это может быть эффективно использовано при реализации данного метода в параллельных вычислительных системах, в которых лучи могут трассироваться одновременно.

Недостатки указанного метода следующие.

- Проблемы с моделированием диффузного отражения и преломления.
- Для каждой точки изображения необходимо выполнять много вычислительных операций. Трассировка лучей относится к числу самых медленных алгоритмов синтеза изображений.

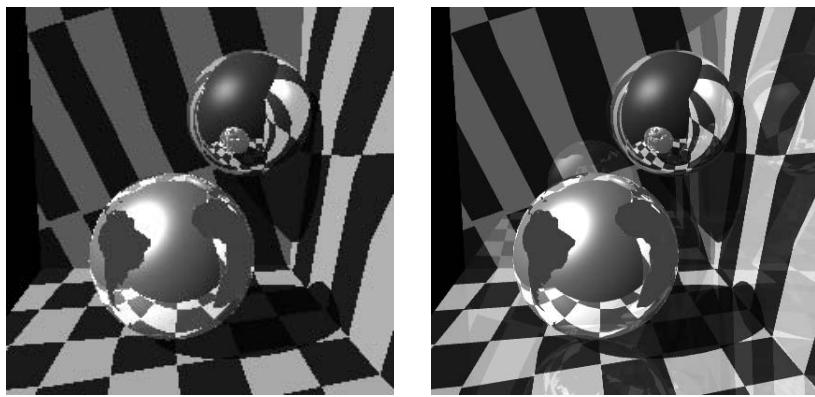
Результат работы алгоритма трассировки лучей при высоком значении количества уровней трассировки почти не уступает по качеству фотографиям реальных объектов. Однако, как и в случае с детализированной поверхностью модели, увеличение уровней трассировки ведет к резкому замедлению алгоритма. Причем чем больше уровней используется, тем меньшее улучшение изображения дает дальнейшее увеличение их количества, а время, затрачиваемое на прохождение лучами каждого нового уровня, в среднем остается постоянным.

Кроме того, количество необходимых уровней напрямую зависит от свойств поверхностей объекта. Если объект непрозрачный и его поверхность слабо отражает свет, то двух уровней трассировки будет вполне достаточно для построения качественного изображения. Если же объектов в сцене несколько, они прозрачные и имеют зеркальную поверхность, то количество уровней в алгоритме желательно многократно увеличить.

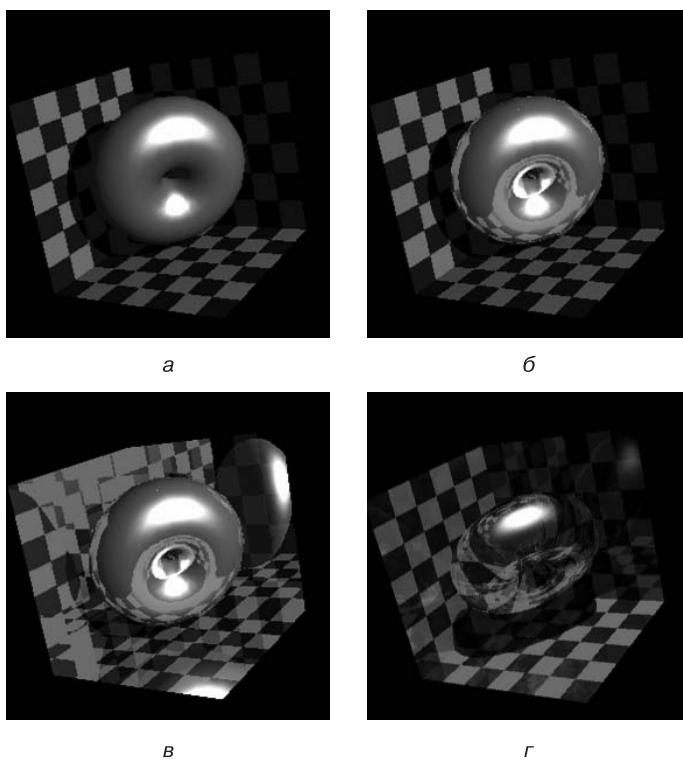
После получения изображения методом трассировки лучей для улучшения его качества можно использовать методы фильтрации. На рис. 14.4, *a* хорошо заметна ступенчатость, которая становится почти невидимой после применения методов фильтрации (рис. 14.4, *б*).

Фильтрация применяется для удаления или уменьшения зазубренных линий. Для этого используются различные методы сглаживания.

Принцип вычисления усредненных величин, или получение окончатательного цвета пикселя по нескольким промежуточным значениям, имеет гораздо более глубокий смысл, чем просто сглаживание. При испускании одного первичного луча на пиксель делается попытка описать вычисленными интенсивностями в отдельных точках пересечения лучей и объектов, которые являются дискретными величинами, диапазон непрерывных значений реальной освещенности. Если это и можно сделать корректно, то только при соблюдении определенных правил. Такие правила используют особенности свойств зрения человека, а именно: зрительное восприятие человека основано на принципе интегрирования, или смешивания,

*а**б*

**Рис. 14.4.** Пример устранения ступенчатости (блочности) в изображении, полученном методом трассировки лучей

*в**г*

**Рис. 14.5.** Пример получения реалистичного изображения сцены методом трассировки лучей:

- а* — зеркальный блеск на торе и отбрасываемая тень;
- б* — полная зеркальность тора и отбрасываемая тень;
- в* — зеркальность тора и стен; *г* — зеркальность тора и прозрачность

цветов близко расположенных пространственных точек. Точнее, человек всегда зрительно воспринимает некоторую пространственную область, а не отдельные точки, и степень смешивания тем больше, чем дальше расположен зрительный объект.

Поэтому усреднение цвета пикселя по нескольким промежуточным значениям очень важно. Его использование дает не только сглаживание, но и повышает общее качество изображения, поскольку делает его естественным для восприятия. Но повышая качество результирующего изображения, нельзя избежать потерь времени при его построении. Чем больше размер ступенчатости, тем больше времени требуется для выравнивания цветов. Между качеством и временем обработки нет прямой линейной зависимости. Более того, при увеличении размера области сглаживания до некоторого значения наблюдается улучшение картинки, после чего появляется размытость и качество резко снижается.

Еще один пример получения реалистичного изображения сцены методом трассировки лучей показан на рис. 14.5. Особый интерес представляет моделирование зеркальности.

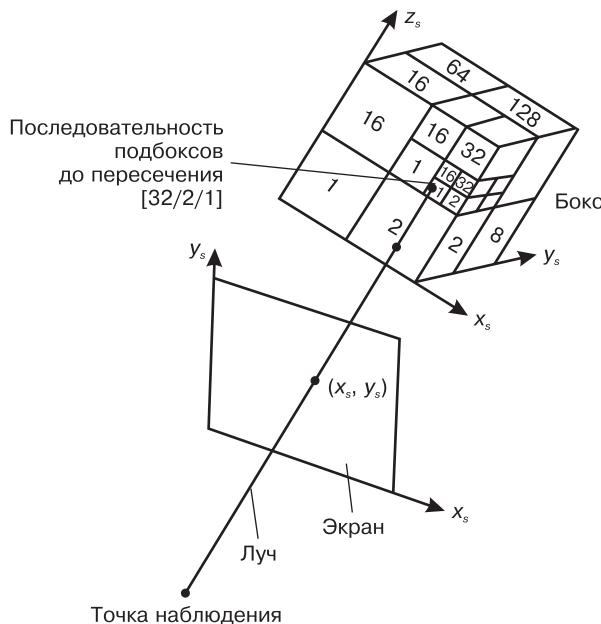
### 14.3. Дискретная трассировка лучей в октантных деревьях

Применяется для воксельных моделей. Основной принцип метода следующий (рис. 14.6): луч отслеживается из точки наблюдения, пересекает плоскость экрана в точке  $(x_s, y_s)$  и ударяется в бокс, содержащий дискретную информацию об объекте. Бокс разделен на подбоксы, пронумерованные от  $2^0$  до  $2^7$ , которые далее рекурсивно делятся на более мелкие подбоксы, формируя октантное дерево. (Все ячейки пространства, отличные от вокселя, именуются подбоксами или боксами, и только элементарные неделимые элементы объема называются вокселями.)

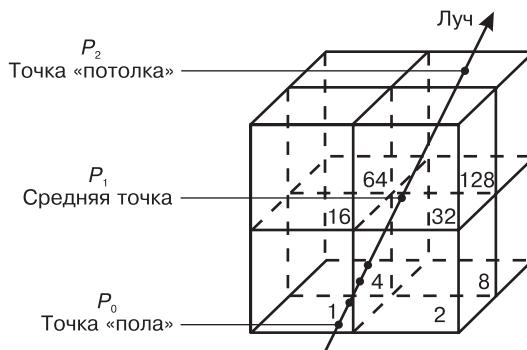
В основном алгоритм работает, как многие методы обратной трассировки лучей. Лучи испускаются из точки наблюдения через пиксели на экране и отслеживаются до момента пересечения с объектом. В этой точке вычисляется цвет пикселя или порождаются вторичные отраженные либо преломленные лучи, если таковые имеются. Точка пересечения луча и объекта находится следующим образом.

1. Ищется точка пересечения луча и бокса, содержащего объект (вычисляются точки входа и выхода луча из бокса).
2. Используя информацию о точках пересечения, отслеживается луч внутри бокса и находится первый пересеченный воксель поверхности по ходу луча.
3. Используя информацию о свойствах поверхности, содержащуюся в вокселе (цвет, нормаль и т. д.), вычисляется цвет пикселя или испускаются вторичные лучи.

Для поиска точки пересечения луч входит в бокс в точке «пола»  $P_0$ , проходит через среднюю точку  $P_1$  и покидает бокс в точке «потолка»  $P_2$  (рис. 14.7). Важно, чтобы три точки располагались на трех параллельных плоскостях, две из которых содержат противоположные грани бокса, а третья делит бокс на две равные половины. Точки  $P_0$  и  $P_2$  необязательно будут принадлежать граням бокса, в то время как средняя точка  $P_1$  однозначно будет находиться внутри бокса.



**Рис. 14.6.** Метод дискретной трассировки лучей в октантных деревьях<sup>1</sup>



**Рис. 14.7.** Метод поиска точки пересечения<sup>2</sup>

Всего могут быть три варианта выбора для плоскостей, содержащих точки «пола» и «потолка»: плоскости могут быть перпендикулярны одной из трех координатных осей  $X$ ,  $Y$  или  $Z$ . Пусть точка на луче описывается в системе координат бокса следующим уравнением:

$$P = S_0 + t \cdot (dx, dy, dz),$$

где  $S_0$  — начальная точка луча,  $(dx, dy, dz)$  — вектор направления луча,  $t$  — скалярный параметр.

<sup>1</sup> <http://graphics.cs.msu.ru/library/>.

<sup>2</sup> Там же.

Тогда осью главного направления луча (ОГНЛ) называется ось, составляющая которой в векторе направления луча преобладает над другими. То есть:

$$\text{ОГНЛ} = \begin{cases} X, & \text{если } dx = \max(dx, dy, dz); \\ Y, & \text{если } dy = \max(dx, dy, dz); \\ Z, & \text{если } dz = \max(dx, dy, dz). \end{cases}$$

Методика выбора главного направления совпадает с использованной в целочисленном алгоритме Брезенхема и служит тем же целям: избежать появления разрывов во время дискретизации прямой луча.

Плоскости «пола» и «потолка» выбираются перпендикулярными осями главного направления луча.

Для определения точки пересечения луча и модели используется двоичный рекурсивный поиск вдоль луча между плоскостями «пола» и «потолка». Процедура поиска должна получить следующие входные параметры:

- ОГНЛ;
- двумерные координаты точек «пола» и «потолка» в соответствующей системе координат;
- текущий узел дерева.

Важно отметить, что при двоичном поиске точки пересечения используются двумерные координаты точек, так как третья координата точки на луче (координата вдоль ОГНЛ) становится необходимой лишь непосредственно во время отслеживания луча.

Несмотря на то что алгоритм трассировки луча очень прост и использует всего несколько целочисленных и логических операций, он требует вычисления точек «пола» и «потолка» для каждого пикселя изображения, чтобы инициализировать процесс трассировки.

Для сокращения вычислительной стоимости используется алгоритм интерполяции точек «пола» и «потолка» для всех пикселов изображения. В случае параллельной проекции соседние значения «пола» и «потолка» отличаются только на фиксированное приращение, так что интерполяция выражается лишь в добавлении констант к координатам. В таком случае получение значений точек «пола» и «потолка» вырождается в четыре операции целочисленного сложения и четыре операции битового сдвига (для поддержания точности без использования плавающей точки).

По сравнению с традиционными полигональными методиками представления поверхностей дискретная трассировка лучей имеет следующие преимущества:

- количество информации и скорость визуализации почти не зависят от сложности объектов;
- нет необходимости в специальных алгоритмах нанесения текстур, так как каждый воксель имеет собственный цвет;
- высокая точность определения нормалей, так как они сопоставлены каждому вокселю;

- высокая точность определения точек пересечения;
- есть возможность непосредственной визуализации наборов данных с измерительных устройств без преобразования в сложные полигональные модели;
- простота реализации алгоритмов определения пересечения двух объектов (например, для CAD/CAM-систем).

Предложенный алгоритм обладает следующими дополнительными преимуществами по сравнению с базовыми методами дискретной трассировки лучей.

- Требуется значительно меньшее количество данных для представления моделей, так как хранится и обрабатывается только информация о поверхности. Увеличение разрешения модели в два раза по каждой из координатных осей приводит к росту объема данных в модели всего в четыре раза (в противоположность однородному представлению воксельного пространства, где увеличение разрешения в два раза приводит к восьмикратному росту объема информации).

- Более высокая скорость.

Представленный метод визуализации комбинирует преимущества дискретного представления моделей в виде воксельных пространств и широко применяемую в системах трассировки лучей методику неоднородного разбиения пространства. Использование октантных деревьев позволяет на порядок сократить объемы памяти, требуемые для хранения моделей. Применение в алгоритме главным образом целочисленных операций существенно упрощает реализацию на аппаратном уровне.

# Глава 15

## Цвет в компьютерной графике

При получении реалистичного изображения трехмерного объекта учитывается не только свет, но и цвет самого объекта. Об особенностях восприятия информации глазом человека уже говорилось ранее. Цвет предмета зависит не только от самого предмета, но также и от источника света, его освещдающего.

Зрительная система человека воспринимает электромагнитную энергию с длинами волн от 400 до 700 нм как видимый свет.

### 15.1. Ахроматический и хроматический цвета

Источник или объект является ахроматическим, если наблюдаемый свет содержит все видимые длины волн в примерно равных количествах (рис. 15.1, *а*). Ахроматический источник кажется белым, а свет от него — белым, черным или серым. Белыми выглядят объекты, ахроматически отражающие более 80 % света белого источника, а черными — менее 3 %. Промежуточные значения дают различные оттенки серого цвета. Ахроматический свет характеризуется интенсивностью (яркостью).

Свет называется хроматическим, если он содержит длины волн в произвольных неравных количествах. Если длины волн сконцентрированы у левого края видимого спектра, то свет кажется красным, если у правого — синим. Между ними располагаются другие цвета спектра (рис. 15.1, *б*).

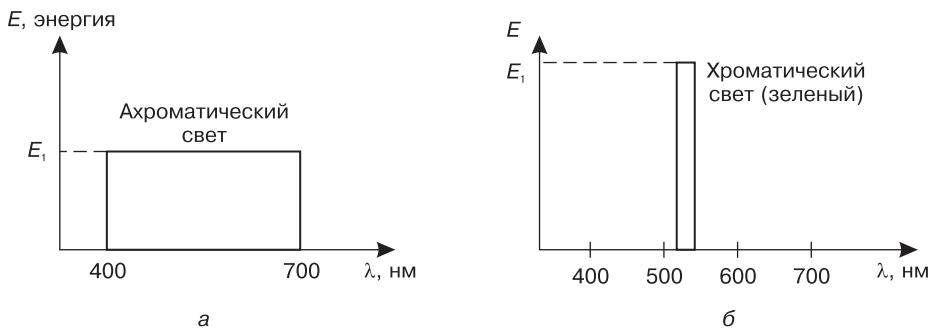


Рис. 15.1. Ахроматический (*а*) и хроматический (*б*) цвета

Но сама по себе электромагнитная энергия определенной длины волны не имеет никакого цвета. Ощущение цвета возникает в результате преобразования физических явлений в глазу или мозге человека. Объект кажется цветным, если он отражает или пропускает свет лишь в узком диапазоне длин волн и поглощает все остальные.

Существуют две системы характеристик цвета — психофизиологическая и физическая. Первая подходит к определению цвета с точки зрения восприятия его человеком, вторая — на базе физических законов.

Психофизиологическое представление света определяется следующими параметрами.

- **Цветовой тон.** Позволяет различать цвета, то есть человек может отличать красный цвет от зеленого.
- **Насыщенность.** Определяет степень ослабления (разбавления) данного цвета белым и позволяет отличать розовый цвет от красного, голубой от синего и т. д. Чистый цвет имеет насыщенность 100 %, и она уменьшается по мере добавления белого. Насыщенность ахроматического цвета составляет 0 %.
- **Светлота.** Это интенсивность, которая не зависит от цветового тона и насыщенности. Ноль определяет черный цвет, более высокие значения характеризуют более яркие оттенки.

Физические определяющие цвета следующие.

- **Доминирующая длина волны.** Определяет монохроматический цвет. Например, при длине волны ( $\lambda$ ) 520 нм (см. рис. 15.1, б) получается зеленый цвет.
- **Чистота.** Характеризует насыщенность цвета и определяется соотношением количеств энергии чистого цвета  $E_2$  и белого  $E_1$  (рис. 15.2). Если  $E_1$  стремится к нулю, то чистота — к 100 %, если  $E_1$  стремится к  $E_2$ , то свет — к белому и чистота — к нулю.
- **Яркость.** Пропорциональна энергии света и рассматривается как интенсивность на единицу площади. Для ахроматического света яркость есть интенсивность.

Существует много систем определения цвета. Так, художники используют другие характеристики цвета, учитывающие их опыт работы с красками:

- разбелы;
- оттенки;
- тона.

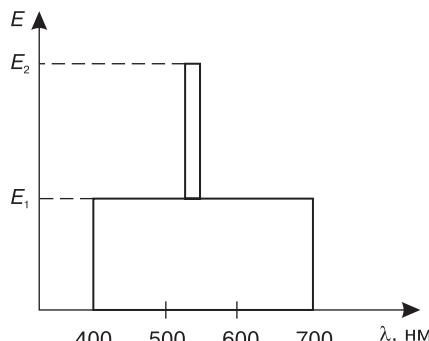


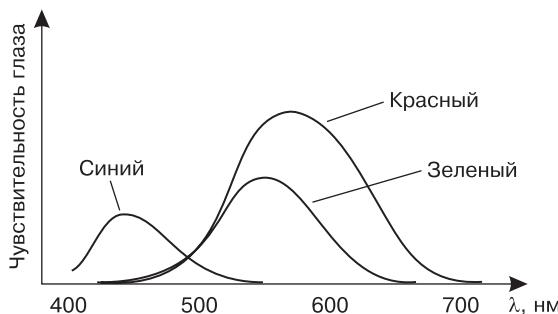
Рис. 15.2. Определение чистоты света



Рис. 15.3. Характеристики цвета

Разбелы получаются при добавлении в чистый цвет белого, оттенки — черного, тона — и черного, и белого (рис. 15.3).

Обычно встречаются не чистые монохроматические цвета, а их сочетания. В основе трехкомпонентной теории света лежит предположение о том, что в сетчатке глаза есть три типа чувствительных к свету колбочек, которые воспринимают соответственно красный, зеленый и синий цвета. Относительная чувствительность глаза максимальна для зеленого цвета и минимальна для синего (рис. 15.4). Если на все три типа колбочек воздействует одинаковый уровень энергетической яркости (энергия в единицу времени), то свет кажется белым.



**Рис. 15.4.** Относительная чувствительность глаза для трех цветов

## 15.2. Цветовые модели

В основе получения цветного изображения на компьютере лежат три цвета: красный, зеленый и синий, которые дают возможность воспроизвести большинство цветов. Большинство, но не все. Цвета, отображаемые монитором, не являются абсолютно чистыми, поэтому и все их оттенки не могут быть воспроизведены точно.

Более того, яркостный диапазон мониторов сильно ограничен. Человеческий глаз в состоянии различать гораздо больше градаций яркости. Максимальная яркость монитора едва ли соответствует и половине максимальной яркости, которую способен различить наш глаз. Это часто может привести к сложностям при отображении сцен из реального мира, которые содержат широкие вариации яркости (например, фотография пейзажа с фрагментом неба и участками земли, находящимися в полной тени).

При моделировании света на компьютере все три цвета обрабатываются отдельно, за исключением каких-либо нестандартных ситуаций, когда цвета не влияют друг на друга. Иногда полноцветные изображения получают путем последовательного просчета красной, зеленой и синей составляющих и их дальнейшего комбинирования.

Обычно компьютеры оперируют со светом в виде величин, определяющих количество содержащихся в нем красного, зеленого и синего цветов. Например, белый — это равное количество всех трех цветов, желтый — равное количество крас-

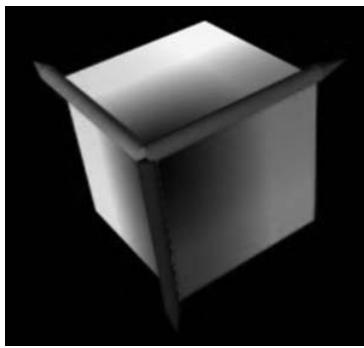


Рис. 15.5. RGB-модель

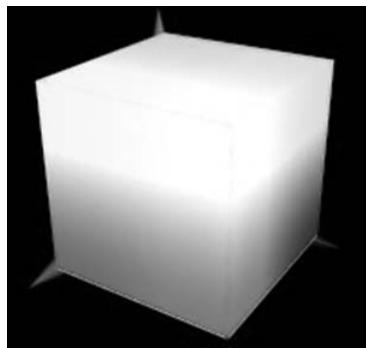


Рис. 15.6. CMY-модель

ного и зеленого и полное отсутствие синего. Все цветовые оттенки можно визуально представить в виде куба, где по осям координат отложены соответствующие величины трех исходных цветов. Это и есть трехцветная световая модель RGB (рис. 15.5).

Отражающие же устройства используют другую модель — CMY (рис. 15.6).

### Системы смешивания основных цветов

Существуют две системы смешивания цветов:

- *аддитивная* (RGB) — красный (Red), зеленый (Green), синий (Blue) цвета (рис. 15.7, а);
- *субтрактивная* (CMY) — голубой (Cyan), пурпурный (Magenta), желтый (Yellow) цвета (рис. 15.7, б).

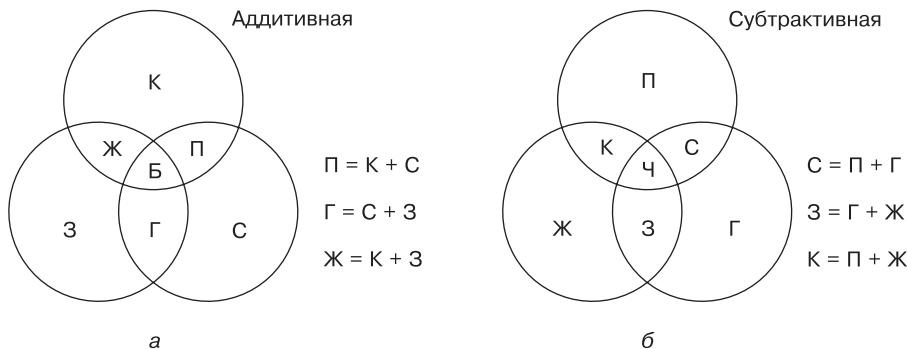


Рис. 15.7. Аддитивная (а) и субтрактивная (б) системы смешивания цветов

Цвета одной системы являются дополнением к другой. Дополнительный цвет — это разность белого и данного цвета. Так, голубой получается вычитанием красного из белого, пурпурный — зеленого из белого, желтый — синего из белого.

Аддитивная цветовая система удобна для светящихся поверхностей (экраны ЭЛТ, цветовые лампы). Субтрактивная цветовая система используется для отражающих поверхностей (цветные печатные устройства, типографские краски, несветящиеся экраны).

Уравнение монохроматического цвета:

$$C = rR + gG + bB,$$

где  $C$  – цвет;

$R, G, B$  – три потока света;

$r, g, b$  – относительные количества потоков света (от 0 до 1).

Соотношение между двумя цветовыми системами можно выразить математически:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}; \quad \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}.$$

Цветовые пространства RGB и CMY трехмерны, и условно их можно изобразить в виде куба (рис. 15.8).

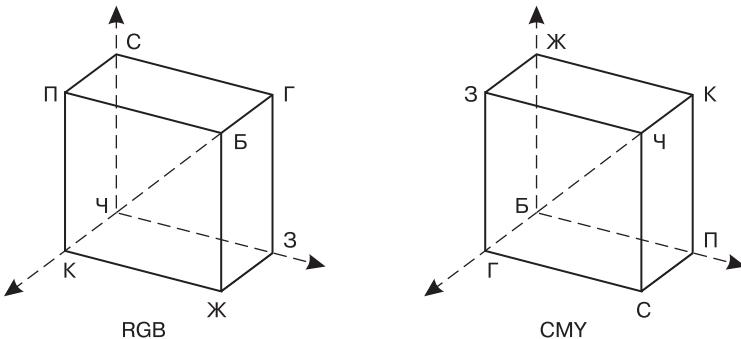


Рис. 15.8. Цветовые пространства RGB и CMY

Началом координат в цветном кубе RGB является черный цвет, а в CMY – белый. Ахроматические, то есть серые цвета, в обеих моделях расположены по диагонали от белого до черного.

Модели RGB и CMY являются аппаратно-ориентированными. Существуют и другие модели, ориентированные на пользователя, в основе которых лежат интуитивно понятные художникам понятия разбела, оттенка и тона.

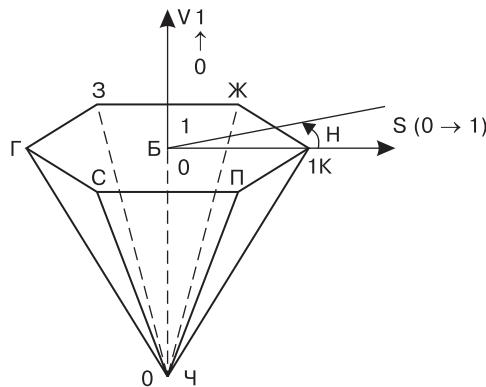
### Цветовая модель HSV

В 1978 году Элви Реем Смитом была предложена модель субъективного восприятия в виде объемного тела HSV, где:

- H – цветовой тон (Hue);
- S – насыщенность (Saturation);
- V – светлота (Value).

Если цветной куб RGB спроектировать на плоскость вдоль диагонали от белого к черному, получится шестиугольник с основными и дополнительными цветами в вершинах. Интенсивность возрастает от нуля в вершине до единицы на верхней

грани. Насыщенность определяется расстоянием от оси, а тон — углом ( $0\text{--}360^\circ$ ), отсчитываемым от красного цвета. Насыщенность меняется от нуля на оси до единицы на границе шестиугольника (рис. 15.9).



**Рис. 15.9.** Цветовая модель HSV<sup>1</sup>

Насыщенность зависит от цветового охвата (расстояния от оси до границы). При  $S = 1$  цвета полностью насыщены. Ненулевая линейная комбинация трех основных цветов не может быть полностью насыщена. Если  $S = 0$ , цветовой тон не определен, то есть лежит на центральной оси и является ахроматическим (серым). Чистые цвета у художников имеют насыщенность и светлоту, равные единице. Разбелы — это цвета с увеличенным содержанием белого, то есть с меньшей насыщенностью. Они лежат на плоскости шестиугольника. Оттенки, цвета с уменьшенной светлотой, образуют ребра от вершины. Тон — это цвета с уменьшенной насыщенностью и с уменьшенной светлотой.

### Цветовая модель HSL

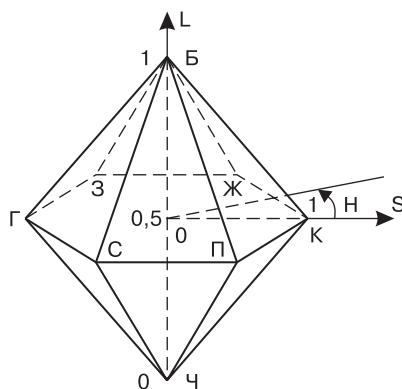
В основе модели HSL, применяемой фирмой Tektronix, лежит цветовая система Освальда:

- H — цветовой тон (Hue);
- S — насыщенность (Saturation);
- L — светлота (Lightness).

Модель представляет собой двойной шестигранный конус. Цветовой тон задается углом поворота вокруг вертикальной оси относительно красного цвета. Цвета следуют по периметру, как и в модели HSV. HSL — результат модификации модели HSV за счет вытягивания вверх белого цвета. Дополнение каждого цвета отстоит на  $180^\circ$  от этого цветового тона. Насыщенность измеряется в радиальном направлении от нуля до единицы. Светлота измеряется вертикально по оси от нуля (черный) до единицы (белый) (рис. 15.10).

Для ахроматических цветов насыщенность равна нулю, а максимально насыщенные цветовые тона получаются при насыщенности 1 и светлоте 0,5.

<sup>1</sup> Фолли Дж., А Вэн Дэм. Основы интерактивной машинной графики. — М.: Мир, 1989.

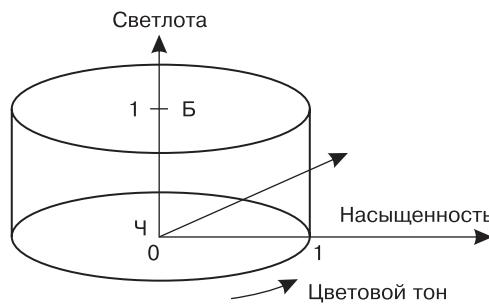
Рис. 15.10. Модель HSL<sup>1</sup>

### Цилиндрическая цветовая модель

В основе цилиндрической цветовой модели лежит цветовая система Манселла, основанная на наборе образцов света. Система Манселла — это стандарт восприятия. Цвет определяется:

- цветовым тоном;
- насыщенностью;
- светлотой.

На центральной оси значения интенсивности меняются от черного к белому (рис. 15.11). Цветовой тон определяется углом. Главное преимущество — одинаковые приращения насыщенности, тона и интенсивности вызывают ощущения одинаковых изменений при восприятии.

Рис. 15.11. Цилиндрическая цветовая модель<sup>2</sup>

### 15.3. Цветовая гармония

Цветные дисплеи и устройства получения твердых копий позволяют передавать широкий диапазон цветов. Одни цветовые сочетания хорошо гармонируют, дру-

<sup>1</sup> Фолли Дж., А Вэн Дэм. Основы интерактивной машинной графики. — М.: Мир, 1989.

<sup>2</sup> Там же.

гие — взаимно несовместимы. Очень важно правильно подбирать цвета, чтобы они гармонировали друг с другом.

Выбор цветов обычно осуществляется путем проведения гладкой траектории в цветовом пространстве и/или путем ограничения диапазона используемых цветов в цветовой модели плоскостями (или шестигранными конусами) постоянной насыщенности. Предпочтительно использование цветов одного и того же цветового тона, двух дополнительных цветов и их смесей, а также цветов постоянной светлоты.

При случайном выборе цветов они будут выглядеть слишком яркими. Если сетку  $16 \times 16$  заполнять цветами случайным образом, она будет иметь не очень привлекательный вид.

Если рисунок включает несколько цветов, то в качестве фона нужно использовать дополнение к одному из них. Если цветов много, то фон лучше сделать серым. Если два примыкающих друг к другу цвета не гармонируют, их можно разделить черной линией.

С физиологической точки зрения низкая чувствительность глаза к синему цвету означает, что на черном фоне трудно различить синий цвет. Отсюда следует, что желтый цвет (дополнительный к синему) трудно различить на белом (дополнительный к черному).

# Глава 16

## Сжатие графических изображений

---

Стоит начать считывать цветные или полутоноевые изображения сканером формата А4, и 100 Мбайт дискового пространства будет заполнено меньше чем за 1 час (размер графического файла – от 400 Кбайт до нескольких мегабайт). А сравниваемый по качеству с телепередачей компьютерный фильм требует хранения данных объемом около 22 Мбайт/с. Поэтому на сегодняшний день остро стоит проблема сжатия и восстановления информации. Но сжатие файла сильно зависит от его структуры.

### 16.1. Графические форматы

Для хранения графических данных используются разные форматы. В файле растровой графики содержится информация, необходимая компьютеру для воссоздания изображения. Каждый формат предусматривает собственный способ кодирования информации о пикселях и других присущих компьютерным изображениям данных.

#### **BMP**

BMP (BitMaP) – это формат растровой графики, используемый для Windows, в котором эта система хранит свои растровые изображения. Для файлов данного формата чаще всего используется расширение BMP, хотя иногда может быть и расширение RLE.

В файлах BMP информация о цвете каждого пикселя кодируется 1, 4, 8, 16 или 24 битами на пикセル. Количество бит на пиксел определяет возможное число цветов в изображении. Изображение при глубине 1 бит на пиксел может иметь всего два цвета, а при глубине 24 бит на пиксел – более 16 млн различных цветов.

Файл состоит из четырех основных разделов: заголовок файла, информационного заголовка растровых данных, таблицы цветов и собственно растровых данных. Заголовок файла содержит информацию о файле, в том числе адрес, с которого начинается область растровых данных. В информационном заголовке растровых данных находятся сведения об изображении, хранящемся в файле, например его высота и ширина в пикселях. В таблице цветов хранятся значения основных цветов RGB (красный, зеленый, синий) для использования в дальнейшем в процессе получения большего количества оттенков в рисунке. Если же видеoadаптер не позволяет отображать более 256 цветов, то для точной цветопередачи можно программно устанавливать такие значения RGB в адаптерах таблиц цветов.

## TIFF

Формат TIFF (Tagged Image File Format) появился в 1993 году. Он применялся в графическом редакторе PhotoStyler. Несмотря на то что эта программа в настоящее время уже не используется, формат TIFF по-прежнему популярен.

В основном он применяется в издательских системах, то есть там, где требуется получать изображения наилучшего качества.

TIFF имеет еще одно неоспоримое достоинство — он очень удобен при переносе изображений между компьютерами различных типов (например, с PC на Macintosh и обратно).

TIFF — один из самых сложных форматов. В его спецификации определено достаточно большое количество разделов. Например, один раздел хранит информацию о ширине изображения в пикселях, другой — о его высоте. В третьем разделе содержится таблица цветов, а в четвертом — сами растровые данные. Изображение, представленное в формате TIFF, полностью определяется его разделами, и этот формат файла легко расширяется, поскольку для придания файлу дополнительных свойств достаточно лишь определить дополнительные типы разделов.

Растровые данные файла TIFF могут сжиматься с использованием любого из нескольких методов, поэтому для чтения файлов TIFF должны быть средства распаковки RLE, LZW и несколько других.

И хотя файловый формат TIFF достаточно сложен, он остается одним из лучших для передачи растровых массивов из одной программы в другую благодаря своей универсальности. Он позволяет кодировать в двоичном виде практически любое изображение без потери его визуальных или каких-либо иных атрибутов.

## GIF

Формат GIF (Graphics Interchange Format), появившийся в 1987 году, является очень старым, но по-прежнему популярен в Интернете. От других графических форматов его отличает использование режима индексированных цветов (не более 256). Но это не мешает GIF быть любимым форматом веб-мастеров, которые применяют его для создания и оформления веб-страниц. Главной причиной этого является использование небольших по размеру файлов.

Файл формата GIF начинается с заголовка, содержащего код, который говорит о том, что это именно GIF-файл; далее следует номер версии GIF и другая информация. Если файл хранит одно изображение, то за заголовком обычно располагается общая таблица цветов, определяющая цвета изображения. Если в файле хранится несколько изображений (формат GIF, аналогично TIFF, позволяет кодировать в одном файле два и более изображений), то вместо общей таблицы цветов каждое изображение сопровождается своей собственной таблицей цветов.

Файл GIF имеет одну особенность — постепенное отображение картинки на экране. В этом случае строки изображения выводятся на экран не подряд, а в определенном порядке: сначала каждая восьмая, затем — четвертая и т. д. Таким образом, полностью изображение показывается в четыре прохода, что позволяет еще до полной загрузки понять его суть и в случае необходимости прервать загрузку.

Очень популярны анимированные GIF-файлы. Например, собачки, смешно машиущие ушами, или крокодил, раскрывающий свою зубатую пасть.

Основные достоинства формата GIF заключаются в широком распространении этого формата и его компактности. Но в изображениях, хранящихся в виде GIF-файла, не может быть использовано более 256 цветов.

## **PSD**

PSD (Photoshop Document) является «родным» форматом Adobe Photoshop. Он может хранить информацию по слоям. Этот формат поддерживает глубину цвета вплоть до 16 бит на канал (48 бит на пиксель для цветных изображений и 16 бит на пиксель для черно-белых). Кроме того, хранится информация об альфа-каналах, слоях, контурах, прозрачности, векторных надписях и т. д.

Формат PSD используется для хранения изображений, содержащих специфические, свойственные только Adobe Photoshop, элементы. PSD-файлы могут быть открыты во многих популярных программах просмотра графических файлов.

## **PDF**

Формат PDF (Portable Document Format) в свое время был разработан компанией Adobe. Он применяется для описания документов. Для создания, редактирования и просмотра PDF-файлов используются специальные программы (например, Acrobat Reader). Этот формат весьма широко применяется в процессе допечатной подготовки.

PDF-формат позволяет выполнять много различных операций. Например, создавать электронные документы для обмена данными. На сегодняшний день существует масса приложений, которые понимают данные в формате PDF и могут читать PDF-файлы. Кроме того, можно формировать интерактивные документы. Файлы в формате PDF могут применяться для создания электронных форм, информация из которых хранится в базе данных.

## **JPEG**

Формат JPEG (Joint Photographic Experts Group) является наиболее популярным среди профессионалов и любителей цифровой фотографии. Это легко объяснимо, поскольку именно данный формат обеспечивает минимальные размеры файлов при возможности сохранения 24-битных полноцветных изображений.

В его основе лежит достаточно сложный алгоритм сжатия графических данных, работа которого основана на особенностях человеческого зрения.

Несмотря на то что при сохранении изображений в формате JPEG обеспечивается высокая степень сжатия, имеют место потери данных. И чем сильнее сжимается изображение, тем большими будут потери. Поэтому при использовании данного формата следует идти на компромисс и выбирать такую степень сжатия, при которой потери данных будут практически незаметны для человеческого глаза.

## 16.2. Основные сведения о сжатии изображений

Сжатие информации делится на архивацию и компрессию. Первая происходит без потери качества, вторая — с потерями. Разница между этими способами в том, что при втором не происходит полного восстановления исходного сохраненного изображения в полном качестве. Но каким бы ни был алгоритм компрессии данных, для работы с ним файл нужно проанализировать и распаковать, то есть вернуть данные в исходный незапакованный вид для их быстрой обработки (обычно это происходит незаметно для пользователя).

*Архивация* графических данных используется как для растровой, так и для векторной графики. При этом способе уменьшения данных программа анализирует наличие в сжимаемых данных некоторых одинаковых последовательностей данных и исключает их, записывая вместо повторяющегося фрагмента ссылку на предыдущий такой же (для последующего восстановления). Такими одинаковыми последовательностями являются пиксели одного цвета, повторяющиеся текстовые символы или некая информация, которая в рамках массива данных повторяется несколько раз. Например, растровый файл с фоном строго одного цвета (например, синего) имеет в своей структуре очень много повторяющихся фрагментов.

*Компрессия* — это способ сохранения данных при использовании которого не гарантируется полное восстановление исходной графической информации. При таком способе хранения обычно графические данные немного теряются по сравнению с оригинальными, но этими искажениями вполне можно пренебречь и управлять. Обычно файлы, сохраненные с использованием этого способа, занимают значительно меньше дискового пространства, чем файлы, сохраненные с помощью простой архивации. Суть методов сжатия с потерей качества — используя особенности восприятия графической информации человеком, отбросить часть данных безвозвратно. Чем выше степень компрессии, тем больше ущерб качеству. Оптимальное решение выбирается для конкретного случая с учетом применения.

Компрессия нужна не всегда. Сократить объем графических данных можно, проанализировав имеющийся файл и уменьшив его размер, цветность или разрешение.

## 16.3. Алгоритмы сжатия файлов без потерь

Как известно, любой файл, невзирая на то, какая информация в нем хранится, состоит из символов и, возможно, невидимых кодов управления печатью. Каждый символ в кодах ASCII представляется 1 байтом. Ниже рассмотрены несколько алгоритмов архивации.

### Алгоритм Хаффмана

Символы заменяются кодовыми последовательностями разной длины. Чем чаще используется символ, тем короче код (например, буквы «а», «е», «и», «с» — 3 бита, «щ», «х», «э», «ю» — 8 бит). Могут использоваться готовые кодовые таблицы или

строиться новые на основе статистического анализа конкретного файла. Гарантируется возможность декодирования, хотя кодовые последовательности имеют разную длину. Степень сжатия — до 50 %.

### **Алгоритм Лемпеля—Зива (LZW)**

Алгоритм сжатия данных LZW основан на поиске и замене в исходном файле одинаковых последовательностей данных для их исключения и уменьшения размера файла. В отличие от предыдущего метода сжатия, LZW более разборчиво просматривает сжимаемые данные, что приводит к лучшему результату.

Данный алгоритм основан на сведении к минимуму избыточности. Вместо кодирования каждого символа кодируются часто встречающиеся последовательности символов (например, слова «который», «также»). Имена же собственные, встречающиеся один раз, не кодируются.

Программа алгоритма просматривает файл с текстом или байтами графической информации и выполняет статистический анализ для построения кодовой таблицы.

Если заменить 60–70 % текста символами, длина которых меньше половины от первоначальной, можно добиться сжатия примерно 50 %.

При применении этого алгоритма к загрузочным файлам (EXE, COM) результат составляет 10–20 %, так как избыточность кода, создаваемого компиляторами, меньше избыточности текста на естественном языке.

Файлы баз данных тоже архивируются незначительно, так как могут содержать редко повторяющуюся информацию (имена, номера телефонов, адреса).

Графические контурные файлы архивируются хорошо, так как обладают большой избыточностью (фон).

Полутоновые и цветные изображения тоже можно архивировать, но с меньшим успехом.

Данный тип сжатия не вносит искажений в исходный графический файл и подходит для обработки растровых данных любого типа: черно-белых, монохромных или полноцветных. Наилучшие результаты получаются при компрессии изображений с большими областями одного цвета или с повторяющимися одинаковыми структурами.

Этот метод демонстрирует самые поразительные результаты степени сжатия (среди других существующих способов сжатия графических данных) при полном отсутствии потерь или искажений в исходных файлах. Используется в файлах формата TIFF, PDF, GIF, PostScript и др.

### **Алгоритм RLE (Run Length Encoding)**

Изображение рассматривается как последовательность байтов. Одинаковые байты кодируются парой: первый байт (count) является счетчиком одинаковых байтов, а второй — байтом из кодируемой последовательности. Для отличия счетчика два его старших бита устанавливаются равными единице. Это позволяет кодировать последовательности длиной не более 63. Байты изображения со значениями больше 191 кодируются двумя байтами.

Достоинствами метода являются простота и скорость декодирования.

Декодирование происходит следующим образом. Для очередного байта осуществляется проверка, установлены ли старшие биты неравными нулю. Если да, то байт является счетчиком, количество повторений равно count (сбрасываются старшие биты). Следующий байт переписывается в видеопамять в count-экземплярах. Если старшие биты байта не установлены, то он переписывается в видеопамять без изменения.

### **CCITT Group 3, CCITT Group 4**

Эти два похожих метода сжатия графических данных работают с однобитными изображениями, сохраненными в цветовой модели Bitmap. Они основаны на поиске и исключении из исходного изображения дублирующихся последовательностей данных (как и в предыдущем типе сжатия — RLE). Оба ориентированы на упаковку именно растровой графической информации, так как работают с отдельными рядами пикселов в изображении.

Изначально алгоритм был разработан для сжатия данных, передаваемых через факсимильные системы связи (CCITT Group 3), а более совершенная разновидность этого метода архивации данных (CCITT Group 4) подходит для записи монохромных изображений с более высокой степенью сжатия.

Как и предыдущий алгоритм, он в основном подходит для сжатия изображений с большими одноцветными областями. Его достоинство — скорость выполнения, а недостаток — ограничения при компрессии графических данных (не все данные удается эффективно сжать таким образом). Этот метод применяется для файлов формата PDF, PostScript и др.

### **Обрезка «хвостов»**

При форматировании жесткий диск разбивается на области (кластеры). Каждый кластер содержит определенное количество секторов по 512 байт. В зависимости от НЖМД (накопителя на жестком магнитном диске) кластер содержит 4, 8 или 16 секторов (2, 4 или 8 Кбайт). Операционная система очень неэкономично управляет ресурсами. Она всегда выделяет для файла целое количество кластеров. Если нужно сохранить файл размером 1 байт, выделяется один кластер размером 8 Кбайт, причем оставшееся место использовано не будет. По теории вероятности на каждом файле теряется в среднем  $1/2$  кластера. (Чтобы проверить потерянное место на диске, можно воспользоваться программой FileSize из пакета Norton Utilities.)

Программы сжатия данных помогают занять практически все свободное пространство диска, упаковывая и размещая больше данных на неиспользованных участках.

## **16.4. Сжатие с потерями цветных и полутоновых файлов**

Цветные и полутоновые файлы содержат большое количество информации (цветные — 24 бита на пиксел, черно-белые — 8 бит на пиксел) и могут «весить» до

нескольких мегабайт (25 Мбайт при сканировании цветного изображения (10 точек на миллиметр)). Для таких файлов характерно постоянное изменение информации вдоль линии сканирования.

Алгоритмы сжатия с потерями основаны на особенностях цветовой чувствительности человеческого глаза. Глядя на картинку, человек выделяет крупные цветовые пятна, переходы между ними, но может проигнорировать мелкие детали, изменения оттенков, абсолютную яркость.

Например, глаз человека воспринимает относительную яркость, а не абсолютную. Точка телевизора не может быть чернее, чем серый цвет выключенного телевизора. Видимый иссиня-черный цвет — не более чем иллюзия, которая возникает из-за соседства с ним контрастных ярких тонов.

Также можно заметить, что на большой площади изображения изменения цвета и интенсивности часто незначительны (например, для неба). Сжатие по методу RLE позволяет уменьшить размеры файлов в два-три раза. Но это не решает проблему полностью, нужны более высокие степени сжатия.

### **Сжатие изображения по стандарту JPEG**

Данный алгоритм обеспечивает уменьшение размера файла в 25–100 раз за счет сжатия с потерями. Он разработан Международной организацией по стандартизации (International Organization for Standardization, ISO). Метод JPEG достаточно сложен с вычислительной точки зрения, так как занимает много процессорного времени.

Первоначальное и восстановленное изображения — не одно и то же. Но информация усекается не просто так. Некоторые данные можно исключить, и большинство людей этого не заметят. Кроме того, пользователь может контролировать уровень потерь, указывая степень сжатия в зависимости от того, что для него важнее — качество изображения или экономия памяти.

Кодирование изображения по алгоритму JPEG подразделяется на несколько этапов.

1. Преобразование цветового пространства из RGB в YUV. Канал Y содержит информацию о яркости, U и V — о цвете. Система зрения человека особенно чувствительна к Y-компоненте и менее чувствительна к U и V.
  - Y-компонента — это цветное изображение, показанное на черно-белом телевизоре.
  - U-компонента — информация о синем цвете.
  - V-компонента — информация о красном цвете.

Поэтому Y-компонента будет сжиматься в меньшей степени, чем U и V.

2. Прореживание. В U- и V-компонентах отбрасываются строки или столбцы пикселов с определенными номерами. Например, при прореживании с коэффициентами 2:1:1 будет отбрасываться информация о цвете для каждой второй строки и каждого второго столбца, в результате чего будет потеряно 75 % данных цветности. При коэффициенте 1:1:1 прореживания нет. На Y-компоненте прореживание не отражается.

3. Дискретное косинусное преобразование (ДКП). Это удивительная математическая операция, которая позволяет представить значения цвета в удобном для последующего сжатия виде (рис. 16.1).

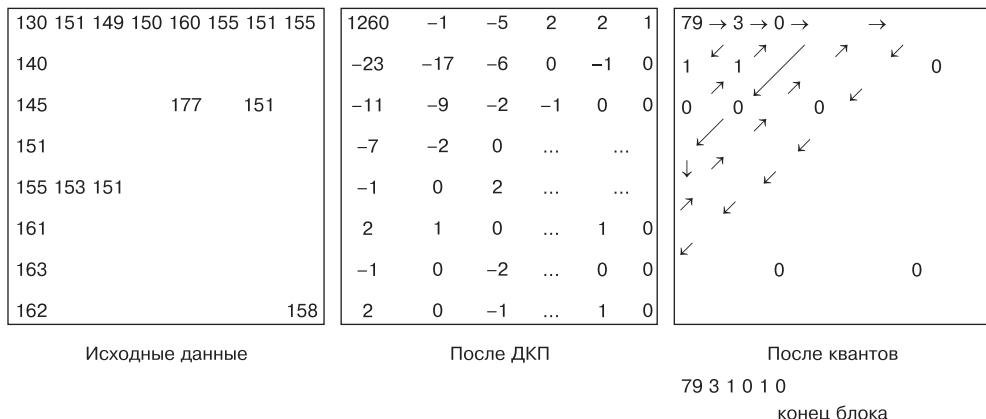
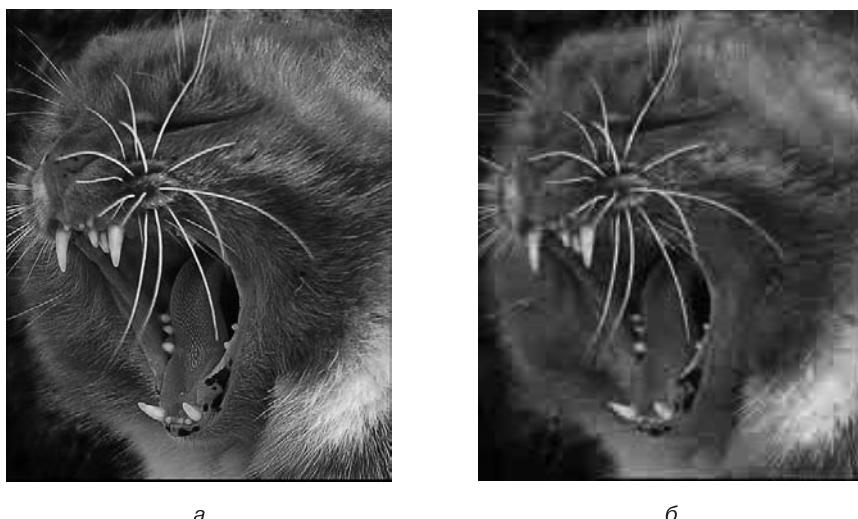


Рис. 16.1. Этапы сжатия

ДКП выполняется отдельно для Y-, U- и V-компонент. Изображение разбивается на блоки размером  $8 \times 8$  пикселов. Такой участок с большой вероятностью содержит пиксели близкого цвета. При ДКП информация о 64 пикселях преобразуется в матрицу из 64 коэффициентов, которые характеризуют «энергию» исходных пикселов. Максимальные значения коэффициентов концентрируются в левом верхнем углу матрицы  $8 \times 8$ , минимальные — в правом нижнем. Первый коэффициент передает подавляющую часть «энергии», а количество «энергии», представляющей остальными коэффициентами, быстро убывает. Таким образом, большая часть информации исходной матрицы  $8 \times 8$  пикселов представляется первым элементом матрицы, преобразованной по ДКП. На этом этапе происходит некоторая потеря информации, связанная с принципиальной невозможностью точного обратного преобразования. Однако она незначительна по сравнению с потерями на следующем этапе.

4. Квантование. Применяется для сокращения разрядности коэффициентов и заключается в делении имеющихся значений на какое-либо число. После отбрасываются малые изменения коэффициентов. Здесь происходит значительная потеря информации. Поэтому после восстановления изображения получаются уже другие значения пикселов. (Для U- и V-компонент квантование более грубое.)
5. Полученные данные сжимаются по RLE-, LZW-алгоритму или алгоритму Хаффмана для достижения еще большей компрессии. Помимо применяемого кодирования наиболее часто встречающихся символов, последние нули в конце строки могут быть заменены символом «конец блока», а так как все блоки имеют одинаковый размер, всегда известно, сколько нулей было опущено.

При восстановлении изображения шаги выполняются в обратном порядке. Изображения, в которых соседние пиксели мало отличаются друг от друга, лучше поддаются сжатию. Однако чем меньше размер выходного файла, тем меньше степень «аккуратности» при работе программы-конвертора и, соответственно, ниже качество выходного изображения. Обычно в программах, позволяющих сохранять растровые данные, есть возможность некоторого компромисса между объемом выходного файла и качеством изображения. При лучшем качестве (рис. 16.2, *а*) объем выходного файла в 3–5 раз меньше исходного незапакованного, при качестве похуже (рис. 16.2, *б*) — меньше исходника в десятки раз, но, как правило, при этом качество изображения уже не позволяет использовать его в ответственных задачах.



**Рис. 16.2.** Примеры сжатия изображения по алгоритму JPEG

Данный формат предназначен для хранения в основном фотографических изображений с большим количеством оттенков и цветовых переходов и почти не подходит для хранения однотонных изображений типа кадров из мультфильмов, скриншотов (сжатие будет слишком низким или качество картинки достигнет критической отметки). Этот метод сжатия графических данных используется в файлах формата PDF, PostScript, собственно в JPEG и др.

6. Сглаживание в процессе восстановления изображения. Из-за потери информации на границах между блоками ( $8 \times 8$  пикселов) могут возникать разрывы. Поэтому необходимо сглаживание.

### **Новый стандарт JPEG 2000**

В разработке данного стандарта приняли участие Международная организация по стандартизации, Международный союз телекоммуникаций (International

Telecommunications Union), компании Agfa, Canon, Fujifilm, Hewlett-Packard, Kodak, LuraTech, Motorola, Ricoh, Sony и др. Он позволяет сжимать изображения в 200 раз без заметной для невооруженного глаза потери качества. Основным отличием JPEG 2000 от предыдущей версии этого формата является использование алгоритма волнового преобразования (изображение описывается с помощью математических выражений как непрерывный поток) вместо преобразования Фурье, что и предотвращает появление характерных блоков. JPEG 2000 также умеет без ущерба модифицировать (масштабировать, редактировать) рисунок, сохраненный в этом формате. Алгоритм волнового преобразования позволяет просматривать и распечатывать одно и то же изображение при различных (заданных пользователем) значениях разрешения и с требуемой степенью детализации. Благодаря этой особенности JPEG 2000 быстро найдёт свое место в Сети, поскольку обеспечит возможность загружать картинку с разными значениями разрешения в зависимости от пропускной способности конкретного канала связи. Немаловажен и тот факт, что пользователи Интернета смогут получать изображения высокого качества. Еще одно значимое преимущество JPEG 2000 – возможность управлять 256 цветовыми каналами, а в результате получать качественные цветные изображения. Специалисты обещают, что в общем случае новый формат будет как минимум на 30 % эффективнее, чем JPEG. И еще один плюс – новый стандарт является открытым.

Главным недостатком компрессии с частичной потерей качества является то, что эти потери, выражаяющиеся в искажении цветового тона или появлении характерной кубической структуры в контрастных участках изображения, возникают каждый раз при сохранении изображения и накладываются друг на друга при многократном сохранении файла в этом формате. Поэтому рекомендуется использовать форматы с частичной потерей качества только для хранения окончательных результатов работы, а не для промежуточных рабочих файлов.

### **Фрактальное сжатие изображений**

Революция в обработке изображений реального мира произошла с выходом в 1977 году книги Б. Мандельброта «Фрактальная геометрия природы». *Фрактал* – это структура, обладающая схожими формами разных размеров. Такие структуры могут имитироваться с помощью рекурсивных функций. Фракталы не зависят от разрешения устройства. Это масштабированные картинки, которые можно описать небольшим конечным набором инструкций с помощью компьютерной программы.

Процесс сжатия изображения состоит из следующих этапов.

1. Разделение изображения на неперекрывающиеся области (домены). Набор доменов должен перекрывать все изображение полностью.
2. Задание набора ранговых областей изображения. Ранговые области могут перекрываться. Они не обязательно закрывать всю поверхность картинки.
3. Фрактальное преобразование. Для каждого домена подбирается такая ранговая область, которая после аффинного преобразования наиболее точно ап-

проксимирует домен. Подобное преобразование не только сжимает и деформирует изображение ранговой области, но и изменяет яркость и контраст.

4. Сжатие и сохранение параметров аффинного преобразования. Файл со сжатым изображением содержит две части: заголовок, включающий в себя информацию о расположении доменов и ранговых областей, и эффективно упакованную таблицу аффинных коэффициентов для каждого домена.

Этапы восстановления изображения такие.

1. Создание двух изображений одинакового размера, А и Б. Их размер может быть не равен размеру исходного изображения. Начальный рисунок областей А и Б не имеет значения. Это могут быть случайные данные, белое или черное.
2. Преобразование данных из области А в область Б. Для этого сначала изображение Б делится на домены так же, как и на первой стадии процесса сжатия (расположение доменов описано в заголовке файла). Теперь для каждого домена области Б проводится соответствующее аффинное преобразование ранговых областей изображения А, описанное коэффициентами из сжатого файла, и результат помещается в область Б. На этой стадии создается совершенно новое изображение.
3. Преобразование данных из области Б в область А. Этот шаг идентичен предыдущему, только изображения Б и А поменялись местами, то есть теперь разделяется на блоки область А и на эти блоки отображаются ранговые области изображения Б.
4. Многократно повторяются второй и третий шаги процедуры восстановления данных до тех пор, пока изображения А и Б не станут неразличимыми.

Обманчиво простой процесс попеременного отображения двух изображений друг на друга приводит к созданию репродукции исходной картинки. Точность соответствия зависит от точности аффинного преобразования, коэффициенты которого вычисляются в процессе сжатия.

Алгоритмы сжатия и восстановления информации используют целочисленную арифметику и специальные методы уменьшения накапливающихся ошибок округления. В отличие от распространенных в настоящее время методов сжатия/восстановления графических изображений, фрактальное преобразование асимметрично: процесс восстановления нельзя провести путем простой инверсии процедуры сжатия. Сжатие требует гораздо большего количества вычислений, чем восстановление.

В процессе фрактального преобразования получается набор цифр, который в очень сжатой форме описывает изображение. Достигаемый при этом большой коэффициент сжатия позволяет хранить и передавать высококачественные изображения. При высоком разрешении исходного изображения фрактальное отображение гораздо более эффективно с точки зрения снижения объема сжатой информации.

## **Список литературы**

---

1. Абраш М. Программирование графики. Таинства. — Киев: ЕвроСИБ, 1996.
2. Александров В. В., Шнейдеров В. С. Рисунок, картина, чертеж на ЭВМ. — Ленинград: Машиностроение, 1988.
3. Аммерал Л. Машинная графика на персональных компьютерах. — М.: Сол-Систем, 1992.
4. Аммерал Л. Принципы программирования в машинной графике. — М.: Сол-Систем, 1992.
5. Гардан И., Люка М. Машинная графика и автоматизация конструирования. — М.: Мир, 1987.
6. Климов В. Е. Графические системы САПР. — М.: Высшая школа, 1990.
7. Котов Ю. В. Как рисует машина. — М.: Наука, 1988.
8. Роджерс Д. Алгоритмические основы машинной графики. — М.: Мир, 1989.
9. Роджерс Д. Математические основы машинной графики. — М.: Мир, 2003.
10. Тихомиров Ю. Программирование трехмерной графики. — СПб., 1998.
11. Фолли Дж., А Вэн Дэм. Основы интерактивной машинной графики. — М.: Мир, 1989.
12. Шикин Е. В., Боресков А. В. Компьютерная графика. Динамика, реалистические изображения. — М.: ДИАЛОГ-МИФИ, 1995.
13. Эндерле Г., Кэнси К. Программные средства машинной графики. Международный стандарт GKS. — М.: Радио и связь, 1988.
14. Эндженел И. Практическое введение в машинную графику. — М.: Радио и связь, 1984.
15. Hearn D., Baker M. P. Computer Graphics. — Prentice Hall, 1994.

## **Адреса сайтов**

---

- <http://www.graphicon.ru/>
- <http://algolist.manual.ru/>
- <http://www.ict.edu.ru/>
- <http://www.ixbt.com/>
- <http://graphics.cs.msu.ru/library/>
- <http://www.compress.ru/>
- <http://3drazer.com/>
- <http://www.render.ru/>

*Сиденко Людмила Адамовна*

**Компьютерная графика и геометрическое моделирование:  
Учебное пособие**

Заведующий редакцией  
Руководитель проекта  
Ведущий редактор  
Художник  
Корректоры  
Верстка

*Д. Гурский  
Ю. Чернушевич  
М. Моисеева  
С. Шутов  
В. Сабайдা, Н. Терех  
В. Нога*

Подписано в печать 19.09.08. Формат 70×100/16. Усл. п. л. 18,06. Тираж 2500. Заказ

ООО «Питер Пресс», 198206, Санкт-Петербург, Петергофское шоссе, 73, лит. А29.

Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.

Отпечатано по технологии СоД в ОАО «Печатный двор» им. А. М. Горького.  
197110, Санкт-Петербург, Чкаловский пр., 15.



# Нет времени ходить по магазинам?



наберите:



**www.piter.com**



Здесь вы найдете:

Все книги издательства сразу

Новые книги — в момент выхода из типографии

Информацию о книге — отзывы, рецензии, отрывки

Старые книги — в библиотеке и на CD



**И наконец, вы нигде не купите  
наши книги дешевле!**



СПЕЦИАЛИСТАМ  
КНИЖНОГО БИЗНЕСА!

**ПРЕДСТАВИТЕЛЬСТВА ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»**  
предлагают эксклюзивный ассортимент компьютерной, медицинской,  
психологической, экономической и популярной литературы

### **РОССИЯ**

**Москва** м. «Электрозводская», Семеновская наб., д. 2/1, корп. 1, 6-й этаж;  
тел./факс: (495) 234-3815, 974-3450; e-mail: sales@piter.msk.ru

**Санкт-Петербург** м. «Выборгская», Б. Сампсониевский пр., д. 29а;  
тел./факс (812) 703-73-73, 703-73-72; e-mail: sales@piter.com

**Воронеж** Ленинский пр., д. 169; тел./факс (4732) 39-43-62, 39-61-70;  
e-mail: pitervrn@comch.ru

**Екатеринбург** ул. Бебеля, д. 11а; тел./факс (343) 378-98-41, 378-98-42;  
e-mail: office@ekat.piter.com

**Нижний Новгород** ул. Совхозная, д. 13; тел. (8312) 41-27-31;  
e-mail: office@nnov.piter.com

**Новосибирск** ул. Станционная, д. 36;  
тел./факс (383) 350-92-85; e-mail: office@nsk.piter.com

**Ростов-на-Дону** ул. Ульяновская, д. 26; тел. (8632) 69-91-22, 69-91-30;  
e-mail: piter-ug@rostov.piter.com

**Самара** ул. Молодогвардейская, д. 33, литер А2, офис 225; тел. (846) 277-89-79;  
e-mail: pitvolga@samtel.ru

### **УКРАИНА**

**Харьков** ул. Сузdalские ряды, д. 12, офис 10–11; тел./факс (1038067) 545-55-64,  
(1038057) 751-10-02; e-mail: piter@kharkov.piter.com

**Киев** пр. Московский, д. 6, кор. 1, офис 33; тел./факс (1038044) 490-35-68, 490-35-69;  
e-mail: office@kiev.piter.com

### **БЕЛАРУСЬ**

**Минск** ул. Притыцкого, д. 34, офис 2; тел./факс (1037517) 201-48-79, 201-48-81;  
e-mail: office@minsk.piter.com



Ищем зарубежных партнеров или посредников, имеющих выход на зарубежный рынок.  
Телефон для связи: **(812) 703-73-73**.

**E-mail:** fuginov@piter.com



Издательский дом «Питер» приглашает к сотрудничеству авторов.  
Обращайтесь по телефонам: **Санкт-Петербург – (812) 703-73-72,**  
**Москва – (495) 974-34-50.**



Заказ книг для вузов и библиотек: (812) 703-73-73.

Специальное предложение – e-mail: kozin@piter.com



**УВАЖАЕМЫЕ ГОСПОДА!  
КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»  
ВЫ МОЖЕТЕ ПРИОБРЕСТИ  
ОПТОМ И В РОЗНИЦУ  
У НАШИХ РЕГИОНАЛЬНЫХ ПАРТНЕРОВ.**

### **Дальний Восток**

Владивосток, «Приморский торговый дом книги»,  
тел./факс (4232) 23-82-12.  
E-mail: bookbase@mail.primorye.ru

Хабаровск, «Деловая книга»,  
ул. Путевая, д. 1а,  
тел. (4212) 36-06-65, 33-95-31  
E-mail: dkniiga@mail.kht.ru

Хабаровск, «Книжный мир»,  
тел. (4212) 32-85-51, факс 32-82-50.  
E-mail: postmaster@worldbooks.kht.ru

Хабаровск, «Мирс»,  
тел. (4212) 39-49-60.  
E-mail: zakaz@booksmirs.ru

### **Европейские регионы России**

Архангельск, «Дом книги»,  
пл. Ленина, д. 3  
тел. (8182) 65-41-34, 65-38-79.  
E-mail: marketing@avfkniga.ru

Воронеж, «Амиталь»,  
пл. Ленина, д. 4,  
тел. (4732) 26-77-77.  
<http://www.amital.ru>

Калининград, «Вестер»,  
сеть магазинов «Книги и книжечки»,  
тел./факс (4012) 21-56-28, 65-65-68.  
E-mail: nshibkova@vester.ru  
<http://www.vester.ru>

Самара, «Чакона», ТЦ «Фрегат»,  
Московское шоссе, д.15,  
тел. (846) 331-22-33.  
E-mail: chaconne@chaccone.ru

Саратов, «Читающий Саратов»,  
пр. Революции, д. 58,  
тел. (4732) 51-28-93, 47-00-81.  
E-mail: manager@kmsvrn.ru

### **Северный Кавказ**

Ессентуки, «Россы», ул. Октябрьская, 424,  
тел./факс (87934) 6-93-09.  
E-mail: rossy@kmw.ru

### **Сибирь**

Иркутск, «Продалитъ»,  
тел. (3952) 20-09-17, 24-17-77.  
E-mail: prodalit@irk.ru  
<http://www.prodalit.irk.ru>

Иркутск, «Светлана»,  
тел./факс (3952) 25-25-90.  
E-mail: kkcbooks@bk.ru  
<http://www.kkcbooks.ru>

Красноярск, «Книжный мир», пр. Мира, д. 86,  
тел./факс (3912) 27-39-71.  
E-mail: book-world@public.krasnet.ru

Новосибирск, «Топ-книга»,  
тел. (383) 336-10-26, факс 336-10-27.  
E-mail: office@top-kniga.ru  
<http://www.top-kniga.ru>

### **Татарстан**

Казань, «Таис»,  
сеть магазинов «Дом книги»,  
тел. (843) 272-34-55.  
E-mail: tais@bancorp.ru

### **Урал**

Екатеринбург, ООО «Дом книти»,  
ул. Антона Валека, д. 12,  
тел./факс (343) 358-18-98, 358-14-84.  
E-mail: domknigi@k66.ru

Челябинск, ТД «Эврика», ул.Барбюса, д. 61,  
тел./факс (351) 256-93-60.  
E-mail: evrika@bookmagazin.ru  
<http://www.bookmagazin.ru>

Челябинск, ООО «ИнтерСервис ЛТД»,  
ул. Артиллерийская, д. 124  
тел. (351)247-74-03, 247-74-09, 247-74-16.  
E-mail: zakup@intser.ru  
<http://www.fkniga.ru ,www.intser.ru>