

Министерство образования Республики Беларусь  
БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

---

Кафедра «Системы автоматизированного проектирования»

Л.А. СИДЕНКО

## ГЕОМЕТРИЧЕСКОЕ МОДЕЛИРОВАНИЕ

Учебное пособие по курсу «Геометрическое моделирование» для студентов специальности «»

Минск 2006

УДК

ББК

И

Авторы:

Рецензенты:

В методическом пособии рассмотрены тематика, требования по оформлению и порядок выполнения курсовой работы, а также приведены примеры расчета пневмогазовой и паровой трубчатой сушилок.

## Содержание

<b>ВВЕДЕНИЕ.....</b>	<b>6</b>
<b>1. ОБЩИЕ СВЕДЕНИЯ О КОМПЬЮТЕРНОЙ ГРАФИКЕ.....</b>	<b>7</b>
1.1. История развития компьютерной графики .....	7
1.2. Основные сведения о графических системах.....	9
1.3. Функции графических систем.....	10
1.4. Графические данные .....	11
1.5. Блок-схема графической системы.....	12
<b>2. ГЕОМЕТРИЧЕСКИЕ ПРЕОБРАЗОВАНИЯ .....</b>	<b>13</b>
2.1. Двумерные преобразования .....	13
2.2. Однородные координаты и матричное представление двумерных преобразований .....	16
2.3. Композиции двумерных преобразований.....	20
2.4. Матричное представление трехмерных преобразований.....	22
2.5. Композиция трехмерных преобразований .....	23
2.6. Преобразования как изменение систем координат .....	26
<b>3. АЛГОРИТМЫ РАСТРОВОЙ ГРАФИКИ .....</b>	<b>29</b>
3.1. Преобразование отрезков из векторной.....	29
3.2 Ускорение алгоритма Брезенхэма.....	32
3.3. Растворная развертка литер .....	33
3.4. Растворная развертка окружностей.....	35
3.5 Растворная развертка эллипсов.....	38
3.6 Методы устранения ступенчатости растровых изображений .....	40
3.7 Устранение искажений в растровых дисплеях.....	41
3.8 Сглаживание линий .....	42
3.9 Заполнение области .....	424
3.10. Разложение в растр сплошных многоугольников .....	46
<b>4. ОТСЕЧЕНИЕ ЛИНИЙ .....</b>	<b>49</b>
4.1. Алгоритм Коэна-Сазерленда .....	49
4.2. Алгоритм разбиения средней точкой .....	52
4.3 Трехмерное отсечение отрезков .....	53
4.4 Отсечение многоугольников .....	56
4.5 Отсечение литер .....	59

<b>5. ПРОЕКТИРОВАНИЕ ГРАФИЧЕСКОГО ДИАЛОГА .....</b>	<b>60</b>
<b>5.1. Языковая аналогия.....</b>	<b>60</b>
<b>5.2. Языковая модель.....</b>	<b>61</b>
<b>5.3. Принципы проектирования.....</b>	<b>63</b>
<b>5.4. Процесс проектирования.....</b>	<b>68</b>
<b>6. ГЕОМЕТРИЧЕСКОЕ МОДЕЛИРОВАНИЕ. ОБЩИЕ СВЕДЕНИЯ. ....</b>	<b>69</b>
<b>6.1. Геометрическая модель .....</b>	<b>69</b>
<b>6.2. Основные виды ГМ .....</b>	<b>71</b>
<b>6.3. Требования, предъявляемые к геометрическим моделям .....</b>	<b>75</b>
<b>6.4. Внутреннее представление, типы данных .....</b>	<b>75</b>
<b>7. ДВУМЕРНОЕ МОДЕЛИРОВАНИЕ .....</b>	<b>77</b>
<b>7.1. Типы данных.....</b>	<b>77</b>
<b>7.2. Построение базовых элементов .....</b>	<b>77</b>
<b>7.3. Примеры моделей .....</b>	<b>79</b>
<b>8. ТРЕХМЕРНОЕ МОДЕЛИРОВАНИЕ .....</b>	<b>80</b>
<b>8.1. Типы данных.....</b>	<b>80</b>
<b>8.2. Методы описания трехмерных объектов .....</b>	<b>82</b>
<b>8.3. Методы построения трехмерных моделей .....</b>	<b>86</b>
<b>9. ОПИСАНИЕ И ХАРАКТЕРИСТИКА ПОВЕРХНОСТЕЙ.....</b>	<b>95</b>
<b>9.1. Описание поверхностей .....</b>	<b>95</b>
<b>9.2. Характеристики поверхностей.....</b>	<b>98</b>
<b>9.3. Моделирование деформации трехмерных полигональных поверхностей         в режиме реального времени .....</b>	<b>103</b>
<b>9.4. Триангуляция поверхностей.....</b>	<b>108</b>
<b>10. ПОЛУЧЕНИЕ РЕАЛИСТИЧНЫХ ИЗОБРАЖЕНИЙ .....</b>	<b>116</b>
<b>10.1. Методы создания реалистических изображений.....</b>	<b>116</b>
<b>10.2. Перспективные изображения .....</b>	<b>118</b>
<b>11. ПРОЕКТИРОВАНИЕ .....</b>	<b>116</b>
<b>11.1. Основные виды проекций .....</b>	<b>119</b>
<b>11.2. Математическое описание прямоугольных проекций.....</b>	<b>124</b>
<b>11.3. Математическое описание косоугольных проекций .....</b>	<b>126</b>
<b>11.4. Математическое описание перспективной проекции .....</b>	<b>126</b>
<b>11.5. Задание произвольных проекций. Видовое преобразование. ....</b>	<b>130</b>

<b>12. АЛГОРИТМЫ УДАЛЕНИЯ СКРЫТЫХ ЛИНИЙ И ПОВЕРХНОСТЕЙ</b>	<b>133</b>
.....	.....
12.1. Общие сведения об удалении скрытых линий и поверхностей .....	133
12.2. Алгоритм сортировки по глубине .....	134
12.3. Алгоритм, использующий z-буфер .....	134
12.4. Алгоритм построчного сканирования.....	136
12.5. Алгоритм разбиения области .....	138
12.6. Сравнительная характеристика алгоритмов .....	140
12.7. Алгоритм плавающего горизонта .....	140
12.8. Алгоритм Робертса .....	143
12.9. Алгоритм трассировки лучей.....	144
12.10. Иерархический Z—буфер.....	146
<b>13. СВЕТ</b> .....	<b>153</b>
13.1. Общие сведения о свете.....	153
13.2. Модель освещения.....	156
13.3. Закраска полигональных сеток.....	164
13.4. Тени.....	166
13.5. Фактура. Нанесение узора. ....	170
13.6. Создание неровностей на поверхности.....	173
13.7. Фильтрация текстур. .....	182
13.8. Полутоновые изображения. ....	187
<b>14. ТРАССИРОВКА ЛУЧЕЙ</b> .....	<b>190</b>
14.1 Метод прямой трассировки .....	190
<b>15. ИСПОЛЬЗОВАНИЕ ЦВЕТА В КОМПЬЮТЕРНОЙ ГРАФИКЕ</b> .....	<b>199</b>
15.1. Ахроматический и хроматический цвет .....	199
15.2. Цветовые модели .....	202
15.3. Цветовая гармония.....	207
<b>16. СЖАТИЕ ИЗОБРАЖЕНИЙ</b> .....	<b>207</b>
16.1. Основные сведения.....	207
16.2. Алгоритмы сжатия файлов без потерь.....	208
16.3. Сжатие цветных и полутоновых файлов. Сжатие с потерями.....	211

## ВВЕДЕНИЕ

«Потенциальные возможности компьютерной графики грандиозны, ограничения же зависят от только от нашей фантазии. И чем она богаче, тем полнее раскрываются возможности компьютерной графики». Эти слова великого американского ученого Ликлайдера, сказанные десятки лет назад, по истине подтверждаются действительностью.

До недавнего времени компьютерная графика представляла собой весьма специфическое занятие, требующее дорогостоящей аппаратуры, значительных ресурсов памяти и своеобразного программного обеспечения.

В настоящее время сформировалась новая отрасль информатики – компьютерная графика. Ее можно определить как науку о математическом и геометрическом моделировании форм и размеров объектов, а также методов их визуализации.

Интерес к синтезу изображений объясняется высокой информативностью последних. Информация, содержащаяся в изображении, представлена в наиболее концентрированной форме, и эта информация, как правило, более доступна для анализа.

Стремление визуализировать информацию наблюдается практически во всех сферах деятельности человека. И начиная с 60-х годов, которые считаются началом ее зарождения, до наших дней пройден большой путь. Сегодня можно наблюдать с экрана дисплея не только графики, диаграммы, чертежи, но и полноценные динамически изменяющиеся трехмерные объекты. Причем они соответствуют реальным объектам не только по форме и расположению, но и по цвету, фактуре, освещению. Все это позволяет проявлять все больший интерес к компьютерной графике, применяя ее для решения большого класса задач. Большинство специалистов, использующих компьютерную графику, уделяет особое внимание проблемам программирования, задачам конкретного проектирования, создания различных технических средств.

Целью данного учебного пособия является систематическое изложение математических методов и технических аспектов, лежащих в основе компьютерной графики. Применена единая система обозначений. Подробно изложены принципы построения графических систем, способы преобразования объектов, методов описания кривых и поверхностей. Особое внимание удалено геометрическому моделированию, так необходимому при проектировании технических объектов. Кроме этого представлена новейшая информация по визуализации объектов, получению реалистичных изображений, использовании освещения, теней, наложении рисунка и фактуры.

# 1. ОБЩИЕ СВЕДЕНИЯ О КОМПЬЮТЕРНОЙ ГРАФИКЕ

## 1.1. История развития компьютерной графики

### Методы вывода

Дисплейные (Д) устройства, разработанные в 60-х годах и используемые в настоящее время, называются векторными.

Они состоят:

- Дисплейный процессор (ДПц).
- Буферная память.
- ЭЛТ.

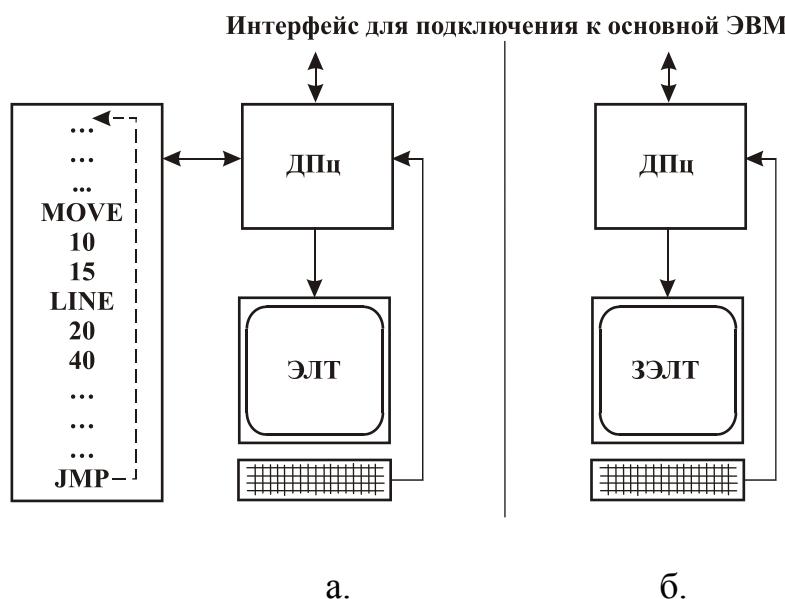


Рис. 1.1

Буфер служит для запоминания, подготовленного, дисплейного списка (или дисплейной программы). Дисплейная программа включает команды вывода точек, отрезков, литер. Эти команды интерпретируются дисплейным процессором, который преобразует цифровые значения в аналоговые напряжения, управляющие электронным лучом. Луч вычерчивает линии на люминофорном покрытии ЭЛТ. Так как светоотдача люминофора падает до нуля за десятки микросекунд, дисплейный процессор должен осуществлять цикл по заданной программе с целью регенерации изображения на люминофоре с частотой не меньше 30 раз в секунду для устранения мерцания. В связи с этим буфер, в котором хранится дисплейная программа называют *буфером регенерации* (рис. 1.1, а).

Команды перехода (JMP) обеспечивает возврат на начало дисплейной программы с целью обеспечения циклической регенерации.

В 60-х годах буферная память большого объема и быстрые дисплейные процессоры с частотой регенерации не меньшей 30 Гц были очень дорогими. В связи с этим важным этапом на пути обеспечения доступности машинной графики

ки было создание в конце 60-х годов запоминающих ЭЛТ, которые позволили отказаться от буфера и регенерации (рис. 1.1, б).

В ЗЭЛТ изображение запоминается путем его однократной записи относительно медленно движущимся электронным лучом на запоминающую сетку с люминофором. Запоминающие трубы до сих пор применяются в тех случаях, когда надо вывести большое количество отрезков и литер и когда нет необходимости в динамических операциях с изображением.

В середине 70-х годов была изобретена растровая графика, основанная на телевизионной технике. В растровых дисплеях примитивы (отрезки литер) хранятся в памяти для регенерации в виде совокупности образующих их точек, называемых пикселями (р). Изображение формируется на растре, представляя 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 30 раз в секунду по отдельным строкам растра в направлении сверху вниз (рис. 1.2).

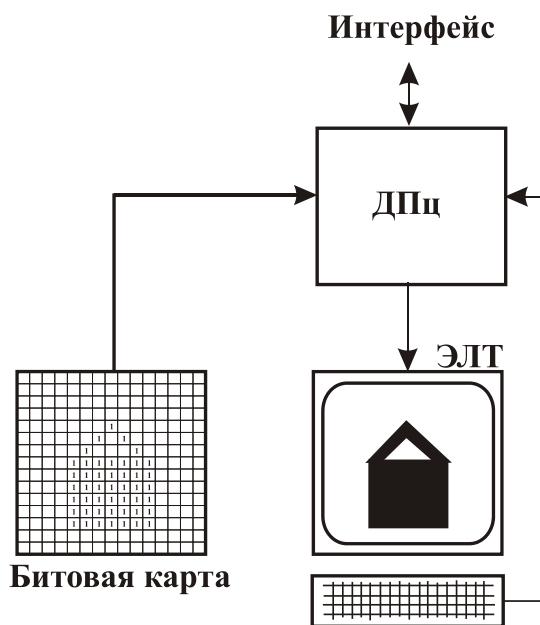


Рис. 1.2

Недостатки:

- резко возрастает потребность в памяти, так как полное изображение, состоящее из большого числа пикселов должно храниться как битовая карта;
- разрешающая способность растровых графических систем пока еще ниже, чем векторных ( $1280 \times 1024$  против  $4096 \times 4096$  пикселов);
- для отображения примитивов надо больше времени, так как в векторных дисплеях задаются две конечные точки (для отрезка), а в растровом надо рассмотреть еще и все промежуточные точки отрезка.

Достоинства:

- растровая графика по сравнению с векторной лучше закрашивает изображения;
- процесс регенерации не зависит от сложности рисунка. Векторные же дисплеи часто начинают мерцать, когда число примитивов в буфере становится таким большим, что его нельзя считать и обработать за 1/30 с, в результате чего изображение регенерируется не достаточно часто;
- дешевизна.

### **Методы ввода**

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

### **Развитие программного обеспечения**

Многие трудности в развитии программного обеспечения были связаны с примитивностью графического ПО. Процесс совершенствования ПО был длительным и медленным. Был пройден путь от аппаратно-зависимых пакетов низкого уровня, поставляемых изготовителем 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 одной ЭВМ на другую.

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

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

Таким образом, компьютерная — это создание, хранение и обработка моделей объектов и их изображений с помощью ЭВМ.

А графические системы служат для:

- создания;
- поиска;
- хранения;
- модификации графических данных.

Графические системы могут быть:

- пассивные;
- интерактивные.

*Пассивные* — обеспечивают только вывод графических изображений, но человек при этом не имеет возможности прямого воздействия на графические преобразования.

*Интерактивные* — обеспечивают возможность пользователю динамически управлять содержанием изображения.

В таких системах используются интерактивные графические дисплеи, позволяющие пользователю работать в диалоге с графическим изображением.

Области применения графических систем представлены в таблице 1.1.

Таблица 1.1

Области применения графических систем

Область применения	Синтез изображения	Анализ изображения	Обработка изображения
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

### 1.3. Функции графических систем

Функции интерактивных графических систем:

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

Функции ввода реализуются с помощью графических устройств ввода (клавиатура, планшет, мышь, световое перо и т.д.).

Функции вывода — с помощью графопостроителей, дисплеев, станков с ЧПУ.

Функции обработки запросов пользователя на входных и командных языках реализуются программой, называемой лингвистическим (диалоговым) процессором. Процессор преобразует описания геометрии объектов, заданных 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;
- Коды для управления графической системой.

Первые поступают из входных языков, вторые — из командных.

Параметры объекта поступают через СУБД в базу данных.

Коды для управления графической системой поступают в монитор. Он управляет работой системы.

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

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

Функции геометрического процессора:

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

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

## 1.4. Графические данные

Изображение объекта строится из простых геометрических элементов. Для преобразования изображения должны быть известны взаимосвязи между элементами.

Если изображением является многоугольник, надо упорядочить и указать взаимосвязи между отдельными его ребрами, и тогда становится возможным удаление отдельных ребер и всего многоугольника, несмотря на то, что в качестве исходных данных указываются только отдельные отрезки. Для подобного рода логического структурирования существуют различные методы, такие как линейные списки, деревья.

Пример древовидной структуры на рис.1.4.1.

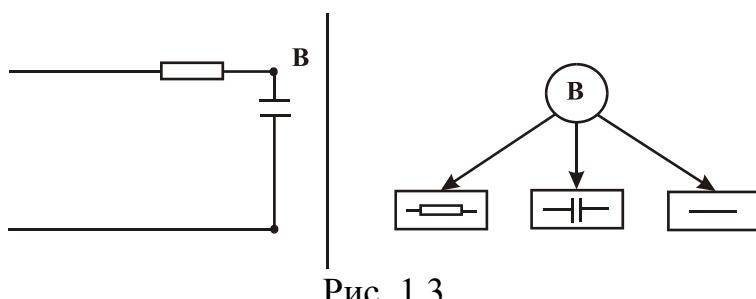


Рис. 1.3

Требования к структурам графических БД:

- гибкость (с точки зрения применимости);
- преобразуемость;
- наглядность иерархической структуры;
- возможность и быстрота доступа к любому месту.

Методы реализации структур данных в памяти ЭВМ:

- Последовательного накопления.

Преобразования последовательно упорядоченных графических данных — сложный и длительный процесс. Однако при таком преобразовании из-за отсутствия индекса, характеризующего взаимосвязи между параметрами, минимальны затраты на организацию памяти.

- Табличная организация.

При табличном методе хранения отдельные параметры связаны между собой. Дополнительная информация, характеризующая взаимосвязи параметров, упрощает из удаление и введение новых. Но жестко установленная последовательность записи значений параметров сохраняется. Структура БД становится более гибкой.

- Прямой доступ.

При прямом доступе известен адрес каждого параметра и поиск осуществляется по этому адресу.

## 1.5. Блок-схема графической системы

Центральный процессор и память на рисунке не показаны. В памяти размещаются два важных информационных модуля:

Прикладная структура данных, которая содержит описание объектов, изображения которых должны показываться на экране. Она же является моделью объектов (от описания объекта в структуре данных к его изображению на экране).

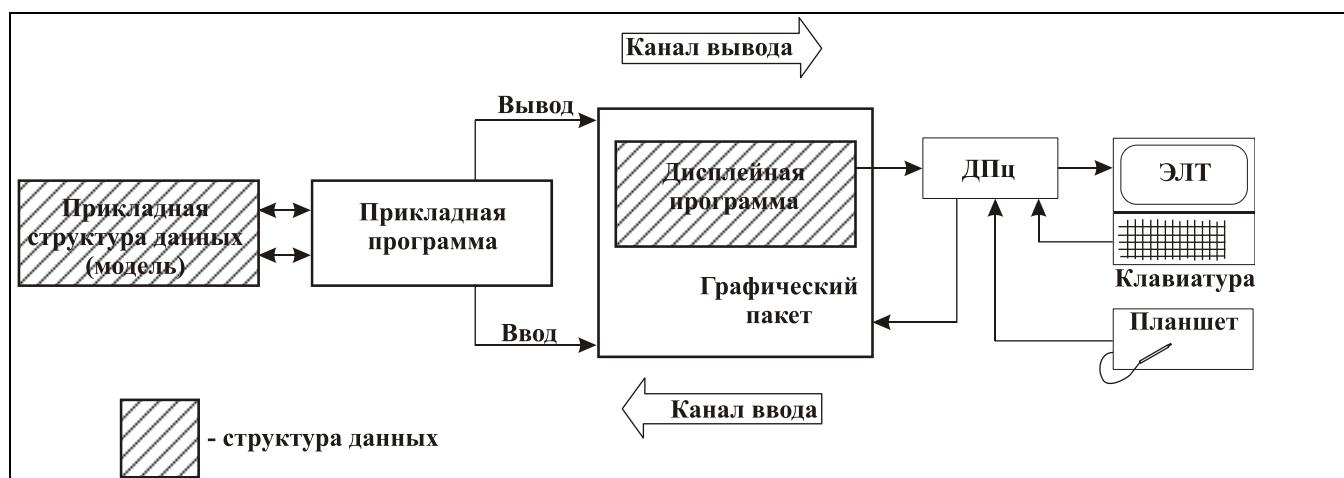


Рис. 1.4

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

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

ется графическим пакетом. Последний формирует дисплейную программу, используемую дисплейным процессором для получения изображения. Таким образом, канал ввода последовательно преобразует описание объекта во все более машинно-зависимые представления.

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

Дисплейная программа, которая формируется графически пакетом и читается дисплейным процессором во время регенерации изображения на экране (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 осуществляет специальная программа ввода, которая передает их прикладной программе. Эти данные меняют состояние прикладной программы. Они могут побудить также прикладную программу модифицировать структуру данных, изменить параметры.

## 2. ГЕОМЕТРИЧЕСКИЕ ПРЕОБРАЗОВАНИЯ

### 2.1. Двумерные преобразования

#### 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:

$$x' = x + Dx, \quad y' = y + Dy. \quad (1)$$

Определим векторы-строки:

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

тогда уравнения (1):

$$[x' \ y'] = [x \ y] + [Dx \ Dy], \quad (2)$$

или кратко:

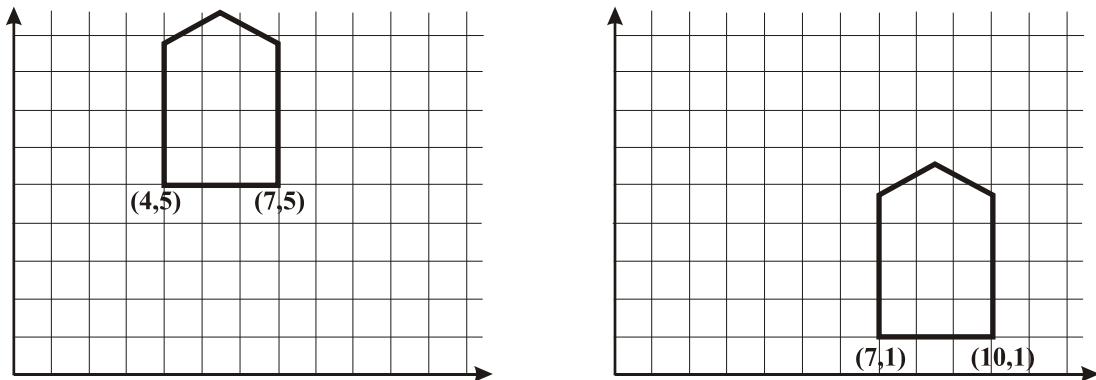
$$P' = P + T. \quad (3)$$

Но объект есть множество точек. Его можно переносить, применяя уравнения (1) к каждой его точке. Но каждый отрезок состоит из бесконечного числа точек и этот процесс длился бы бесконечно долго. Удобнее все точки, принадлежа-

щие отрезку перенести путем перемещения одних лишь крайних точек отрезка. И потом вычертить новый отрезок между ними.

### Масштабирование

Точки можно масштабировать (растянуть) в  $S_x$  раз по оси  $X$  и в  $S_y$  — по оси  $Y$ .



$$Dx = 3, \quad Dy = -4$$

Рис. 2.1

Получим новые точки с помощью умножения:

$$x' = x \cdot Sx, \quad y' = y \cdot Sy. \quad (4)$$

Определив  $S$  как:

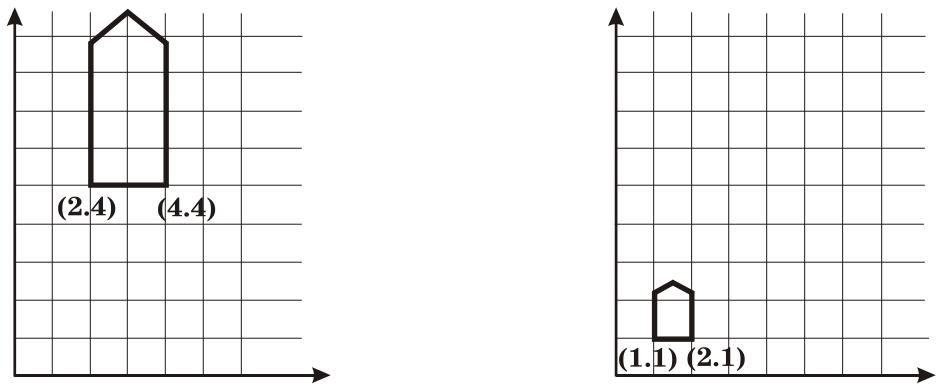
$$S = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix},$$

можно записать в матричной форме:

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \quad (5)$$

или

$$P' = P \cdot S \quad (6)$$



$$Sx=1/2, \quad Sy=1/4$$

Рис. 2.2

Масштабирование производится относительно начала координат — в результате домик стал меньше и ближе к началу координат. Если бы масштабные коэффициенты были больше единицы, домик бы увеличился и отдалился бы 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. Тогда координаты точки  $P'$ :

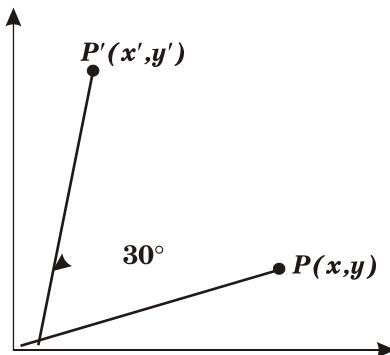


Рис. 2.3

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

В матричной форме:

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} \cos \Theta & \sin \Theta \\ -\sin \Theta & \cos \Theta \end{bmatrix} \quad (8)$$

или

$$P' = P \cdot R \quad (9)$$

где  $R$  — матрица поворота.

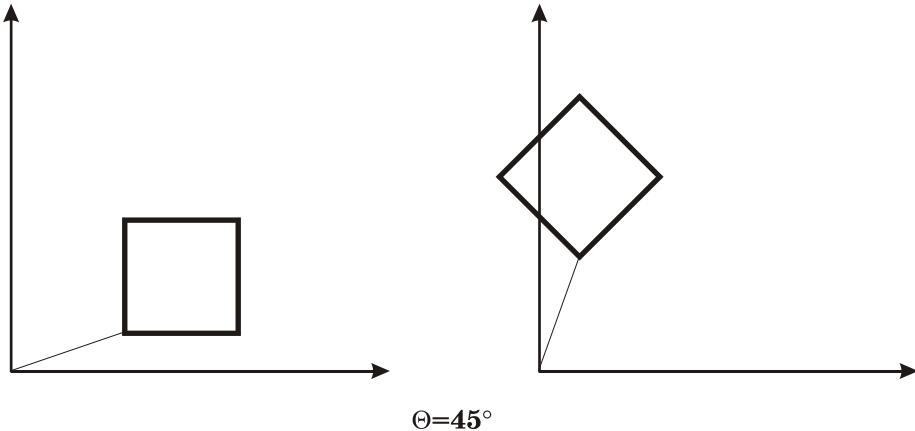


Рис. 2.4

Положительное направление — против часовой стрелки, отрицательное — по часовой стрелке.

## 2.2. Однородные координаты и матричное представление двумерных преобразований

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 — операции умножения. Удобно было бы эти преобразования представить в единой форме. Рассмотрим, как это сделать.

Если мы выразим точки в однородных координатах, то все три преобразования можно реализовать с помощью операции умножения.

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 декартовые координаты:

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

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

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

### Перенос

Уравнение переноса (1) запишется в виде матрицы преобразования:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx & Dy & 1 \end{bmatrix}, \quad (10)$$

или

$$P' = P \cdot T(Dx, Dy), \quad (11)$$

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 получим перенос на расстояние  $(Dx_1 + Dx_2, Dy_1 + Dy_2)$ .

Доказательство:

$$P' = P \cdot T(Dx_1, Dy_1), \quad (12)$$

$$P'' = P' \cdot T(Dx_2, Dy_2). \quad (13)$$

Теперь подставим (12) в (13):

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

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 (4) в матричной форме имеет вид:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (14)$$

Определяя

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

имеем

$$P' = P \cdot S(Sx, Sy). \quad (15)$$

Перемножив, получим:

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

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:

$$\begin{aligned} P' &= P \cdot S(Sx_1, Sy_1), \\ P'' &= P' \cdot S(Sx_2, Sy_2). \end{aligned} \quad (16)$$

Подставляя (16) в (17):

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

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

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

### Поворот

Уравнение поворота (3) можно представить в виде:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \Theta & \sin \Theta & 0 \\ -\sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (18)$$

Полагая

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

имеем:

$$P' = P \cdot R(\Theta). \quad (19)$$

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  $P'$  — в точку  $P''$  при повороте на угол  $\Theta_2$ , то общий поворот равен  $\Theta_1 + \Theta_2$ .

Доказательство:

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

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  $P_1(x_1, y_1)$ .

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

- Перенос точки  $P_1$  в начало координат,  $T(-x_1, -y_1)$ .
- Поворот,  $R(\Theta)$ .

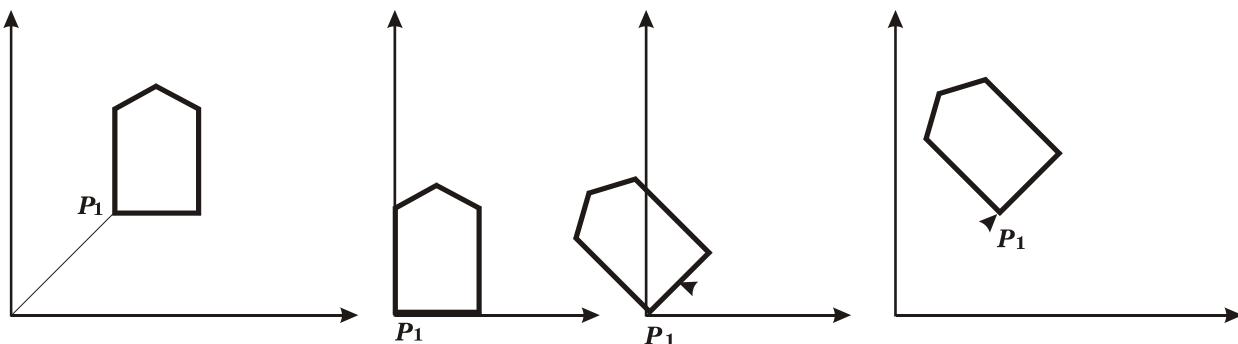


Рис. 2.5

- Перенос точки  $P_1$  из начала координат в начальную позицию,  $T(x_1, y_1)$ . Результирующее преобразование имеет вид:

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

или:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_1 & -y_1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \Theta & \sin \Theta & 0 \\ -\sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_1 & y_1 & 1 \end{bmatrix} =$$

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

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

Аналогично, если надо промасштабировать объект относительно точки  $P_1(x_1, y_1)$ , а не начала координат, то надо:

- Перенести точку  $P_1$  в начало координат,  $T(-x_1, -y_1)$ .
- Масштабировать,  $S(Sx, Sy)$ .
- Перенести точку  $P_1$  назад,  $T(x_1, y_1)$ .

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 надо промаштабировать, 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  $(x_1, y_1)$ . Но может быть и записана матрица результирующего преобразования:

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

## 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:

$$T(Dx, Dy, Dz) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Dx & Dy & Dz & 1 \end{bmatrix}$$
$$P' = P \cdot T(Dx, Dy, Dz)$$
$$[x' y' z' 1] = [x \ y \ z \ 1] \cdot T(Dx, Dy, Dz) = [x + Dx \ y + Dy \ z + Dz \ 1].$$

### 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

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \cdot S(Sx, Sy, Sz) = [x \cdot Sx \ y \cdot Sy \ z \cdot Sz \ 1].$$

### Поворот

Двумерный поворот, рассмотренный ранее, является в то же время трехмерным поворотом вокруг оси  $z$ .

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

Матрица поворота вокруг оси  $X$ :

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

Матрица поворота вокруг оси  $Y$ :

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

При сложном повороте, он раскладывается на составляющие:

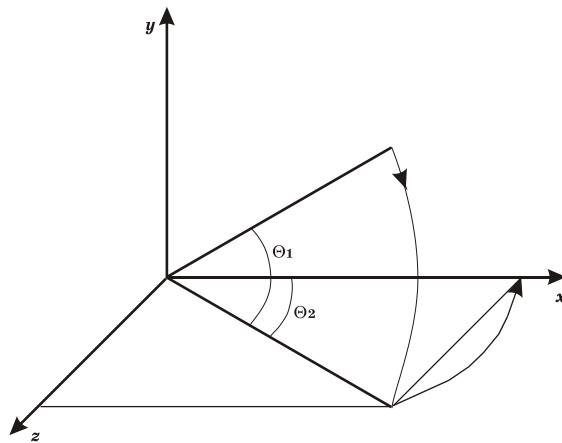


Рис. 2.6

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

## 2.5. Композиция трехмерных преобразований

Путем объединения элементарных трехмерных преобразований можно получить другие преобразования.

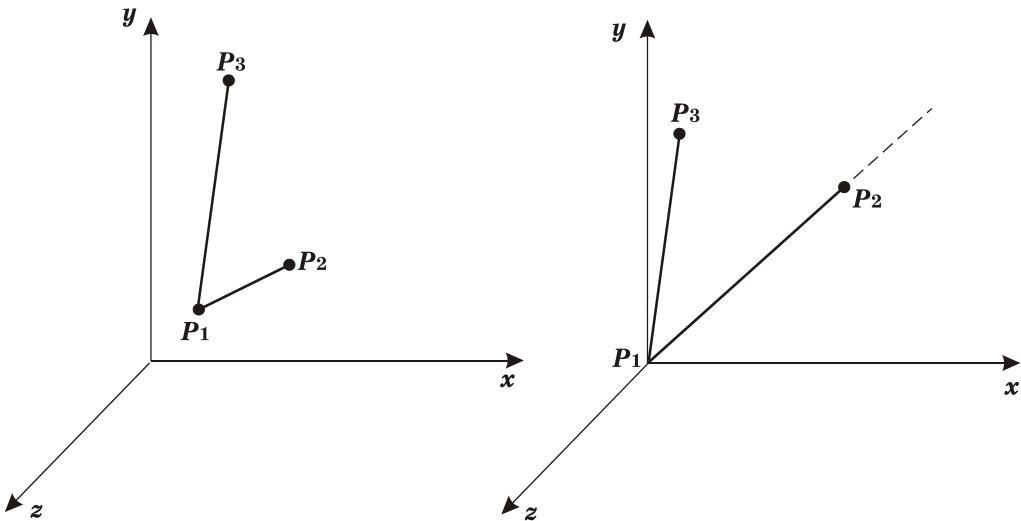


Рис. 2.7

Пример. Преобразовать отрезок  $P_1P_2$  из начальной позиции в конечную таким образом, чтобы точка  $P_1$  совпала с началом координат, а отрезок  $P_1P_2$  располагается вдоль отрицательной полуоси  $Z$ .

На длины отрезков преобразование не воздействует.

Для выполнения этой задачи рассмотрим три шага:

- Перенос точки  $P_1$  в начало координат.
- Поворот вокруг оси  $Y$  до совмещения отрезка  $P_1P_2$  с плоскостью  $YZ$ .
- Поворот вокруг оси  $X$  до совмещения отрезка  $P_1P_2$  с отрицательной полуосью  $Z$ .

### Шаг 1

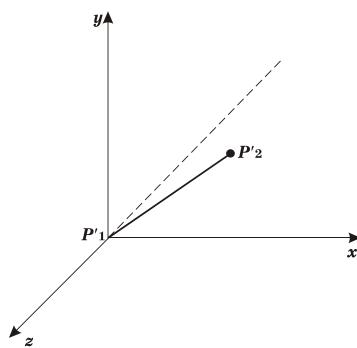


Рис. 2.8

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

Применим  $T$  к  $P_1, P_2, P_3$ :

$$\begin{aligned} P'_1 &= P_1 \cdot T(-x_1, -y_1, -z_1) = [0 \ 0 \ 0 \ 1] \\ P'_2 &= P_2 \cdot T(-x_1, -y_1, -z_1) = [x_2 - x_1 \ y_2 - y_1 \ z_2 - z_1 \ 1] \\ P'_3 &= P_3 \cdot T(-x_1, -y_1, -z_1) = [x_3 - x_1 \ y_3 - y_1 \ z_3 - z_1 \ 1] \end{aligned}$$

## Шаг 2

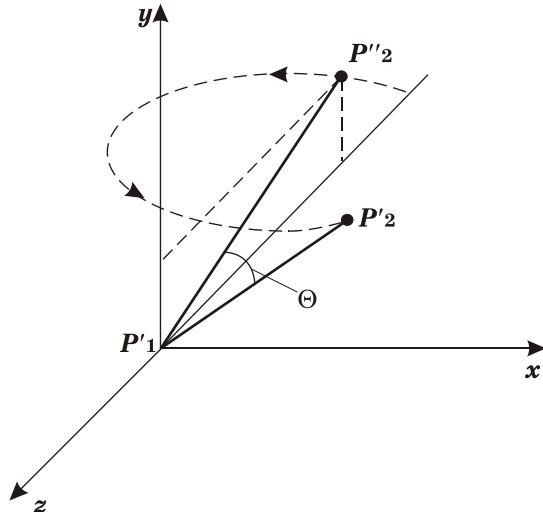


Рис. 2.9

Поворот вокруг оси  $Y$  на угол  $\Theta$  (угол положительный)

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

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

Подставляя эти выражения в матрицу поворота, находим:

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

### Шаг 3

Поворот вокруг оси  $x$  (угол отрицательный)

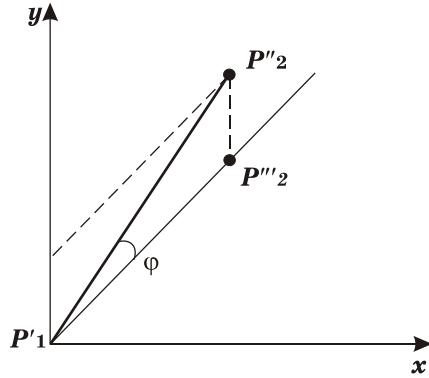


Рис. 2.10

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

Где

$$|P_1 P_2| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

Результат поворота:

$$P'''_2 = P''_2 \cdot R_x(\varphi) = P'_2 \cdot R_y(\Theta) \cdot R_x(\varphi) = P_2 \cdot T \cdot R_y(\Theta) \cdot R_x(\varphi) = \\ = \begin{bmatrix} 0 & 0 & -|P_1 P_2| & 1 \end{bmatrix},$$

теперь отрезок  $P_1 P_2$  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.

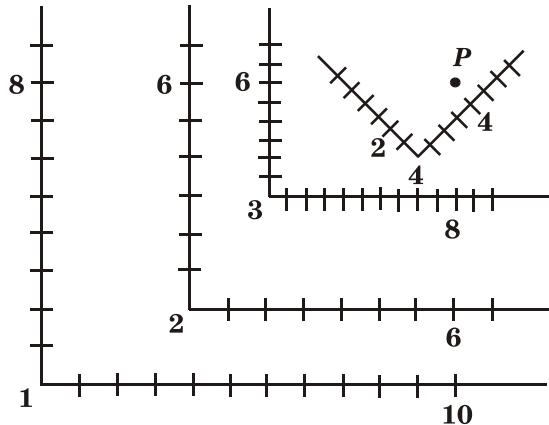


Рис. 2.11

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).

Вначале рассмотрим, как положение точки, заданной в одной СК, можно описать в любой другой СК.

Точка  $P$  имеет координаты:

$$1 \text{ СК} — P_1(10;8);$$

$$2 \text{ СК} — P_2(6;6);$$

$$3 \text{ СК} — P_3(8;6);$$

$$4 \text{ СК} — P_4(4;2).$$

Преобразования СК имеет вид:

$$1 \text{ СК} — 2 \text{ СК}: T_{12} = T(-4, -2);$$

$$2 \text{ СК} — 3 \text{ СК}: T_{23} = T(-2, -3) \cdot S(2, 2);$$

$$3 \text{ СК} — 4 \text{ СК}: T_{34} = T(-6, -2) \cdot R(-45^\circ).$$

Тогда координаты точки  $P$  во 2 СК через 1 СК:

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

Обратное преобразование из 2 СК в 1 СК:

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

$$T_{32} = T_{23}^{-1} = (T(-2, -3) \cdot S(2, 2))^{-1} = S^{-1}(2, 2) \cdot T^{-1}(-2, -3) = S(0.5, 0.5) \cdot T(2, 3).$$

Причем:

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

Таким образом, существует три способа преобразования объектов:

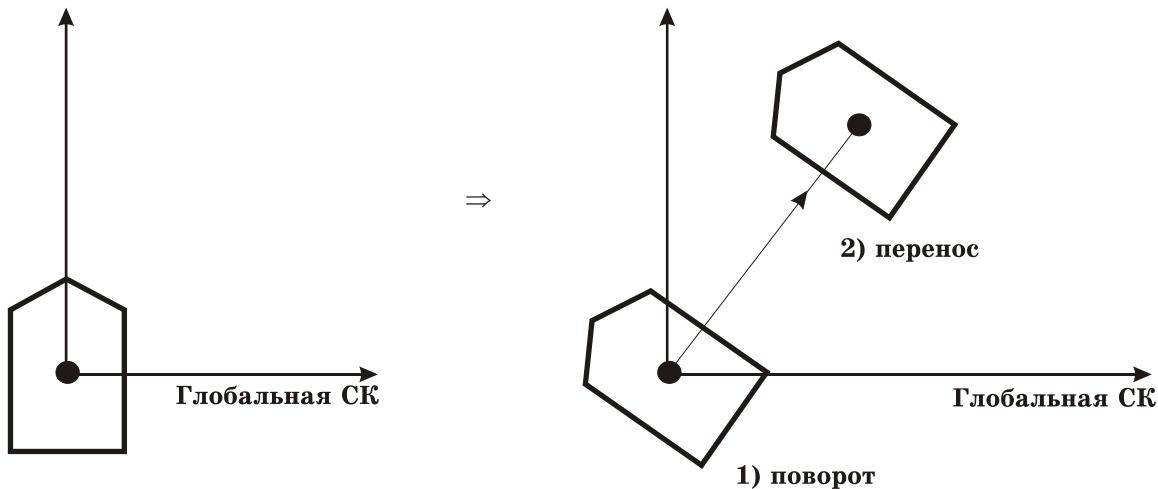


Рис. 2.12 Все объекты описаны в глобальной СК и с помощью преобразований приводятся к новым позициям в той же глобальной СК.

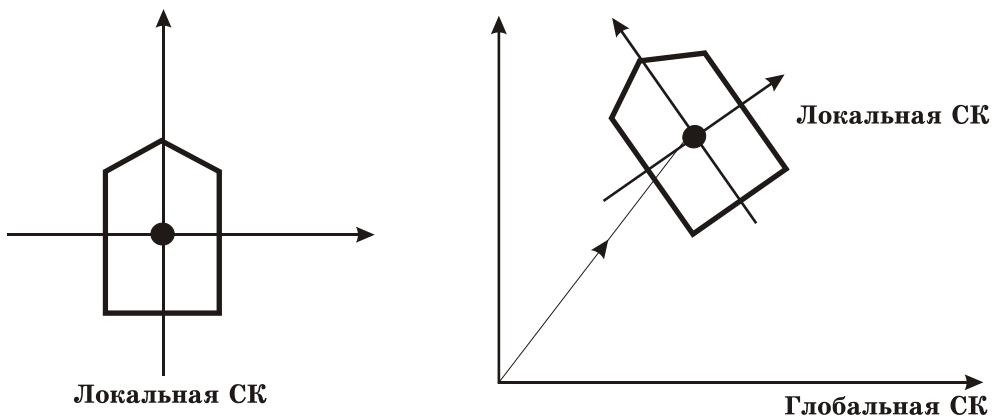


Рис. 2.13 Каждый объект задан в собственной локальной СК и затем преобразуется в глобальную СК.

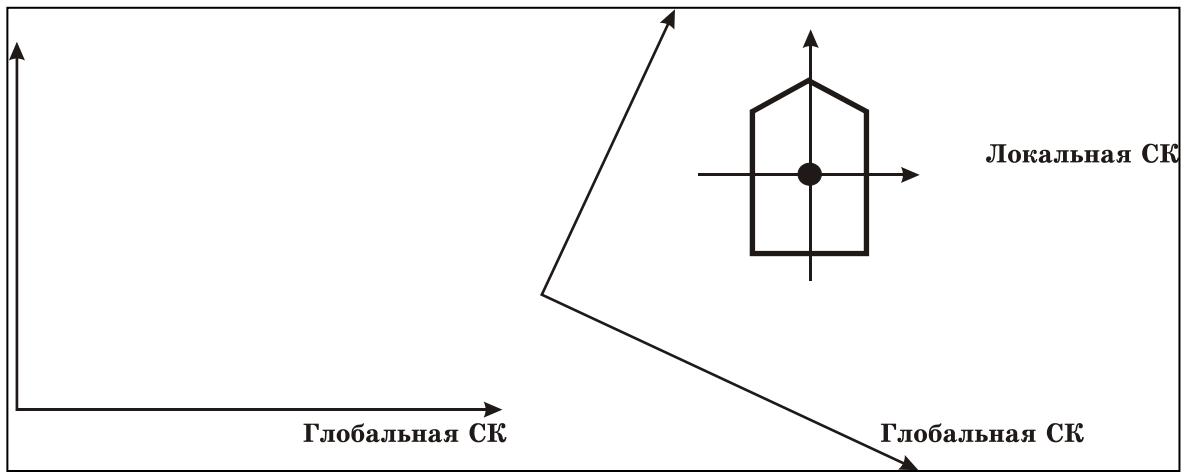


Рис. 2.14 Происходит преобразование систем координат с помощью определения новой глобальной СК относительно локальной СК.

### 3. АЛГОРИТМЫ РАСТРОВОЙ ГРАФИКИ

Алгоритмы развертки, применяемые при построении растрового изображения, вызывается очень часто – при каждом создании или модификации изображения. Поэтому алгоритмы должны давать не только хорошее изображение, но и делать это как можно быстрее.

Основные критерии алгоритмов:

- скорость,
- качество изображения.

Но обычно предпочтение отдается быстродействию.

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

Главной задачей алгоритма развертки отрезков является вычисление координат пикселов ( $\vec{p}$ ) на двумерной растровой сетке. При решении этой задачи будем предполагать, что конечные т. отрезков имеют целые координаты. Простой алгоритм заключается в пошаговом  $\uparrow x$ , вычислении  $y = mx + b$  и высвечивании  $\vec{p}$  в т.  $(x, Round(y))$ . Вычисление произведения  $m \cdot x$  требует  $t$  и замедляет процесс разложения в растр.

##### Пошаговый алгоритм

Операцию умножения можно устранить, если заметить, что при  $\Delta x = 1$   $m = \Delta y / \Delta x$  сводится к  $m = \Delta y$ , т.е. изменение  $x$  на 1 приводит к изменению  $y$  на  $m$  (тангенс угла наклона).

Т.о. если  $x_{i+1} = x_i + 1$ , то

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

т.е. последующие значения т. отрезка определяются, исходя из предыдущих.

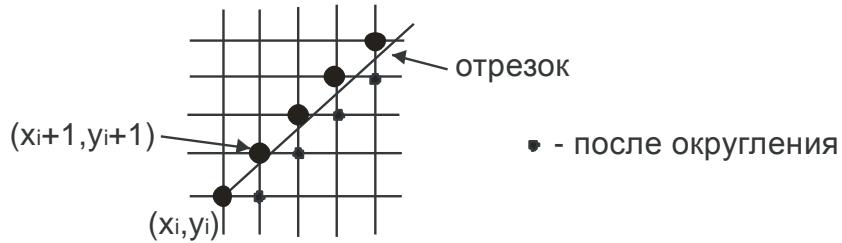


Рис. 3.1

Если  $m > 1$ , тот шаг по  $x$  будет приводить к шагу по  $y > 1$  ( $\Delta y > 1$ ). Поэтому надо  $x$  и  $y$  поменять местами -  $y \uparrow$  на 1, а  $x \uparrow$  на  $1/m$ . После вычисления очередной т. происходит округление ее координат до ближайших целых.

### Алгоритм Брезенхэма

Трудности применения предыдущего метода состоят в том, что округление  $y$  до целого значения требует времени, и кроме того,  $y$  и  $m$  должны быть вещественными числами.

Более привлекателен в этом отношении алгоритм Брезенхэма, т.к. в нем используется только целая арифметика. Вещественные переменные не используются совсем, и, значит, округление не нужно.

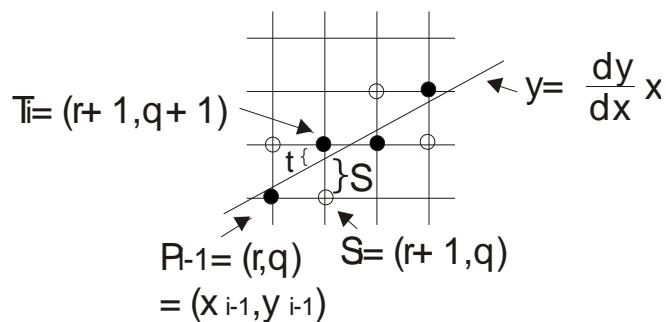


Рис. 3.2

Суть метода: В алгоритме используется управляющая переменная  $d_i$ , которая на каждом шаге пропорциональна разности между  $s$  и  $t$ . Так для т.  $P_i$  надо определить, какой из  $\bar{p}$  будет выбран:  $T_i$  и  $S_i$ ? Если  $s < t$ , то  $S_i$  ближе к отрезку, в противном случае ближе будет  $T_i$ .

Т.е. если  $s - t < 0 \rightarrow S_i$ ,

$$s - t \geq 0 \rightarrow T_i.$$

А теперь рассмотрим алгоритм подробнее.

Есть отрезок  $(x_1, y_1) — (x_2, y_2)$ . Пусть 1-я т. находится ближе к началу координат, тогда перенесем обе точки при помощи  $T(-x_1, -y_1)$  так, чтобы начальная точка отрезка стала  $(0,0)$ , 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 т.  $P_{i-1}$  через  $(r, q)$ . Тогда координаты:

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

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

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

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

Найдем разность:

$$\begin{aligned} s - t &= 2 \frac{dy}{dx}(r + 1) - 2q - 1 \\ dx(s - t) &= 2 \cdot (r \cdot dy - q \cdot dx) + 2dy - dx \end{aligned}$$

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

Обозначим:  $d_i = dx(s - t)$ , имеем:

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

Т.к.

$$r = x_{i-1}, \quad q = y_{i-1},$$

то

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

Прибавляя 1 к каждому индексу:

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

Вычитая  $d_i$  из  $d_{i+1}$ :

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

Но  $x_i - x_{i-1} = 1 \Rightarrow$

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

Т.е. вначале вычисляется  $d_i$ .

Если  $d_i \geq 0$ , выбирается  $T_i$ , тогда

$$\begin{aligned} x_i &= x_{i-1} + 1 \\ y_i &= y_{i-1} + 1 \end{aligned}$$

и

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

Если  $d_i < 0$ , выбирается  $S_i$ , тогда

$$\begin{aligned} x_i &= x_{i-1} + 1 \\ y_i &= y_{i-1} \end{aligned}$$

и

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

Т.о. мы получили итеративный способ вычисления  $d_{i+1}$  по предыдущему значению  $d_i$  и выбора между  $S_i$  и  $T_i$ .

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

### 3.2 Ускорение алгоритма Брезенхэма

1) Частные случаи – горизонтальные, вертикальные и диагональные линии.

2) При разложении отрезка можно заметить, что при движении вдоль основной оси (X) по 1  $\vec{p}$ , получаются группы  $\vec{p}$ , имеющие постоянную координату по неосновной оси (Y).

В стандартном алгоритме Брезенхэма проверка производится на каждом шаге, используя же группы  $\vec{p}$ , количество проверок можно ↓.

Пусть есть отрезок длиной  $35 \vec{p}$  по X и  $10 \vec{p}$  по Y.

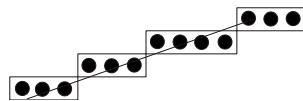


Рис. 3.3

$m = \left[ \frac{1}{3}; \frac{1}{4} \right]$ . Тогда группы  $\vec{p}$  с неизменной Y – координатой будут иметь то 3 то  $4 \vec{p}$ .

Поэтому достаточно 1 раз вычислить  $m$ , чтобы определить группы  $\vec{p}$ . Количество вычислений  $\downarrow$  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  $\vec{p}$  с максимальной и минимальной длиной? На каждый шаг по неосновной оси мы ставим  $n_{\min} \vec{p}$  вдоль основной. После этого решается ставить ли еще  $1 \vec{p}$  или  $\uparrow$  у на 1. Для этого анализируется накопленная ошибка отклонения.



Рис. 3.4

2-ой момент – балансирование групп  $\vec{p}$ . Это делается так: вначале оценивается ошибка накопления не за полный шаг на  $1 \vec{p}$ , а за  $\frac{1}{2} \vec{p}$ . Если 1-ая и последняя группы  $\vec{p}$  состоят из нечетного числа  $\vec{p}$ , получатся несимметричные начала и конец.

### 3.3. Растворная развертка литер

Ранее мы уже рассмотрели общий метод изображения литер, основанный на матрице точек. Тот же метод используется и при построении растровых изобра-

жений. Наименьшей сеткой, с помощью которой можно описывать литеры с приемлемым качеством, является сетка  $5 \times 7$ ; 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  $\vec{p}$  используются для разделения литер и для строчных литер с выносными элементами. Маски литер хранятся в ПЗУ.

При изображении литер надо учитывать 2 особенности:

- 1) пропорциональное размещение литер (это изменение интервалов между центрами литер с учетом их ширины (Пример —  $i$  и  $w$ )).

В случае пропорционального размещения не все литеры будут занимать одно и то же число пикселов по горизонтали.

- 2) нижние выносные элементы букв (части литер, опущенные ниже базовой линии ( $g, p, q, y$ ) изображаются путем сдвига матриц вниз по отношению к другим литерам.)

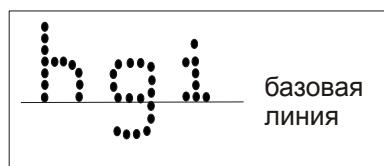


Рис. 3.5

Каждая литерра представляет собой код матрицы из 0 и 1.

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

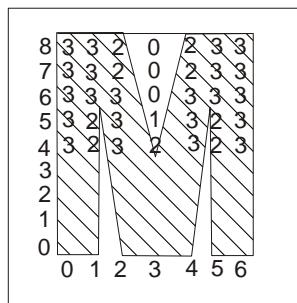


Рис. 3.6

При изображении литер, так же как и при изображении отрезка возникает необходимость сглаживания. Используется принцип выравнивания – надо менять яркость пропорционально площади  $\vec{p}$ , покрытой изображением. Если взять 4 уровня яркости (от 0 до 3), то литерра  $M$  будет выглядеть как показано на рис.

### 3.4. Растворная развертка окружностей

#### Четырехсторонняя симметрия

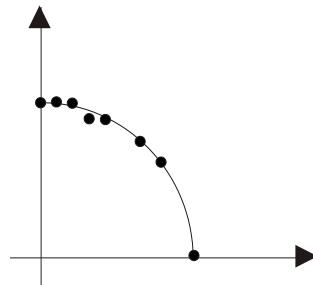


Рис. 3.7

Существует несколько простых, но не эффективных способов развертки окружности. Рассмотрим окружность с центром в начале координат:

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

Чтобы изобразить  $\frac{1}{4}$  окружности, можно  $\uparrow x$  с 0 до  $R$  и на каждом шаге вычислить  $y$ . Остальные четверти получаются симметричным отображением. Этот метод неэффективен, т.к. в него входят операции “\*” и “ $\sqrt{}$ ”. Более того, при значениях  $x$  близких к  $R$  в окружности появляются незаполненные промежутки. Можно было бы воспользоваться расчетом координат окружности, заданной в полярных координатах –  $R\cos\theta$ ,  $R\sin\theta$  путем пошагового изменения  $\theta$  от  $0^0$  до  $90^0$ . Но недостатки остаются все те же.

#### Восьмисторонняя симметрия

Процесс преобразования окружностей в растворную форму можно улучшить, если полнее использовать симметричные окружности. Поэтому удобно вычислить значения окружности на дуге в  $45^0$ , а потом симметрично отобразить их.

#### Алгоритм Брзенхэма для окружностей

Брзенхэм разработал пошаговый генератор дуг, который более эффективен, чем рассмотренные ранее. Необходимо сгенерировать только  $1/8$  часть окружности. Остальные части получаются путем отображения:

- 1-ой октанте – отражение относительно  $y=x$  1-го октанта;  $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
- 2-ой квадрант – отражение относительно  $x=0$  1-го квадранта;  $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$

- нижняя полуокружность – отражение относительно  $y=0$  верхней полуокружности  $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ .

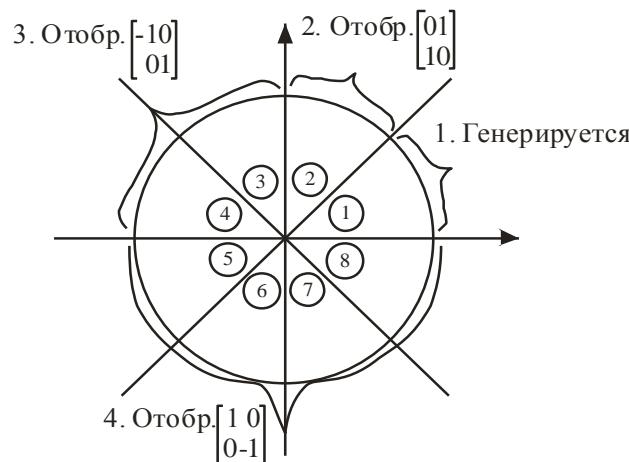


Рис. 3.8

Рассмотрим 1-ую четверть окружности с центром в начале координат. Начнем работу в т.  $x=0$ ,  $y=R$ , окружность генерируется по часовой стрелке,  $y$  – монотонно убывающая функция аргумента  $x$  до т.  $x = \frac{R}{\sqrt{2}}$ .

Для любой заданной т. на окружности существует только 2 возможности выбрать следующий  $\vec{p}$ , наилучшим образом приближающий окружность.

Алгоритм выбирает  $\vec{p}$ , для которого минимален квадрат расстояния между одним из этих пикселов и окружностью.

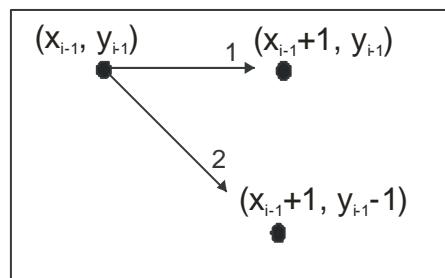


Рис. 3.9

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

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

Вводится упрощающая переменная  $d_i$ , значение которой можно вычислить в пошаговом режиме, используя лишь небольшое число сложений, вычитаний и сдвигов.

Возможно 7 способов прохождения истинной окружности через сетку. Пусть  $\vec{p}_{i-1}$  был выбран как ближайший к окружности при  $x = x_{i-1}$ . Теперь найдем, какой из  $\vec{p}$  ( $T_i$  или  $S_i$ ) расположен ближе к окружности при  $x = x_{i+1} + 1$ .

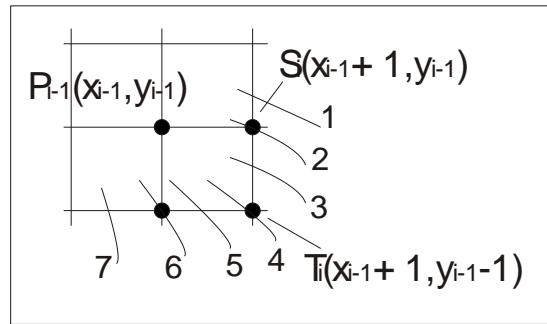


Рис. 3.10

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

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

Если  $|D(S_i)| \geq |D(T_i)|$ , то  $\vec{p}_i T_i$  ближе к окружности, чем  $S_i$ .

Если  $|D(S_i)| < |D(T_i)|$ , то  $\vec{p}_i S_i$  ближе к окружности.

Введем упрощающую переменную  $d_i$ :

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

Если  $d_i \geq 0 \rightarrow T_i$

$$d_i < 0 \rightarrow S_i.$$

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

$$\begin{aligned} D(S_i) &< D(T_i) \\ d_i &< 0 \rightarrow S_i. \end{aligned}$$

Для случая 3:  $D(T_i) < 0$   $(T_i$  - внутри)  
 $D(S_i) > 0$   $(S_i$  - снаружи)

$$\begin{array}{ll} \text{Для случаев 4,5:} & D(T_i) \geq 0 \\ & D(S_i) \leq 0 \end{array} \quad \begin{array}{l} (T_i - \text{снаружи}) \\ (S_i - \text{снаружи}) \end{array}$$

$$\begin{array}{l} D(S_i) > D(T_i) \\ d_i \geq 0 \rightarrow T_i \end{array}$$

После некоторых математических выводов было получено:

$$d_1 = 3 - 2R.$$

$$\begin{array}{ll} \text{если } d_i < 0 \ (\rightarrow S_i) & \text{если } d_i \geq 0 \ (\rightarrow T_i) \\ d_{i+1} = d_i + 4x_{i-1} + 6 & d_{i+1} = d_i + 4(x_{i-1} - y_{i-1}) + 10 \end{array}$$

### 3.5 Растворная развертка эллипсов

#### Простой метод

Эллипс — это сплюснутая или расплошнатая окружность.

Эллипс имеет 2 фокуса, сумма расстояния от любой т. эллипса до каждого фокуса = const.

Окружность — это вырожденный случай эллипса, когда фокусы совпадают.

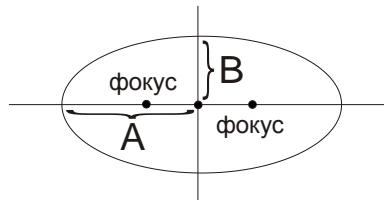


Рис. 3.11

Эллипс имеет 2 радиуса – А и В.

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\4 элемента. Вначале основная ось – X, X изменяется от 0 до т. P, в которой наклон касательной к эллипсу =  $45^0$ .

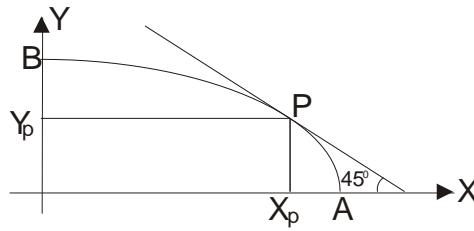


Рис. 3.12

После этого основная ось – Y.

Вначале  $x=0, y=B, \quad x=1, \quad y=y_1$ ,  
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  $x=1, \quad y = \left( \sqrt{B^2 - \frac{B^2}{A^2}} \right) Round$  и т.д. до тех пор, пока  $\frac{y^2}{B^2} \leq \frac{x^2}{A^2}$ . Далее у изменяется на 1 и вычисляется  $x$ .

### Инкрементивный метод

Уравнение элемента можно переписать:

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

Решим приведенное уравнение для  $x=0$  и  $y=B-0,5$ , что позволит оценить, как далеко  $x$  от того значения, при котором  $y=B-0,5$  ( $B-0,5$  – это т. перехода на следующий  $\vec{r}$  по оси Y). Потом пересчитывается уравнение для  $x+1$  и т.д.

Когда результат станет положительным, пора декрементировать  $y$ .  
Итак, установим  $x=0, y=B-0,5 \Rightarrow$

$$\begin{aligned} B^2 \cdot 0^2 + A^2 \cdot (B - 0,5)^2 - A^2 \cdot B^2 &= 0 \\ 0,25 \cdot A^2 - A^2 \cdot B &= 0 \end{aligned}$$

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 надо корректировать с каждым шагом по оси X, пока она не станет положительной, указывая изменить у. Это делается так. Для текущей точки  $= B^2 \cdot x^2$ . Для следующей т.  $= B^2 \cdot (x+1)^2 = B^2 \cdot x^2 + B^2 \cdot x + B^2$ .

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

Когда ошибка  $\geq 0$ , делается шаг по оси Y и пересчитывается ошибка для следующего шага. Для этого вместо у в уравнение эллипса подставляется  $y-1$ :

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

Т.к.  $A^2 \cdot y^2$  уже есть, то инкрементивная часть =

$$2A^2 \cdot y + A^2.$$

Рисование прекращается по достижению наклона касательной  $45^0$ , что определяется тем, что приращение по неосновной оси  $\geq$  приращению по основной оси.

### 3.6 Методы устранения ступенчатости растровых изображений

Пусть задан отрезок ненулевой толщины.

Какие  $\vec{r}$  выставить?

Те, которые более чем на половину покрыты отрезком.

Результат на рисунке  $\Rightarrow$

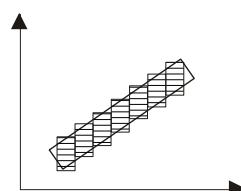


Рис. 3.13

#### Лестничный эффект

Избавится от него помогает принцип выравнивания: каждый  $\vec{r}$  высвечивается с яркостью, пропорциональной площади  $\vec{r}$ , занимаемой отрезком.

Но использование этого принципа замедляет процесс разложения в растр, т.к. требуются дополнительные алгоритмы.

Другой путь: ↑ разрешение экрана.

При  $\uparrow$  разрешающей способности экрана в 2 раза по осям X и Y общее число  $\vec{p}$   $\uparrow$  в 4 раза, а скорость разложения в растр – в 2 раза.

Рассмотрим 2 отрезка – горизонтальный и под углом  $45^0$ .

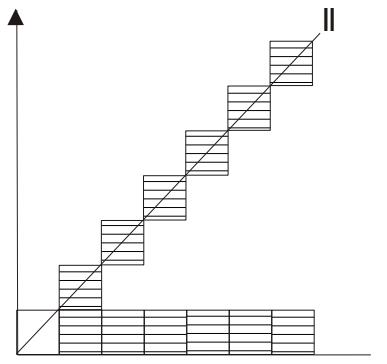


Рис. 3.14

Яркость отрезков I и II не одинаковая, хотя в их отрисовке участвуют по  $7 \vec{p}$ .

Отрезок I имеет максимальную яркость, а отрезок II – минимальную.

Следовательно, яркость отрезков зависит от их угла наклона. Чтобы сделать яркость отрезков одинаковой, надо менять интенсивность  $\vec{p}$ .

Так яркость  $\vec{p}$  в отрезке II=1

$$\text{яркость } \vec{p} \text{ в отрезке I} = \frac{1}{\sqrt{2}}.$$

### 3.7 Устранение искажений в растровых дисплеях

Рассмотрим мелкие объекты. Не используя принцип выравнивания, объекты I и III будут проигнорированы, т.к. не покрывают центр  $\vec{p}$ , а объект II будет выведен в виде целого  $\vec{p}$  с максимальной яркостью.

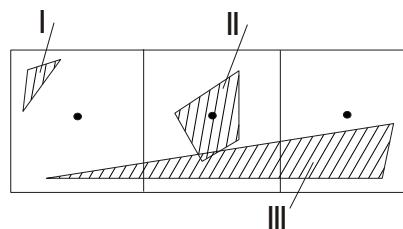


Рис. 3.15

При движении объекта (см. рис) будет наблюдаться мерцание. Кадр I, III – объекта нет (по той же причине), кадр II – объект есть.

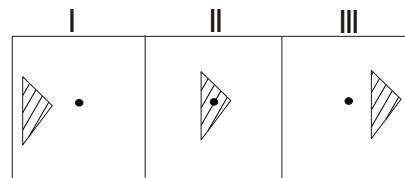


Рис. 3.16

В случае мелких объектов (соизмеримых с  $\vec{p}$ ) физический  $\vec{p}$  разбивается на несколько мнимых  $\vec{p}$  (разрешение увеличивается в 2 раза). Значение интенсивности физического  $\vec{p}$  определяется как усредненное значение интенсивностей мнимых  $\vec{p}$ .

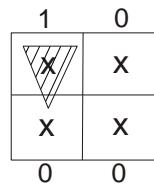


Рис. 3.17

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

Для цветных изображений:

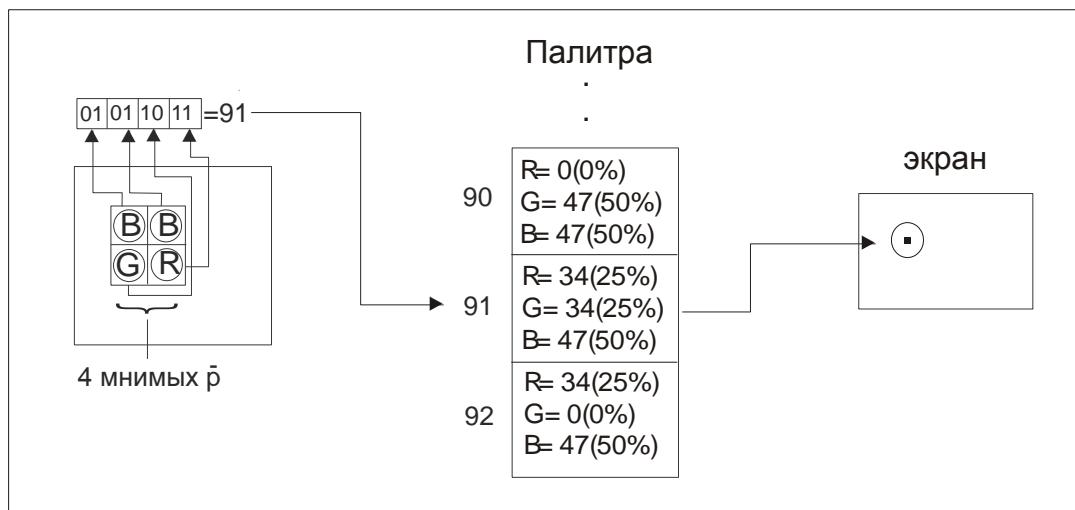


Рис. 3.18

Вместо 256 независимых цветов оценивается R-, G-, B- составляющие для каждого из 4 мнимых  $\vec{p}$ .

### 3.8 Сглаживание линий

Есть итальянская поговорка: “Необязательно говорить правду, главное, чтобы то, что вы скажите, хорошо звучало”.

Это относится и к сглаживанию линий; не нужно рисовать картинку идеально. Нужно, чтобы картинка лишь выглядела как настоящая. 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?

### Алгоритм Ву

Идея: при рисовании линий обычным образом с каждым шагом по основной оси высвечивается два  $\vec{p}$  по неосновной оси.

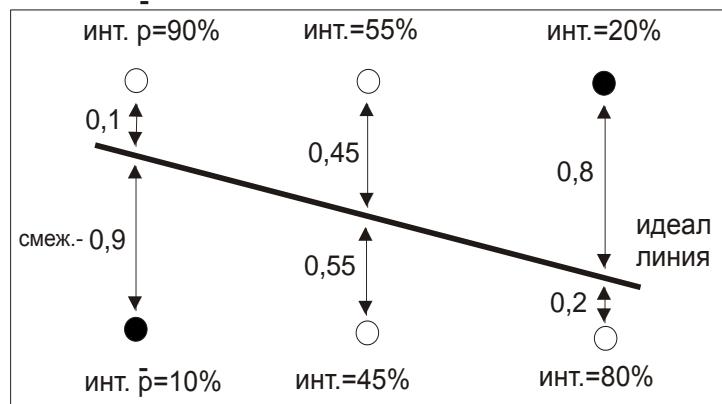


Рис. 3.19

Их интенсивность подбирается пропорционально расстоянию от центра  $\vec{p}$  до идеальной линии. Чем дальше т., тем  $<$  ее интенсивность. Значение интенсивности 2-ух  $\vec{p}$  дают в сумме 1, т.е. это интенсивность  $1 \vec{p}$ , в точности попавшего на идеальную линию. Это придает линии одинаковую интенсивность на всем ее протяжении, создавая иллюзию, что точки расположены вдоль линии не по 2, а по одной в точности по идеальной линии.

Горизонтальные, вертикальные и диагональные линии не требуют сглаживания. Для других линий алгоритм Ву проходит их вдоль основной оси, подбирая координаты по неосновной оси. Смещение вдоль неосновной оси вычисляется 1 целочисленным делением (для линий с наклоном  $< 1$  берется сам наклон, с наклоном  $> 1 - 1/2$  наклона). Это значение называется ошибкой смещения. Ошибка накопления ( $d$ ) показывает, как далеко ушли  $r\vec{p}$  от идеальной линии по неосновной оси, и как только она достигает критического значения, делается шаг на  $1 \vec{p}$  вдоль неосновной оси. Если основной осью является X, то будут установлены 2 т. с координатами  $(x,y)$  и  $(x,y+1)$ . Короче говоря, продвижение вдоль линии аналогично алгоритму Брезенхэма, только на каждом этапе устанавливается не  $1 \vec{p}$ , а 2. Осталось определить их интенсивность. Ошибка смещения суммируется с ошибкой накопления.

Пусть число уровней интенсивности кратно 2,

мин. интенсивность —  $2^n - 1$ ,

max интенсивность — 0.

Старшие  $n$  битов ошибки накопления покажут необходимую интенсивность для одного из  $\vec{p}$ . Интенсивность 2-го  $\vec{p}$  пары =  $(2^n - 1)$  - интенсивность  $\vec{p}_1$ . Ошибка накопления содержит соотношение расстояний от центра каждого из двух  $\vec{p}$  до идеальной линии.

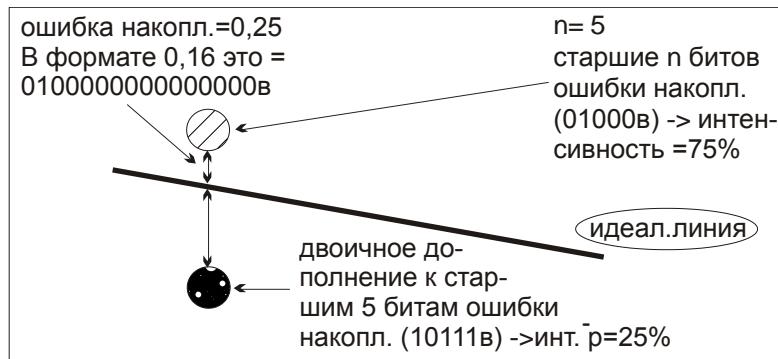


Рис. 3.20

### **3.9. Заполнение области**

## **Алгоритм построчного сканирования**

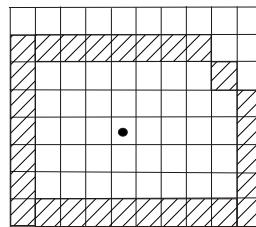


Рис. 3.21

Есть область, граница которой разложена в растр. Требуется: заполнить область внутри. Пусть внутри задана точка “boundary” — значение граничных  $\vec{p}$ . “old” — значение  $\vec{p}$  внутри области до заполнения. “new” — значение  $\vec{p}$  внутри области новое.

Объект заключается в прямоугольную оболочку и проверяется принадлежность  $\vec{p}$  объекту.

Проводится построчное сканирование. Находится  $\vec{p}$  со значением “boundary”,  $\vec{p}$ , следующий за ним и имеющий значение “old” меняется на “new” и так до тех пор, пока не будет встречен еще один  $\vec{p}$  со значением “boundary”. После этого осуществляется переход на следующую строку.

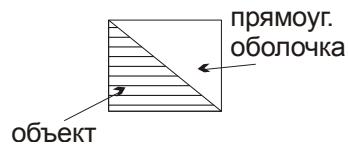


Рис. 3.22

Недостаток: Просматривается больше  $\vec{p}$ , чем необходимо.

### Метод заполнения с затравкой

Область называется 4-хсвязной, если из любой внутренней т. можно достичь любой другой т., двигаясь в одном из 4-х направлений.

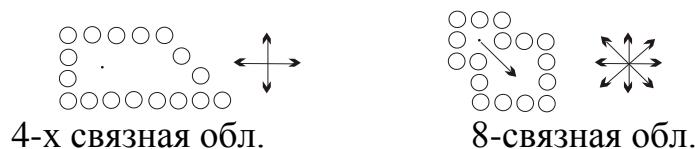


Рис. 3.23

### Алгоритм:

- 1) Поместить затравочный  $\vec{p}$  в стек.
  - 2) Пока стек не пуст:
    - a) извлечь  $\vec{p}$  из стека;
    - б) присвоить  $\vec{p}$  требуемое значение;
    - в) для каждого из соседних 4-хсвязных  $\vec{p}$  проверить:
      - является ли он граничным;
      - не присвоено ли  $\vec{p}$  требуемое значение.
    - г) проигнорировать  $\vec{p}$  в любом из этих двух случаев.
- иначе поместить  $\vec{p}$  в стек.

### Пример:

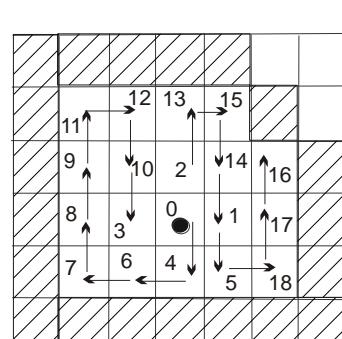
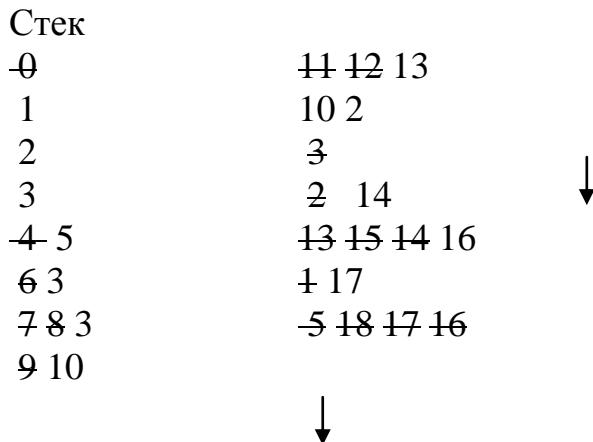


Рис. 3.24



### Заполнение линиями

Для заданной т.  $(x, y)$  определяется и заполняется максимальный отрезок, содержащий эту т. и лежащий внутри области. Потом в поисках еще не заполненных  $\vec{p}$  проверяются отрезки, лежащие выше и ниже. Если такие  $\vec{p}$  находятся, то функция рекурсивно вызывается дальше. Алгоритм эффективен и для областей с отверстиями.

### 3.10. Разложение в растр сплошных многоугольников

В отличие от заполнения области, где граница многоугольника (области) уже была разложена в растр, сейчас требуется получить растровый образ сплошной области, граница которой задана в векторном виде.

#### Когерентность сканирующих строк

В самом простом случае, чтобы определить  $\vec{p}$ , которые надо закрасить, нужна проверка каждого на принадлежность многоугольнику (все  $\vec{p}$  экрана).

Но такой подход очень трудоемок, приходится проделывать много лишних операций.

Упростить эту задачу можно, если использовать тот принцип, что соседние  $\vec{p}$  часто ведут себя одинаково, т.е. мы можем работать сразу с группой  $\vec{p}$ , включая их в изображение многоугольника или отбрасывая.

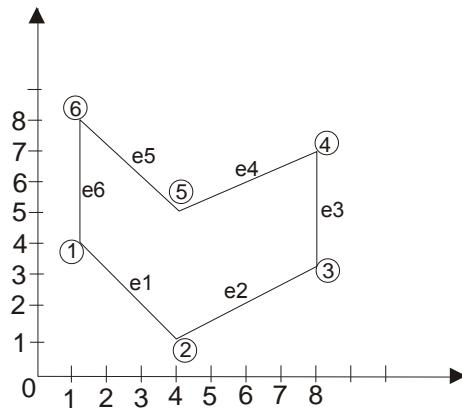


Рис. 3.25

На этом и основан принцип пространственной когерентности:

— перемещаясь от  $\vec{p}$  к  $\vec{p}'$  или от одной сканирующей строки к другой, многоугольник чаще всего остается постоянным.

Используя этот принцип, можно предположить следующий алгоритм:

- Определить точки пересечения текущей сканирующей строки со сторонами многоугольника.
- Сортировать т. пересечения по увеличению  $x$  – координаты.
- Попарно закрасить.

Рассмотрим строку 2:

- 1) т. пересечения: 3,6
- 2) сортировка: 3,6
- 3) закраска: 3-6

строка 3:

- 1) 2,8,8
- 2) 2,8,8  $\Rightarrow$  не верно!
- 3) 2,-8,8 – правая граница буфера

строка 6:

- 1) 8,6,3,1
- 2) 1,3,6,8
- 3) 1,-3,6,-8

строка 5:

- 1) 8,4,4,1
- 2) 1,4,4,8  $\Rightarrow$  верно!
- 3) 1,-4,4,-8

Чтобы алгоритм работал верно, во всех случаях поступают следующим образом:

Все вершины разбивают на 2 типа:

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

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

Тогда для строки 3: (ребро  $e_3$  укоротили на 1, появилась вершина 3<sup>1</sup>)

- 1) 2,8

- 2) 2,8  
3) 2,-8

### Когерентность ребер

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

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

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

Алгоритм: Создается таблица ребер (ТР), в которой все ребра сортируются по увеличению  $y$  – координаты. В ней содержатся все ребра и каждое ребро представлено в виде:

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

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

Каждое ребро представлено в виде:

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

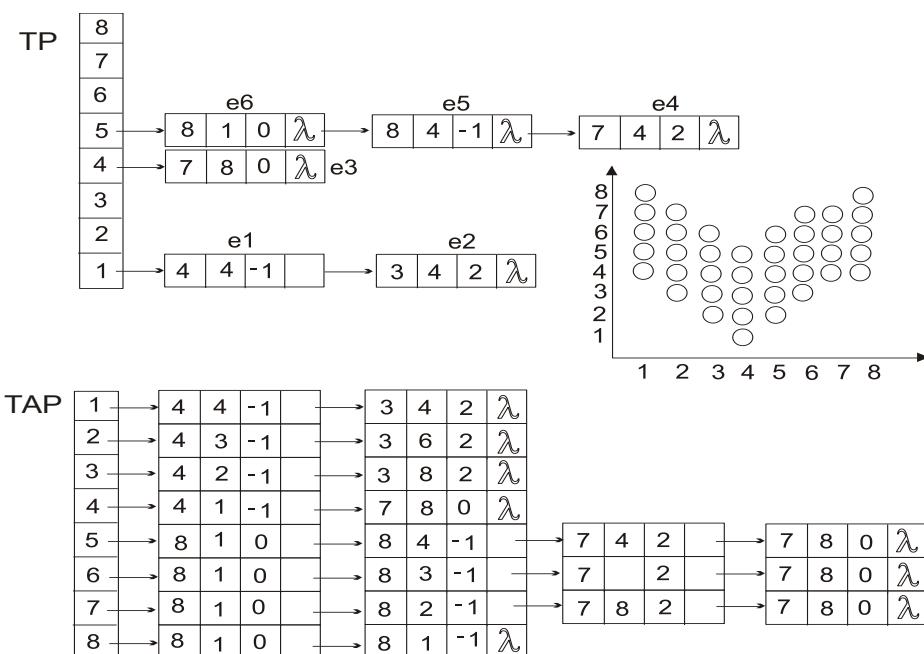


Рис. 3.26

### Алгоритм создания ТАР:

- 1) Установить  $y = \min$  – ому значению координаты  $y$  среди элементов ТР.
- 2) Инициализировать ТАР, сделать ее пустой.
- 3) Повторять шаг 3 до тех пор, пока ТАР и ТР не станут пустыми.
  - Слить информацию из группы  $y$  таблицы ребер (ТР) с информацией в ТАР, сохраняя упорядочивание по  $x$  – координате.
  - Занести желаемые значения в  $\vec{r}$  на сканирующей строке, определяемой текущим значением  $y$ , используя пары  $x$  – координат из ТАР.
  - Удалить из ТАР те элементы, которые  $y > y_{\max}$ .
  - Для всех элементов, содержащихся в ТАР, заменить  $x$  на  $x + \frac{1}{m}$ .
  - Т.к. на предыдущем шаге могла нарушиться упорядоченность ТАР по  $x$ , провести пересортировку ТАР.
  - Увеличить  $y$  на 1 и т.о. перейти к следующей сканирующей строке.

## 4. ОТСЕЧЕНИЕ ЛИНИЙ

### 4.1. Алгоритм Коэна-Сазерленда

В процессе работы с ГС пользователь формирует графические объекты произвольных размеров и сложности в СК наблюдателя. Используемые графические устройства имеют фиксированные границы (обычно прямоугольные). Иногда надо задать некоторую прямоугольную область на экране, которая определяет размеры желаемого изображения. Эта область 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.

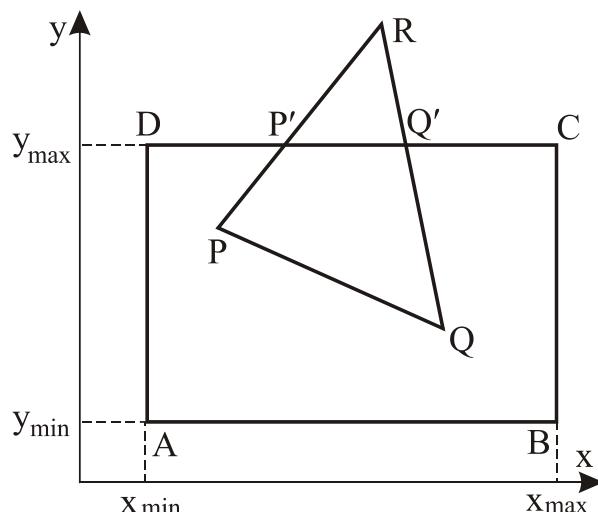


Рис. 4.1

Рассмотрим прямоугольник  $ABCD$ , который является окном (областью вывода). 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 надо вычислить, зная координаты точек  $P, Q$  и  $R$ . Наклон отрезка  $PR$  вычисляется следующим образом:

$$m = \frac{\Delta y}{\Delta x} = \frac{y_R - y_P}{x_R - x_P} = \frac{y_{P'} - y_P}{x_{P'} - x_P}$$

$$y_{P'} = y_{\max}$$

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

Мы рассмотрели ситуацию, когда  $P$  находится внутри окна, а  $R$  — удовлетворяет неравенствам:

$$x_{\min} < x_R < x_{\max}$$

$$y_R > y_{\max}$$

Однако необходимо рассмотреть значительно больше ситуаций. Большое разнообразие логических операций, которые надо выполнять для решения этой задачи, делают проблему отсечения линий очень интересной 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 четырехбитовый 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.

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 $y \leq y_{\max}$

$$b_{3,} = 1, \text{ если } y > y_{\max}$$

Возможны 9 из 16 битовые комбинации показаны на рисунке.

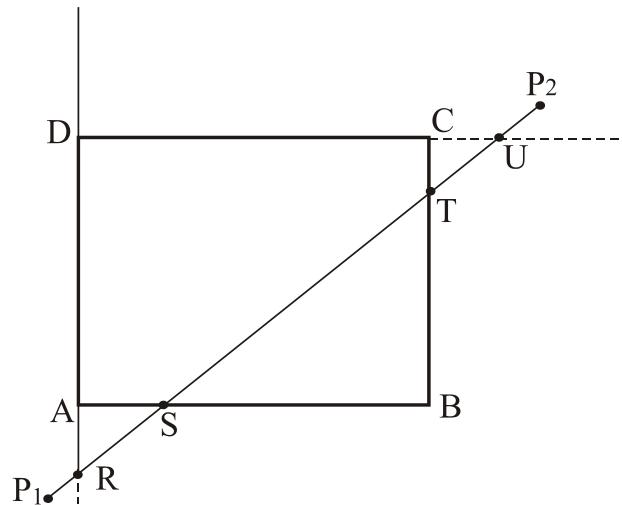


Рис. 4.2

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 начертен полностью.
- Если коды содержат единичный бит в одной и той же позиции, то отрезок целиком лежит за границами окна и не вычерчивается.

Остальные случаи рассмотрим подробнее. Если хотя бы 1 из кодов содержит единичный бит, то либо  $P_1$ , либо  $P_2$  перемещаются из области вне окна к одной из границ окна или к ее продолжению (точка  $P_1$  перемещается в точку  $R$ , точка  $P_2$  — в точку  $U$ ). В последнем случае точка по-прежнему будет находиться вне окна и понадобится еще одно перемещение (точка  $R$  переместиться в точку  $S$ , 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 (ST),

Пример.

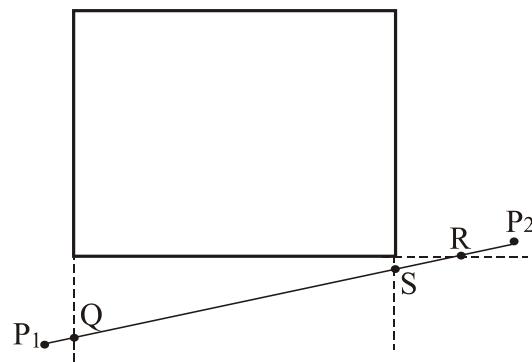


Рис. 4.3

$$cod(P_1) = 1010, cod(P_2) = 0100$$

Так как расположение отрезка не соответствует рассмотренным двум случаям, то проводится отсечение. Точка  $P_1$  перемещается в точку  $Q$ .

$$cod(Q) = 0010, cod(P_2) = 0100$$

Требуется дальнейшее отсечение,

$$\begin{aligned} P_2 &\rightarrow R, R \rightarrow S, \\ cod(Q) &= 0010, cod(R) = 0100, \\ cod(S) &= 0010 \end{aligned}$$

Теперь условие соответствует случаю 2, и обе точки находятся ниже окна. Следовательно, отрезок  $P_1 P_2$  (или его усеченный вид QR) чертить не надо.

#### 4.2. Алгоритм разбиения средней точкой

В предыдущем алгоритме надо было вычислить пересечение отрезка со стороной окна. Этого можно избежать, если реализовать двоичный поиск такого пересечения путем деления отрезка его средней точкой. Алгоритм был предложен Спруллом и Сазерлендом. Программная реализация его медленнее, но аппаратная быстрее и эффективнее, так как аппаратно сложение и деление на 2 очень быстры (деление на 2 эквивалентно сдвигу побитовому вправо: 6=0110, 3=0011).

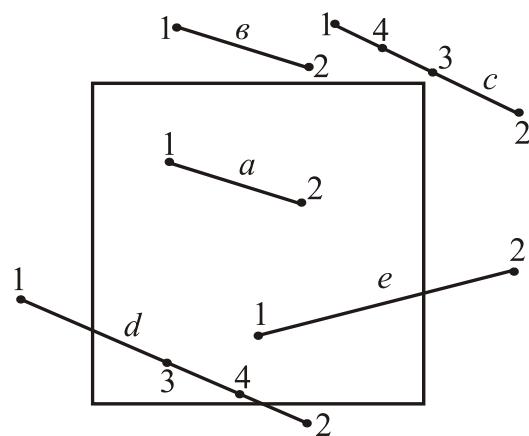


Рис. 4.4

В алгоритме используются коды концов отрезка и проверки, выявляющие полную видимость (отрезок  $a$ ) и тривиальную невидимость (отрезок  $b$ ). Осталь-

ные отрезки, не определяемые тривиально, разбиваются на 2 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 число разбиений пропорционально точности задания координат концов отрезка.

Рассмотрим отрезок  $c$ . Хотя он невидим, он пересекает диагональную прямую окна и не может быть тривиально отвергнут. Разбиение его средней точкой 3 позволяет тривиально исключать 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 обоих 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 также обрабатывается отрезок 1-3. Наиболее удаленной от точки 2 видимой точкой будет его пересечение с левой стороной окна.

Таким образом, для отрезков, подобных  $d$ , реализуется 2 двоичных поиска двух видимых точек, наиболее удаленных от концов отрезка. Это точка пересечения отрезка со сторонами окна. Для отрезков типа  $e$  один из двух поисков не нужен.

### 4.3 Трехмерное отсечение отрезков

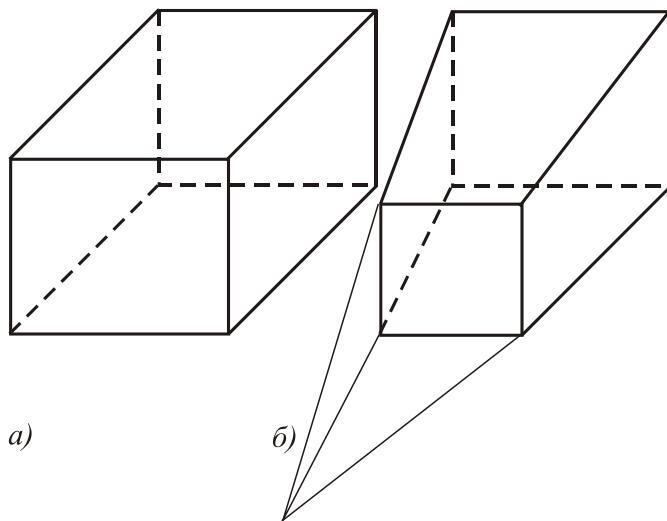


Рис. 4.5

Существуют 2 наиболее распространенные формы трехмерных отсекателей:

- Прямоугольный параллелепипед (см. рис *a*);
- усеченная пирамида. (см. рис *б*);

У каждой из них 6 граней. Для определения видимости отрезков обобщим алгоритм, используемый для двумерного случая. В трехмерном случае используется 6-битовый 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.

Вид усеченной пирамиды сверху:

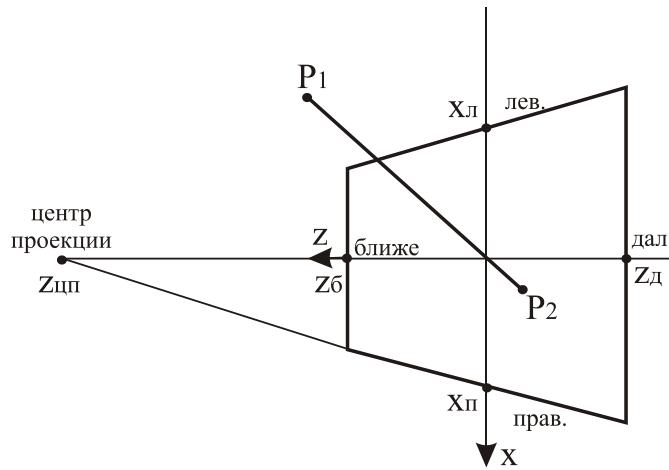


Рис. 4.6

Теперь легко найти уравнение прямой на плоскости  $XZ$ , несущей проекцию правой грани отсекателя:

$$x = \frac{(z - z_{un})\alpha_n}{(z_o - z_{un})} = z\alpha_1 + \alpha_2,$$

$$\text{где } \alpha_1 = \frac{x_n}{z_o - z_{un}}, \alpha_2 = -\alpha_1 z_{un}.$$

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:

$$f_n = x - z\alpha_1 - \alpha_2 \quad \begin{cases} >0, \text{ если } P \text{ справа от плоскости} \\ =0, \text{ если } P \text{ на плоскости} \\ <0, \text{ если } P \text{ слева от плоскости} \end{cases}$$

Аналогично находятся функции для других 5 граней.

$$f_n = x - z\beta_1 - \beta_2 \quad \begin{cases} >0, \text{ если } P \text{ справа от плоскости} \\ =0, \text{ если } P \text{ на плоскости} \\ <0, \text{ если } P \text{ слева от плоскости} \end{cases}$$

$$\beta_1 = \frac{x_n}{z_o - z_{un}}, \beta_2 = -\beta_1 z_{un}$$

$$f_e = y - z\gamma_1 - \gamma_2 \quad \begin{cases} >0, \text{ если } P \text{ выше плоскости} \\ =0, \text{ если } P \text{ на плоскости} \end{cases}$$

$<0$ , если Р ниже плоскости

$$\gamma_1 = \frac{y_n}{z_o - z_{un}}, \gamma_2 = -\gamma_1 z_{un}$$

$$f_u = y - z\delta_1 - \delta_2$$

$$\left\{ \begin{array}{l} >0, \text{ если } P \text{ выше плоскости} \\ =0, \text{ если } P \text{ на плоскости} \\ <0, \text{ если } P \text{ ниже плоскости} \end{array} \right.$$

$$\delta_1 = \frac{y_n}{z_o - z_{un}}, \delta_2 = -\delta_1 z_{un}$$

$$f_\delta = z - z_\delta$$

$$\left\{ \begin{array}{l} >0, \text{ если } P \text{ ближе к плоскости} \\ =0, \text{ если } P \text{ на плоскости} \\ <0, \text{ если } P \text{ дальше от плоскости} \\ >0, \text{ если } P \text{ ближе к плоскости} \\ =0, \text{ если } P \text{ на плоскости} \\ <0, \text{ если } P \text{ дальше от плоскости} \end{array} \right.$$

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

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

#### 4.4 Отсечение многоугольников

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

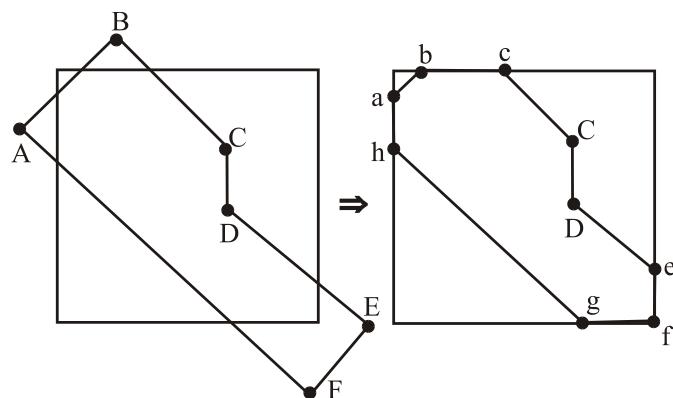


Рис. 4.7

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.

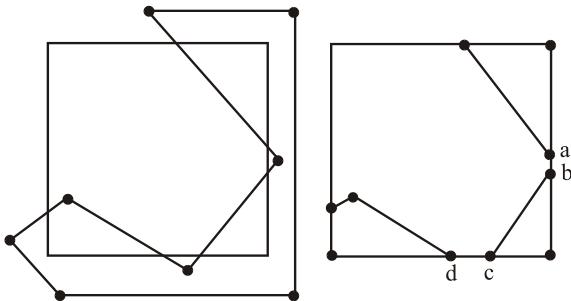


Рис. 4.8

Иногда отрезок  $ab$  и  $cd$  включается в описание результата. И если исходный многоугольник красный на синем фоне, то отрезок  $ab$  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 на рис.:

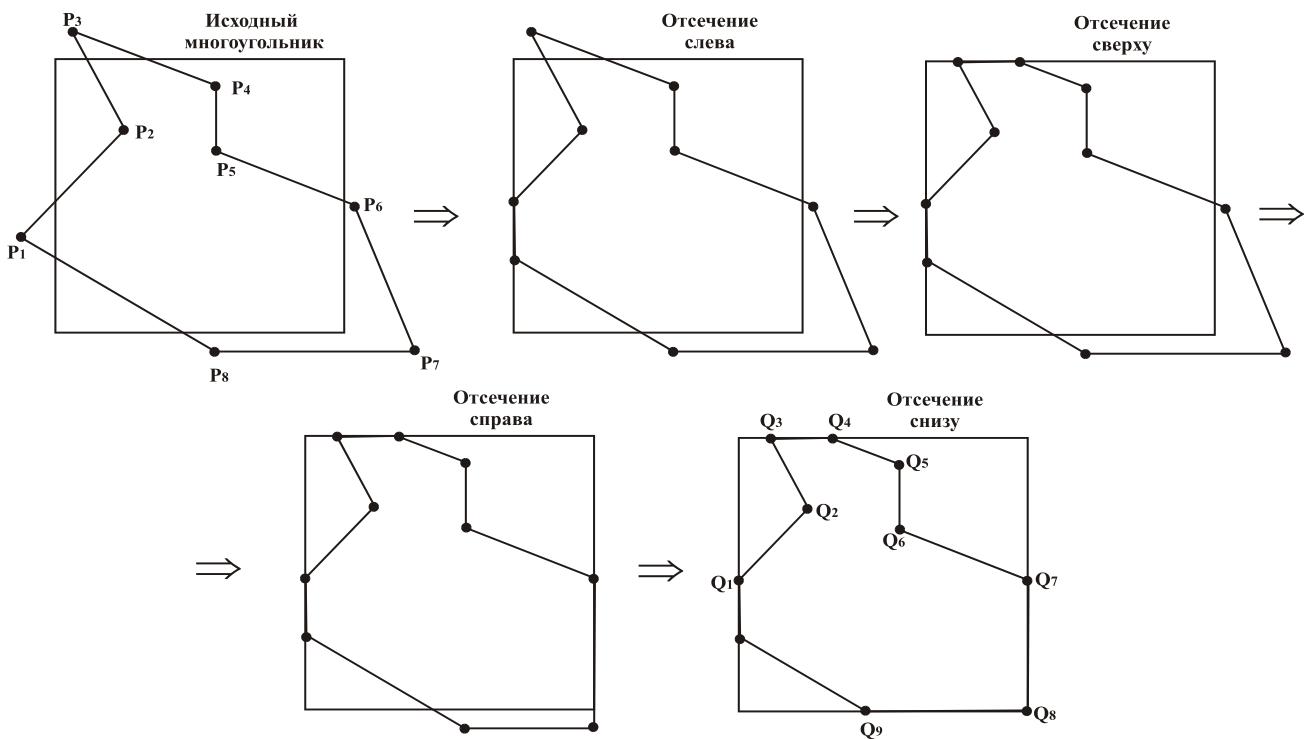


Рис. 4.9

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, то 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 ситуации расположения ребра и отсекающей плоскости:

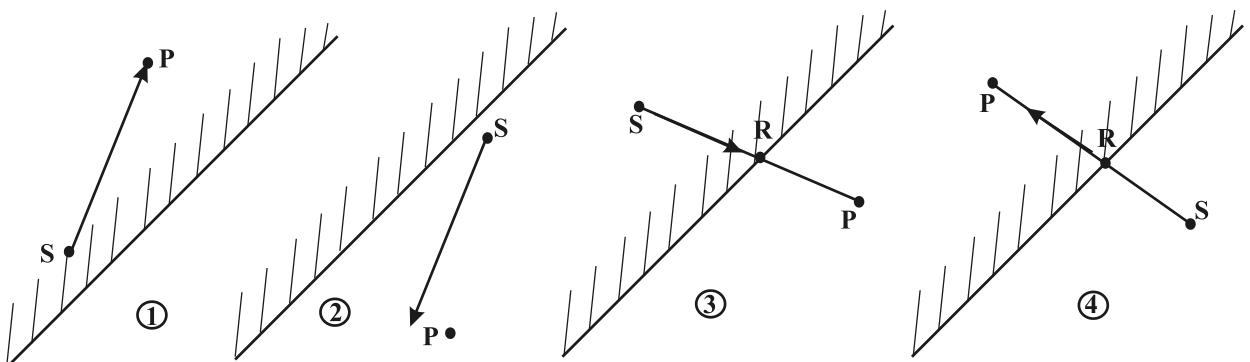


Рис. 4.10

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

- Полная видимость. Результат — вершина  $P$  (1 точка) (заносить в результат начальную точку  $S$  не надо, так как если вершины рассматриваются поочередно, то точка  $S$  уже была конечной точкой предыдущего ребра и уже попала в результат).
- Полная невидимость. Результат — 0 точек.
- Выход из области видимости. Результат — точка  $R$  (1 точка).
- Вход в область видимости. Результат — точки  $R, P$  (2 точки) (так как конечная вершина  $P$  видима, она тоже должна попасть в результат).

Для первой вершины многоугольника надо 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 или аппаратно. Они могут состоять из отдельных отрезков (штрихов) или быть образованными точечной матрицей.

Штриховые литеры, сгенерированные программно, можно обрабатывать как любые отрезки: поворачивать, переносить, масштабировать, отсекать по любым окнам, используя рассмотренные ранее алгоритмы.

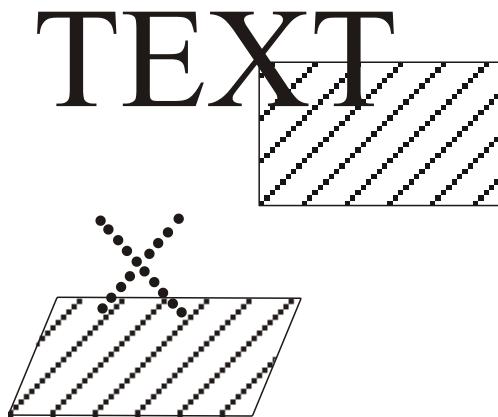


Рис. 4.11

Программно сгенерированные символы в форме точечной матрицы можно обрабатывать также. Если прямоугольная оболочка литеры пересекается с окном, то надо проверить, будет ли каждый  $\bar{p}$  маски символа находится внутри окна. В этом случае этот  $\bar{p}$  активизируется, иначе — нет.

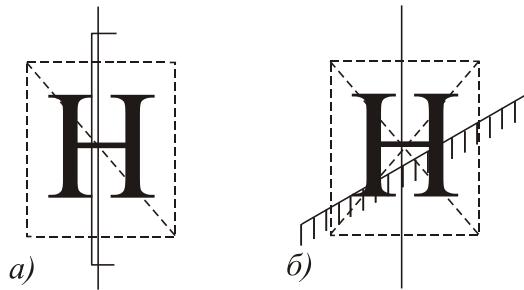


Рис.4.12

Литер складывается больше ограничений. Обычно любая не полностью видимая литера удаляется (для этого сравнивается прямоугольная оболочка литеры с границами окна). Если прямоугольная оболочка литеры ориентировано так же, как и граница окна (рис. а), то тест видимости можно провести только для одной из диагоналей оболочки. Если прямоугольная оболочка литеры ориентирована иначе, чем окно, то тесты видимости надо провести для двух диагоналей (рис. б).

## 5. ПРОЕКТИРОВАНИЕ ГРАФИЧЕСКОГО ДИАЛОГА

При проектировании графической интерактивной системы особое внимание надо уделить организации диалога.

Подобно архитектуре, проектирование интерфейса с пользователем является в какой-то мере искусством, а не наукой. В этом разделе мы рассмотрим 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.

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

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

### 5.1. Языковая аналогия

Существует полезная аналогия между диалогом пользователя с компьютером и человеческим общением. Конечно же язык машинной графики почти не содержит слов, которые говорят или пишут. «Словами» этого языка является кар-

тинки и действия (нажатие кнопок, выбор элементов селектором, позиционирование локатором). Тем не менее, многие атрибуты общения человека с человеком следует сохранить в графическом диалоге.

#### Основной принцип

Язык диалога должен быть языком пользователя, без излишней ориентации на компьютер, то есть надо обходиться простыми правилами, простым словарем и использовать понятия, которые пользователю уже знакомы или которые ему легко изучить (пример с биологом и программистом).

#### Требования к языку диалога

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

Рассмотрим теперь разговор 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. И то и другое является *формой обратной связи*.

Иногда говорящий может сделать ошибку, но тут же может исправить ее. Возможность исправлять ошибки важна также и в диалоге с компьютером.

## **5.2. Языковая модель**

При разработке интерфейса пользователя с компьютером существует два языка:

- *входной*: пользователь задает команды для компьютера, (язык выражается в действиях с различными диалоговыми устройствами);
- *выходной*: компьютер отвечает пользователю, (язык выражается графически с помощью примитивов и их атрибутов).

Язык включает четыре части:

- Концепция.
- Семантика.
- Синтаксис.
- Лексика.

Подробно эти части рассмотрим на примере модели проектирования обстановки в комнате.

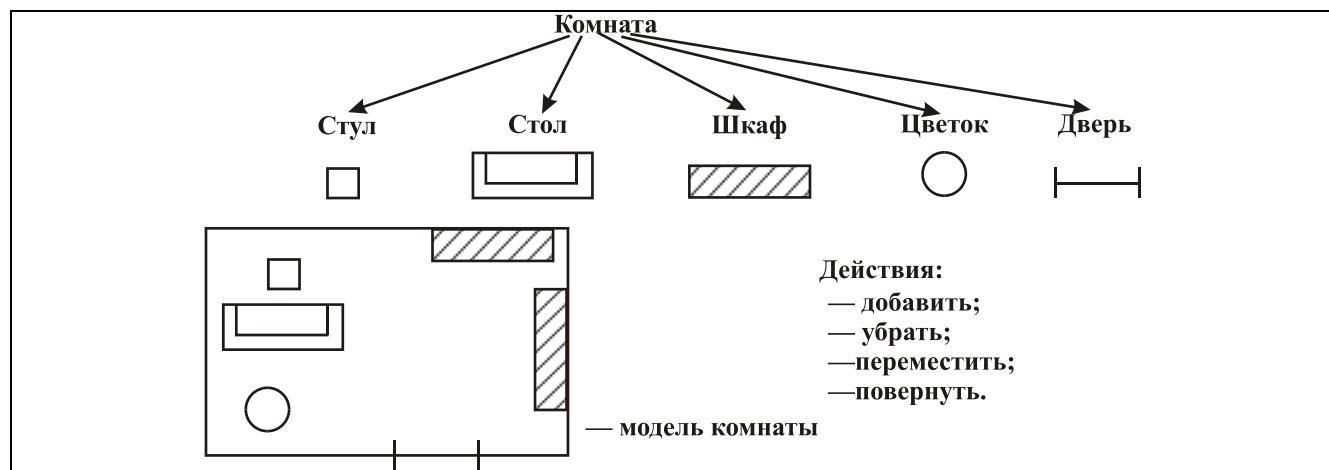


Рис. 5.1

При *концептуальном проектировании* определяется 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, удалении, перемещения, копирования; над объектами-файлами — операции создания, удалении, включения, переименования, копирования...

Концептуальная модель проектирования обстановки в комнате имеет один объект — комната и один класс объектов — предметы мебели. Связь между ними — комната содержит мебель. Действия — добавить, удалить, перенести, повернуть.

При *семантическом проектировании* подробно определяется 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: создать (AAD), удалить (ERASE), переместить (MOVE).

При *синтаксическом проектировании* определяется последовательность, в которой должны появляться элементарные единицы языка. Для входного языка эта последовательность является грамматикой: набор 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 добавить объект (ADD) представляет собой следующую последовательность действий:

- Ввести команду ADD.
- Выбрать добавляемый объект.
- Переместить объект в нужную позицию.

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 к планшету, как к средству определения позиции, выбора команды из меню, выбора объекта.

### 5.3. Принципы проектирования

#### 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 вам вряд ли захочется разговаривать, вы не имеете никакого подтверждения, что собеседник действительно вас слушает. То же самое относится и к общению с компьютером. Обратная связь является существенной компонентой диалога с компьютером, также как и диалога человека с человеком. Различие состоит в том, что в человеческом разговоре есть много источников обратной связи (жесты, мимика, глаза), графический же терминал не представляет почти никакой автоматической обратной связи (исключение составляет индикатор «включено»). Так что обратную связь необходимо разработать заранее.

Три уровня обратной связи:

- Лексическая.
- Синтаксическая.
- Семантическая.

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

Обратная связь на *синтаксическом уровне* возникает при вводе в систему каждого знака входного языка (команды, позиции, выбранного объекта): команда, выбранная из меню или объект, выбранные селектором для перемещения, могут выделяться изменением яркости, так что пользователь знает, что действия восприняты.

Также формами обратной связи на синтаксическом уровне является:

- приглашение на ввод следующего знака;

- подсвечивание кнопки программируемой функциональной клавиатуры, нажатой пользователем;
- повторение с помощью синтезатора слов входного языка, введенных голосом.

Другой формой обратной связи на синтаксическом уровне является 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, сохранить рисунок 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 более нескольких секунд, является извещение пользователя о том, что компьютер работает над его командой. Известны случаи, когда отсутствие такой обратной связи приводило к тому, что пользователи физически «расправлялись» с дисплеем и даже с самим создателем прикладной системы!

Такая обратная связь может принимать много форм, особенно удобен круг с вращающейся стрелкой, которая совершает полный оборот за время выполнения команды. По темпу продвижения стрелки пользователь может быстро оценить, успеет ли он выпить чашечку кофе.

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

Важно также и 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 место на экране для обратной связи отводить в том месте, куда смотрит пользователь, т.е. в месте нахождения курсора.

## **Помощь пользователю**

Многие графические системы создаются для широкого круга пользователей. Пользователей можно разделить на три категории:

- новички;
- пользователи средней опытности;
- опытные пользователи.

Как спроектировать интерактивную систему, удобную для работы любому пользователю?

Новичку сначала требуются направляемые компьютером «экскурсии» по возможностям системы и уроки основных команд.

Для реализации такой системы используются два метода:

- подсказка;
- запрос «помощи».

Подсказка говорит пользователю, какие действия он может предпринять в данный момент. Чем опытнее пользователь, тем менее он нуждается в подсказках, особенно если они делаются навязчивым образом, замедляя диалог. Поэтому не плохо, когда системы имеют несколько уровней подсказок, пользователь сам выбирает наиболее удобный. Неопытные 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 простой — вывод записи, указывающей, 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 надо задать позицию — высветить на экране следящее перекрестие, если надо ввести текст — сделать мигающим курсор и так далее. Такие ненавязчивые подсказки удобны для пользователя с опытом, а для новичков могут оказаться слишком сложными.

Кроме подсказки может использоваться и запрос «помоги», что 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 команды «помоги» система должна оказаться в том состоянии, в каком она была до вызова этой команды.

### **Возможность исправления ошибок**

Все мы совершаем ошибки во время работы с компьютером, и нуждаемся в простых и удобных средствах для их исправления, чтобы не наделать еще более серьезных ошибок.

Возможности исправления ошибок существуют на трех уровнях:

- Лексическом;
- Синтаксическом;
- Семантическом.

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

2. 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. (в AUTOCADe — клавиши «упр», «с»).

3. Менее обычной является возможность полностью ликвидировать результаты выполнения последней команды, даже если и произошли изменения в файлах..

В некоторых системах сохраняется несколько версий файлов, что частично упрощает проблему.

Создатель системы должен рассмотреть возможность восстановления каждого типа, учитывая при этом, что «стоимость» семантического восстановления может быть высока. Отсутствие средств для исправления ошибок угнетает пользователя и снижает его производительность. Их наличие толкает пользователя к 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 — пользователь может согласится с ее результатами или потребовать их ликвидации.



Рис. 5.2

Но при таком подходе возрастаёт число действий пользователя, затрачивается больше времени.

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

Представление средств для отмены команд требует дополнительных затрат на программирование. Альтернативой отмене является запрос у пользователя подтверждения на выполнение команд, которые сложно потом отменить. Удаление файла является наилучшим примером команд такого типа (в AutoCAD — команда QUIT, LAYER...).



Рис. 5.3

Существует множество примеров, когда отсутствие такого запроса приводило к печальным последствиям.

Подтвердить выполнения команды пользователь может, указав «Y» или «N» после ввода команды или путем нажатия два раза клавиши «ввод».

В одной популярной системе клавиша «RESET» была расположена над клавишей «RETURN». И случайное нажатие клавиши «RESET» очищало память компьютера.

Таким образом, при отсутствии возможности отмены команд запрос подтверждения являлся наилучшим средством предотвращения ошибок.

### **Управление временем отклика**

Чем должен руководствоваться программист при выборе значения времени отклика?

К сожалению, на этот вопрос нет единственного ответа. Обратимся опять к разговору человека с человеком: задавая вопрос о времени дня, вы ждете получить ответ в течение нескольких секунд, задержка в 30 с вызовет у вас раздражение. С другой стороны 30 с не такое уж большое время, если вы ждете ответа на вопрос, требующего сложных вычислений. Имеет значение и то, какие аппаратные средства мы используем.

*Вывод.* Чем сложнее задача, тем больше допустимое время отклика.

Системы с большим временем отклика заставляют пользователя бесполезно тратить время двояким образом: во-первых, на ожидание ответа от компьютера, во-вторых, получив ответ, он тратит некоторое время на восстановление прежнего хода мысли.

Быстрое время отклика должно быть в двух случаях:

— отклик на рефлекторные действия (это действия, выполняемые с помощью рефлекса — печать литеры или перемещение курсора; отклик должен быть немедленным, задержка между нажатием клавиши или перемещением курсора появление результата не должна превышать 100 мс);

— отклик на простой запрос (простыми называются запросы, которые не должны вызывать в компьютере сложных вычислений).

Каково предельное время отклика, при возрастании которого пользователь замечает, что имеет место задержка, удивляется медлительности компьютера и, следовательно теряет ход рассуждений? Около 2 с.

Двухсекундная задержка является допустимой для большинства систем, реализующих обратную связь на синтаксическом уровне, и для простых семантических действий.

Другим требованием, предъявляемым ко времени отклика является *постоянство*. Эксперименты показывают, что пользователь более продуктивно работает с более медленной системой, имеющей постоянное время отклика, чем с более быстрой системой, имеющей непостоянное время отклика. Предсказуемость предпочтительнее, чем изменчивость, по крайней мере, в некоторых пределах.

## Структуризация изображения

Пользователю предъявляется большое количество информации, которую надо структурировать. Для этого полезно разделить экран на несколько областей и в каждой области показывать информацию определенного типа. Подсказки, сообщения об ошибках, меню, графическая информация может 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 может улучшить, используя различные способы визуального кодирования:

- цвет;
- тип линии;
- яркость.

Рассмотрим более подробные данные о способах кодирования графической информации.

Таблица 5.1

Метод кодирования	Максимальное число кодов для безошибочного распознавания
Цвет	6
Геометрические формы	10
Ширина линии	2
Тип линии	5
Яркость	2

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

## 5.4. Процесс проектирования

Сначала надо определить прикладную область и будущих пользователей системы. Это может показаться банальным, но нередко этот этап выполняется недостаточно хорошо. Надо изучить методы, применяемые в данной проблемной области, пронаблюдать, что обычно делает человек во время решения задач.

То есть *основной принцип* проектировщика: «познай пользователя!». Надо наблюдать, изучать, общаться с будущим пользователем, пытаясь понять их образ мышления, и почему они делают то, что они делают.

Но это не означает, что графическая система должна имитировать способ работы пользователя вручную. С помощью компьютерной графики часто можно существенно улучшить методы работы.

Процесс изучения прикладной области и пользователя называется проектирование «сверху вниз» в соответствии с языковой моделью. Таким образом, проектировщик должен:

- сформулировать концептуальную модель системы;
- спроектировать семантику — операции, выполняемые над объектами; изменения, которые они вызывают; информацию, показываемую на экране;
- определить синтаксис — последовательность знаков, необходимых для каждой операции; организацию изображения на экране;
- выполнить лексическое проектирование.

Опыт показывает, что даже тщательно продуманные проекты оказываются после реализации не вполне соответствующими требованиям. Поэтому необходимо придерживаться при проектировании *принципа модульности*, чтобы легче потом было внести изменения.

Проектирование удобного интерфейса может занять много времени, но результаты оправдывают затраченные усилия. Например, создание интерфейса с пользователем в системе Star фирмы Xerox потребовало 20 – 30 человеко-лет. Результатом явился один из наиболее лучших интерфейсов в истории интерактивных систем.

## 6. ГЕОМЕТРИЧЕСКОЕ МОДЕЛИРОВАНИЕ. ОБЩИЕ СВЕДЕНИЯ.

### 6.1. Геометрическая модель

При решении большинства задач в области автоматизированного конструирования (К) и технологической подготовки производства (ТПП) надо иметь модель объекта проектирования.

Под моделью объекта понимают его некоторое абстрактное представление, удовлетворяющее условию адекватности этому объекту и позволяющее осуществлять его представление и обработку с помощью компьютера.

Т.о. модель – набор данных, отображающих свойства объекта и совокупность отношений между этими данными.

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

Для обработки модели в графических системах САПР существенным является не весь объем информации об объекте, а та часть, которая определяет его геометрию, т.е. формы, размеры, пространственное размещение объектов.

Описание объекта с точки зрения его геометрии называется геометрической моделью объекта.

Но геометрическая модель может в себя включать еще и некоторую технологическую и вспомогательную информацию.

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

В традиционном процессе конструирования обмен информацией осуществляется на основе эскизных и рабочих чертежей с использованием нормативно-справочной и технической документации. В САПР этот обмен реализуется на основе внутримашинного представления объекта.

Под геометрическим моделированием понимают весь многоступенчатый процесс – от верbalного (словесного) описания объекта в соответствии с поставленной задачей до получения внутримашинного представления объекта.

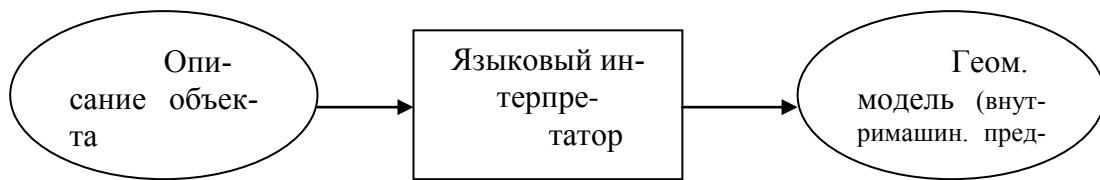


Рис. 6.1

В системах геометрического моделирования могут обрабатываться 2-мерные и 3-хмерные 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.

## 6.2. Основные виды ГМ

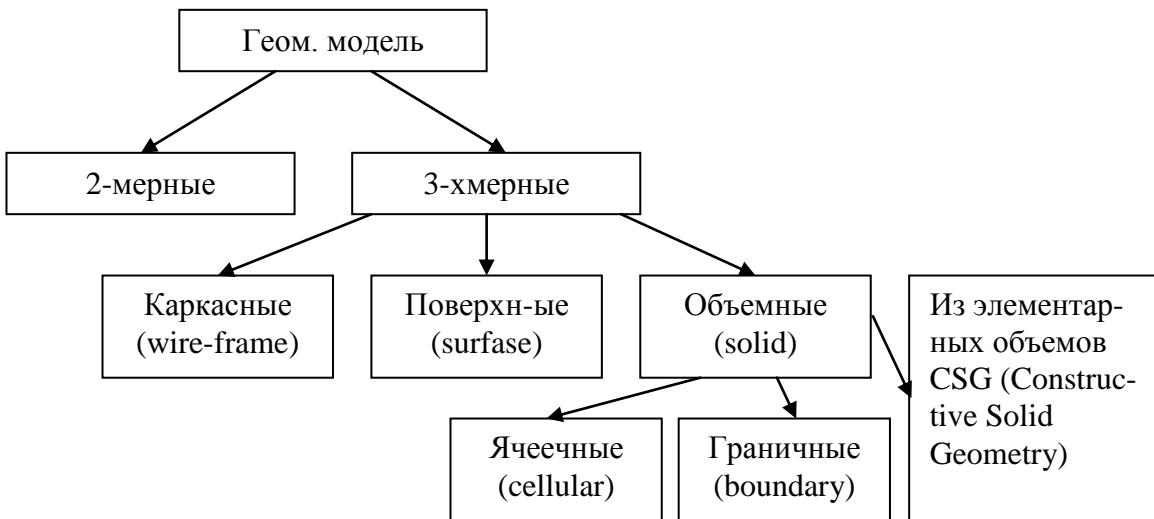


Рис. 6.2

2-мерные модели, которые позволяют формировать и изменять чертежи, были 1-ми моделями, нашедшими применение. Такое моделирование часто применяется и до сих пор, т.к. оно намного дешевле (в отношении алгоритмов, использования) и вполне устраивает промышленные организации при решении разнообразных задач.

В большинстве 2-мерных 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 2-мерное представление часто не удобно для достаточно сложных изделий. При традиционных способах конструирования (без САПР) пользуются чертежами, где изделие может быть представлено несколькими видами. Если изделие очень сложное, его можно представить в виде макета. 3-хмерная модель служит для того, чтобы создать виртуальное представление изделия во всех 3-х измерениях.

Различают 3 вида 3-хмерных моделей:

- каркасные (проводочные)
- поверхностные (полигональные)
- объемные (модели сплошных тел).

- Исторически 1-ми явились каркасные модели. В них хранятся только координаты вершин ( $x, y, z$ ) и соединяющие их ребра.

На рисунке видно, как куб может быть воспринят неоднозначно.

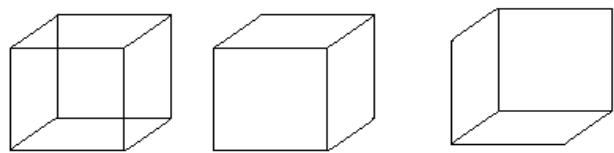


Рис. 6.3

Т.к. 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.

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

При построении поверхностной модели предполагается, что объекты ограничены поверхностями, которые отделяют их от окружающей среды. Поверхность объекта тоже становится ограниченной контурами, но эти контуры являются результатом 2-х 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 задания поверхностей (плоскости, поверхности вращения, линейчатые поверхности). Для сложных поверхностей используются различные математические модели аппроксимации поверхностей (методы Кунса, Безье, Эрмита, В-сплайна). Они позволяют изменять характер поверхности с помощью параметров, смысл которых доступен пользователю, не имеющему специальной математической подготовки.

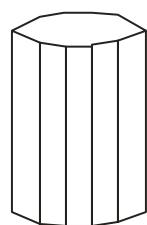


Рис. 6.4

Аппроксимация поверхностей общего вида плоскими гранями дает преимущество: для обработки таких поверхностей используются простые математи-

ческие методы. Недостаток: сохранение формы и размеров объекта зависит от числа граней, используемых для аппроксимаций. Чем  $>$  число граней, тем  $<$  отклонение от действительной формы объекта. Но с увеличением числа граней одновременно увеличивается и объем информации для внутримашинного 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 настоящее время известны следующие способы построения объемных моделей:

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

Структура может быть усложнена внесением действий переноса, поворота, масштабирования.

#### Достиныства:

- гарантия генерации правильной модели,
- большие возможности моделирования форм,
- быстрый и эффективный доступ к геометрической информации (например, для прорисовки).

#### Недостатки:

- больший объем исходных данных, чем при CSG способе,
- модель логически  $<$  устойчива, чем при CSG, т.е. возможны противоречивые конструкции,
- сложности построения вариаций форм.

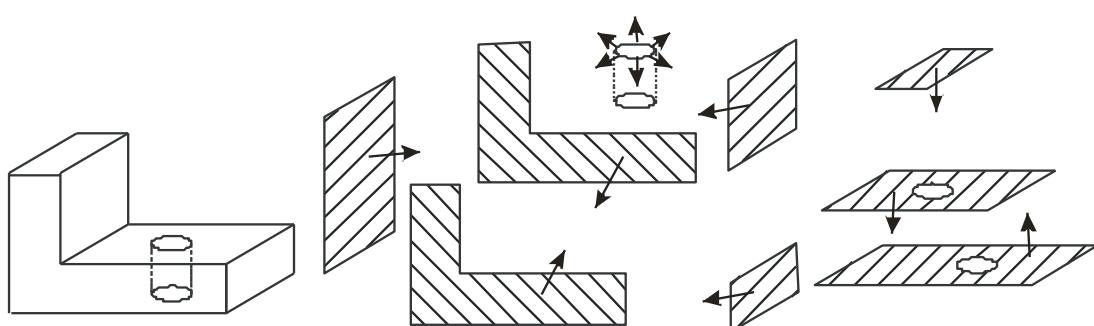


Рис. 6.5

- 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 древовидная структура. Узлы (нетерминальные вершины) – операции, а листья – элементарные объемы.

Достоинства:

- концептуальная простота,
- малый объем памяти,
- непротиворечивость конструкции,
- возможность усложнения модели,
- простота представления частей и сечений.

Недостатки:

- ограничение рамками булевых операций,
- вычислительноемкие алгоритмы,
- невозможность использовать параметрически описанных поверхностей,
- сложность при работе с функциями > чем 2-го порядка.

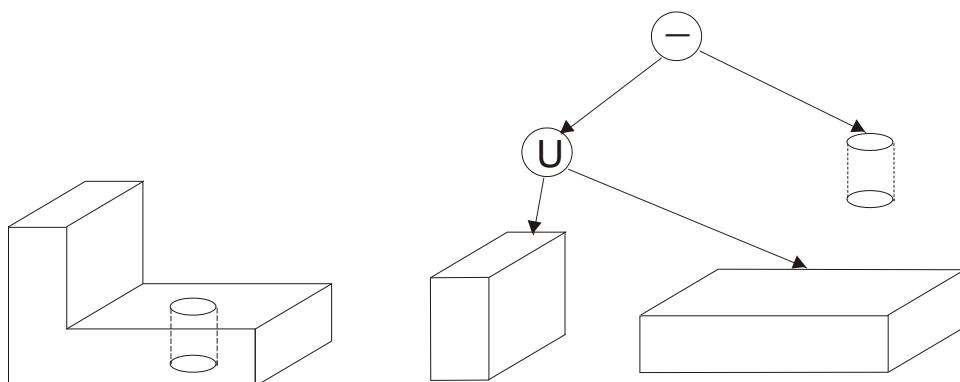


Рис. 6.6

- Ячеичный метод. Ограниченный участок пространства, охватывающий весь моделируемый объект, считается разбитым на большое число дискретных кубических ячеек (обычно единичного размера).

*МОДЕЛИРУЮЩАЯ СИСТЕМА ДОЛЖНА ПРОСТО ЗАПИСАТЬ ИНФОРМАЦИЮ О ПРИНАДЛЕЖНОСТИ КАЖДОГО КУБА ОБЪЕКТУ.*

Структура данных представляется 3-хмерной матрицей, в которой каждый элемент соответствует пространственной ячейке.

Достоинства:

- простота.

Недостатки:

- большой объем памяти.

Для преодоления этого недостатка используют принцип разбиения ячеек на подъячейки в особо сложных частях объекта и на границе.

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

Каркасное представление м.б. использовано не при моделировании, а при отражении моделей (объемных или поверхностных) как один из методов визуализации.

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

Целью геометрического моделирования является представление объектов. Эти объекты являются реальными и должны удовлетворять целому ряду требований.

Считается, что модель тем лучше, чем она учитывает ограничений, связанных с реальным объектом, его изготовлением и использованием. Так, например, 2-мерная 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 же, которая позволяет в любой момент рассматривать представленные в ней объекты как сплошные тела, считается хорошей моделью с точки зрения геометрии.

#### Требования при геометрическом моделировании высокого уровня:

- 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 (ЧПУ, расчет конструкций, МКЭ).

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

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

### **6.4. Внутреннее представление, типы данных**

Одно из первых решений, которое д. принять разработчик систем, состоит в выборе представления либо с помощью структур данных, либо с помощью процедурной формы. Преимуществом процедурного представления является компактность и меньшая избыточность, чем у структур данных, и оно в определенной степени обеспечивает непротиворечивость информации. Но далеко не всегда можно дать ответ типа “либо то, либо другое”. Так, например, параметрическая форма может существовать одновременно и в виде вычисленных значений, и в процедурном виде.

При определении модели очень важно указать, какими данными эта модель должна управлять. Можно выбрать модель, очень близкую к представлению при визуализации (отрезки и дуги – для 2-мерной модели, ребра и вершины – для 3-хмерной модели). Такое решение ограничивает диапазон возможностей модели. Обычно предпочитают модели, которые позволят легко выполнять традиционные функции (простановка размеров, геометрические преобразования, разрезы, удаление скрытых частей).

Данные, которые хранятся в модели, также сильно различаются в зависимости от требуемого качества моделирования. Рассмотрим, какого рода данные применяются в каждом типе моделей.

### **Двумерная модель**

В более простых моделях удовлетворяются тем, что 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 (например, построение окружности, касательной к заданным прямым)).

Структуры 2-мерных моделей весьма разнообразны и зависят от области применения: в машиностроении удовлетворяются информацией, близкой к чистой графике; в схемотехнике можно оперировать символами и их соединениями.

### **Каркасная модель**

Хотя эта модель и 3-хмерная, она имеет мало возможностей. В ней хранится информация 2-х типов:

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

### **Поверхностная модель**

В моделях этого типа хранится только описание поверхностей. Однако эти поверхности могут сильно отличаться.

### **Объемная модель**

В такой модели хранится информация, позволяющая отличать материал от пустоты (при этом пустота может рассматриваться как особый вид материала). В настоящее время обычно используются 2 метода:

— объект представлен в модели охватывающей его “оболочкой”. Тогда, как и в каркасной модели, сохраняется информация топологического и геометрического типов, но она более полная (границы заданы и ориентированы т.о., 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 моделей. Во многих системах существует геометрическая модель весьма высокого уровня – объемная или поверхностная и модель для визуа-

лизации, которая дает возможность работать с информацией, близкой к чисто графической.

## 7. ДВУМЕРНОЕ МОДЕЛИРОВАНИЕ

### 7.1. Типы данных

Двумерная геометрическая модель оперирует данными следующих типов:

- геометрические (координаты точек, уравнения прямых, окружностей и т.д.)
- топологические (отрезок, соединяющий 2 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 А касается элемента В)

### 7.2. Построение базовых элементов

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

#### **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 применения. И в соответствии с этим вводятся параметры элементов.

Например, отрезок может быть задан двумя точками. Точка п.с. пару координат (x и y); окружность — т. центра и радиус и т.д.

Пример - AutoCad.

#### **С помощью уравнений**

Базовые элементы задаются с помощью уравнений. Общее решение построения можно получить в 2 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, т.к. для добавления нового ограничения достаточно написать соответствующие уравнения.

Недостаток — 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

- 1) Проходит через  $n$  точек
- 2) Касается  $n$  объектов
- 3) Параллельно другому объекту
- 4) Образует некоторый угол с объектом
- 5) Отстоит от другого объекта на некотором расстоянии

Достоинство этого способа состоит в том, что не приходиться прибегать к очень сложным методам вычислений. Поиск решений полностью управляем. Кроме того можно организовать библиотеку подпрограмм для каждого применения.

Недостаток — для добавления нового ограничения или нового типа элемента надо писать новые подпрограммы.

Пример — построить окружность, касательную к заданным прямой и окружности, если известны  $R$  искомой окружности и примерное положение ее центра.

В зависимости от расположения заданных прямой и окружности, искомая окружность может размещаться следующим образом:

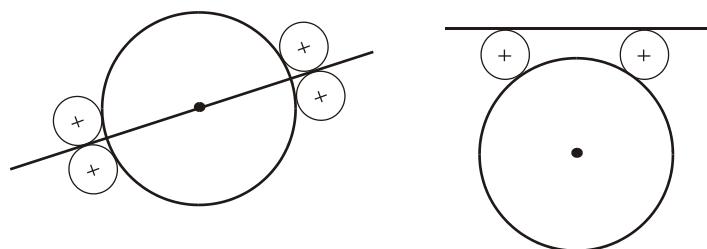


Рис. 7.1

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.

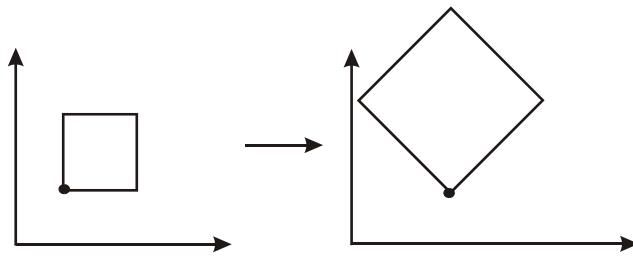


Рис. 7.2

Преобр=S (2,2)·R (45°)

### 7.3. Примеры моделей

#### Техническое черчение

Под техническим черчением в данном случае понимается использование методов, аналогичных тем, которые традиционно применяются чертежниками, но с использованием средств информатики (дисплеи, средства диалога). Соответствующее ПО 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 1 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

Этот способ позволяет представить линейный чертеж в детализированном виде на клетчатой поверхности. В результате дискретизации кривая описывается последовательностью коротких элементарных векторов, ориентированным по восьми направлениям.

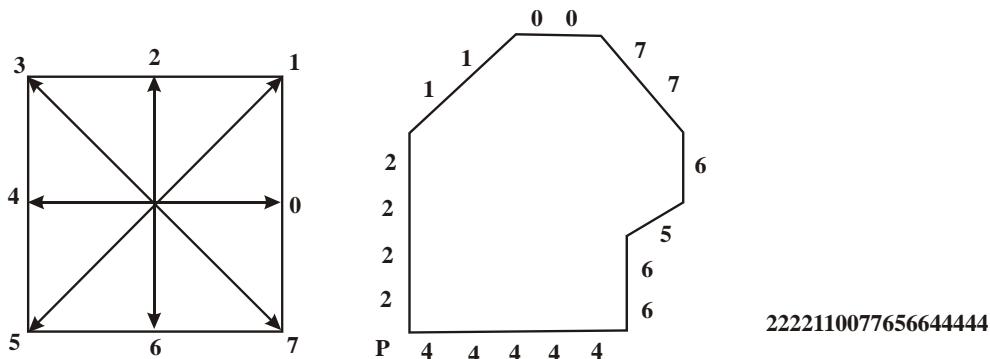


Рис. 7.3

Цепь п.с. упорядоченную последовательность кодов векторов.

$$\text{Длина вектора} = T(\sqrt{2})^p$$

Угол наклона вектора =  $n \cdot 45^\circ$ , где

$n$  — номер кода (от 0 до 7)

$$T — \text{размер ячейки сетки}, p = \begin{cases} 0, & \text{если } n - \text{четн.} \\ 1, & \text{если } n - \text{нечетн.} \end{cases}$$

Т.о., имея цепочку кодов, можно генерировать изображение. Для уменьшения длины цепочки кодов используются операторы повторения одинаковых элементарных векторов.

4·22·12·02·7652·65·4

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). Его главный недостаток, ограничивающий его применение то, что его примитивы принадлежат к слишком низкому уровню.

## 8. ТРЕХМЕРНОЕ МОДЕЛИРОВАНИЕ

### 8.1. Типы данных

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

## **Базовые элементы:**

1) Элементы нулевого уровня, то есть двумерные элементы (точки, отрезки, окружности, дуги, кривые, контуры).

2) Элементы первого уровня, то есть поверхности (плоскости, линейчатые поверхности, поверхности вращения, криволинейные поверхности).

3) 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, построение объемной модели может быть осуществлено двумя методами:

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

## **Представление с помощью границ**

Для простоты рассмотрим случай представления объекта в виде совокупности плоских граней, ограниченных ребрами, которые в свою очередь ограничены вершинами.

В этом случае используются данные трех типов:

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

Топологические и геометрические данные, как правило, не смешивают. В их разделении есть свои достоинства. Например, если надо перенести объект, то координаты вершин умножают на матрицу переноса. Топология объекта остается без изменения.

Рассмотрим первую модель, достаточно низкого уровня, так как хранятся только ребра и вершины (каркасная модель):

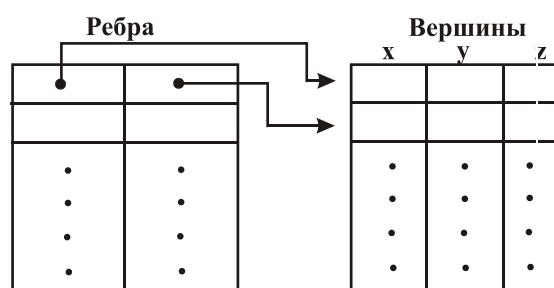


Рис. 8.1

Рассмотрим вторую модель, более высокого уровня. В ней топологические и геометрические данные полностью разделены. Данна также информация о связи

между ребрами, связи между гранями и уравнения граней, что дает значительные дополнительные возможности при работе с такой моделью:

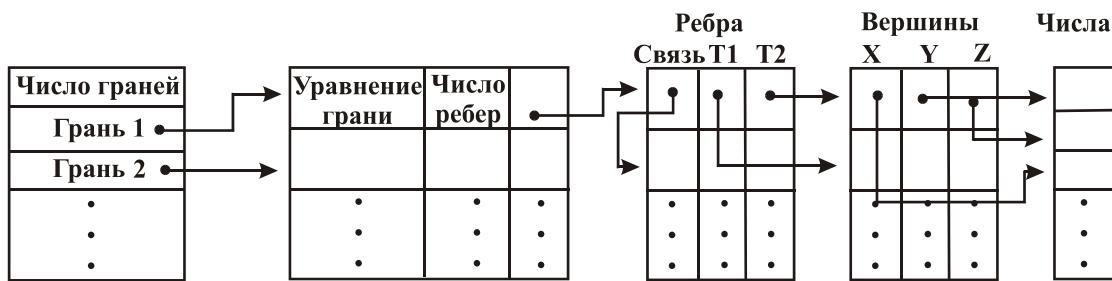


Рис. 8.2

### Представление с помощью дерева

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

Обычно используются следующие операции:

- объединение;
- пересечение;
- вычитание.

Упрощенным вариантом этой модели может быть случай, когда надо разместить некоторое число тел, что часто случается при архитектурном проектировании, при выполнении планировки цеха. Тогда модель можно представить следующим образом:

## 8.2. Методы описания трехмерных объектов

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

Возможны следующие методы описания трехмерных объектов:

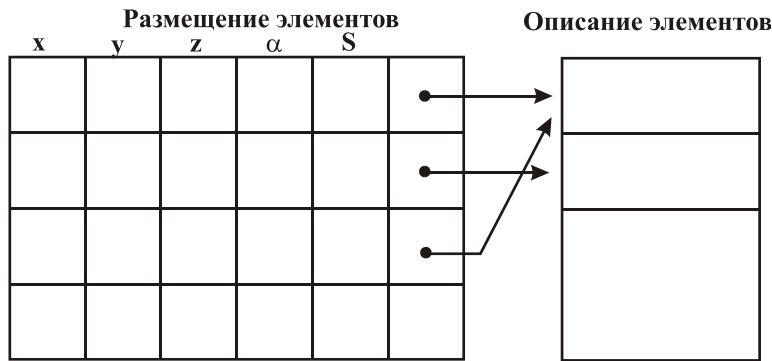


Рис. 8.3

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

В большинстве систем, оперирующих с 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 их соединения, установку системы координат и так далее. Формализованный язык может содержать не только геометрические параметры, но и информацию конструктивного и технологического плана (простановка размеров, шероховатость поверхностей, отклонения, допуска ...).

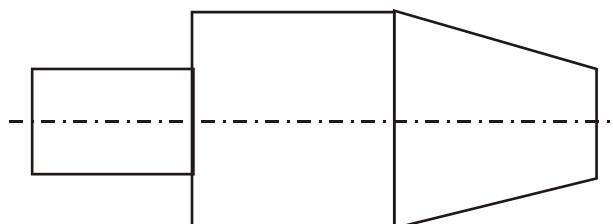


Рис. 8.4

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 — пользователю надо освоить специфику используемого формализованного языка.

### **Описание объекта в режиме графического диалога**

Это возможно только при наличии соответствующих технических средств — дисплеев для генерации динамических изображений и устройств ввода графической информации.

Пример. Создание в AutoCade библиотеки КЭ с помощью блоков. Конструирование осуществляется в диалоге путем соединения КЭ с нужными размерами.

### **Получение модели объекта путем ввода эскизов и восстановлением модели по имеющимся проекциям**

Этот метод соответствует традиционным методом конструирования и осуществляется в два этапа:

- 1) ввод эскизов;
- 2) восстановление модели;

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

- 1) Пц для обработки эскизов (осуществляет ввод эскизов и на их основе формирует точный контур);
- 2) Пц восстановления (создает проекции и по ним генерирует объемную модель);
- 3) Пц генерации изображения (осуществляет графический вывод точного 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. Система распознавания образов разделяет

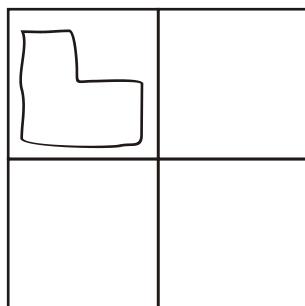


Рис. 8.5

эскиз проекции на контурные элементы. Эти элементы относятся аналитически описываемым (отрезок, окружность, дуга). Основой для такого распознавания яв-

ляется 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.

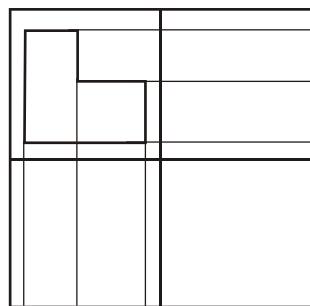


Рис. 8.6

Этап 3. Ввод эскизов остальных проекций.

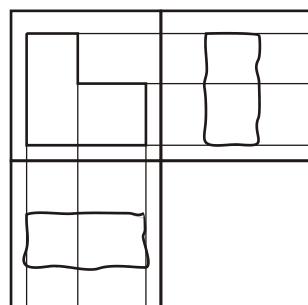


Рис. 8.7

Этап 4. Получение полного комплекта проекций.

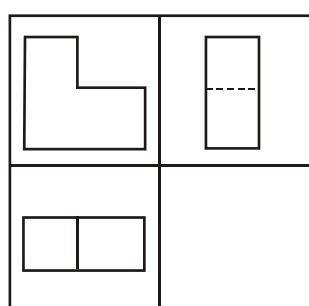


Рис. 8.8

Этап 5. Получение каркасной модели.

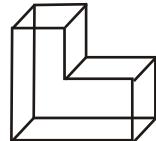


Рис. 8.9

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

Этап 6. Получение объемной модели. Из контурных элементов конструируются поверхности, которые затем объединяются в объемы.

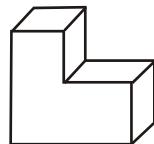


Рис. 8.10

### 8.3. Методы построения трехмерных моделей

#### Построение кривых и поверхностей

Способы построения кривой:

- интерполяция по точкам;
- деформация кривой (перемещение точки, изменение полинома);
- вычисление эквидистанты к 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 эквидистантой поверхности на заданном расстоянии.

Эти способы построения часто используются для объектов сложной формы в автомобильной и авиационной промышленности, в судостроении (формы Эрмита, Безье, В-сплайны). В машиностроении же часто 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:  $l = \langle p1, p2 \rangle$ ; и каждая точка — тремя координатами:  $p = (x, y, z)$ . Для реализации аналитических операций над гранями, для каждой грани задаются 4 коэффициента —  $A, B, C, D$ , однозначно определяющие уравнение плоскости.

Модель может быть реализована в виде графа:

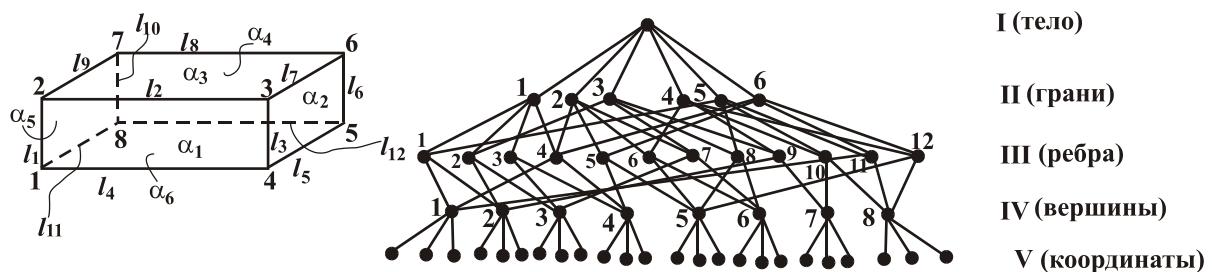


Рис. 8.11

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

Координаты вершин — 24 слова.

Коэффициенты уравнений плоскостей граней — 24 слова.

Ссылки первого уровня — 6 слов.

Ссылки второго уровня — 24 слова.

Ссылки третьего уровня — 24 слова.

Ссылки пятого уровня — 24 слова.

Итого — 126 слов.

### Кинематический принцип

Задание толщиной:

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

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

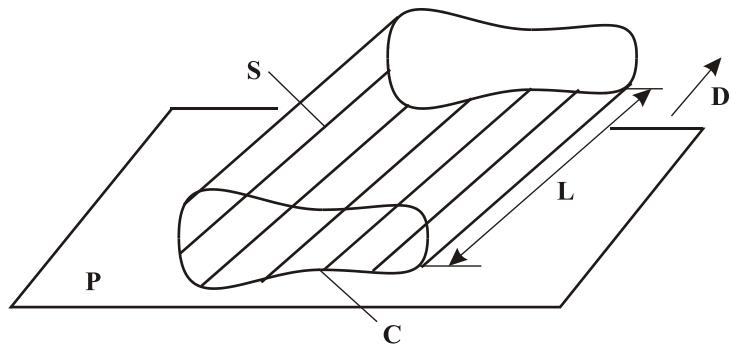


Рис. 8.12

### Задание вращением

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

Тело получается путем вращения контура  $C$  вокруг оси  $A$ .

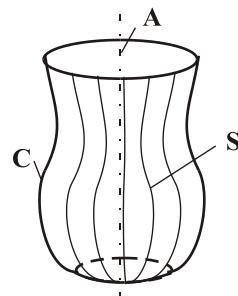


Рис. 8.13

### **Булевы операции**

Модель представляет собой дерево, узлы которого — операции, 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 существует два метода геометрического объединения:

- 1) метод контактного соединения;
- 2) метод 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:

- 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.

Метод соединения с проникновением. В данном случае используются не контурные плоские элементы, а поверхностные и соответственно рассчитываются кривые пересечения. Расчет кривых пересечения требует больших затрат, так как для каждой комбинации поверхностей надо разрабатывать свой алгоритм вычисления кривой пересечения.

Обычно существует два пути:

- 1) когда поверхности заданы аналитически (но тогда есть ограничение — поверхности должны быть максимум второго порядка);
- 2) численное определение кривой пересечения.

Этапы метода соединения с проникновением.

Этап 1. Определение объемов  $V_1$  и  $V_2$  на основе математического представления поверхностей, образующих эти объемы.

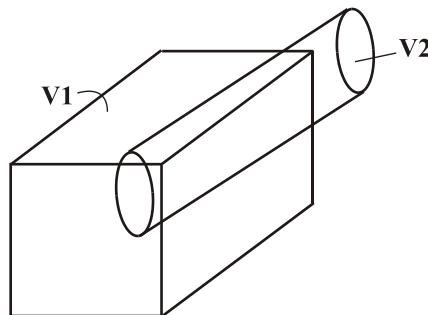


Рис. 8.14

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

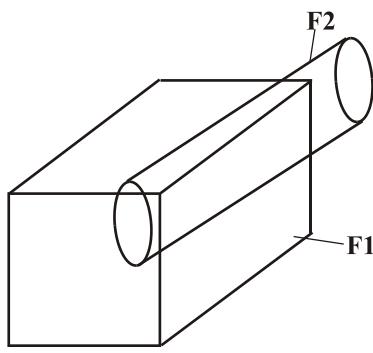


Рис. 8.15

Этап 3. Аналитическое определение кривой пересечения для пары пересекающихся поверхностей и удаление тех сегментов кривой, которые не лежат внутри пересекающихся поверхностей.

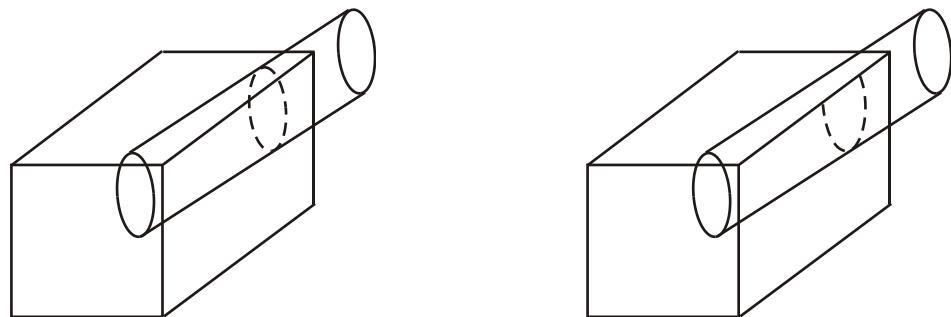


Рис 8.16

Этап 4. Сегментация поверхности в соответствии с полученной кривой  $F1 \rightarrow F1' + F1''$ .

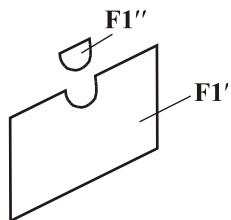


Рис. 8.17

Этап 5. Удаление сегментов поверхностей.

Рассмотрев другую пару поверхностей, получим:

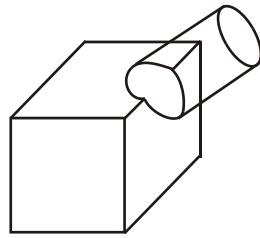


Рис. 8.18

### 5. Полигональные сетки

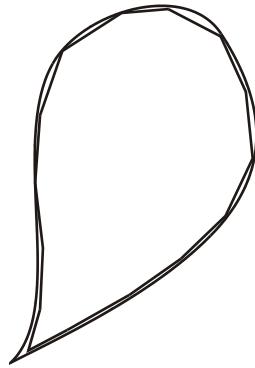


Рис. 8.19. Еречное сечение криволинейного объекта и его полигональная аппроксимация

Полигональной сеткой называют совокупность связанных между собой плоских многоугольников, с помощью которых можно аппроксимировать сложные криволинейные поверхности. Недостаток метода — его приблизительность.

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

#### **Явное задание многоугольников**

Каждый многоугольник можно задать в виде списка координат его вершин:

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

Вершины запоминаются в том порядке, в котором они встречаются при обходе вокруг многоугольника. При этом все последовательные вершины, а также первая и последняя соединяются ребрами. Для каждого отдельного многоугольника данный способ записи является эффективным, но для полигональной сетки дает большие потери памяти вследствие дублирования информации о координа-

таких общих вершин (недостаток 1). 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 (недостаток 2). Наиболее эффективный способ выполнить такое сравнение — сортировка всех  $N$  троек координат: для этого потребуется в лучшем случае —  $N \log_2 N$  сравнений. Но и при этом существует опасность, что одна и та же вершина вследствие ошибок округления может в разных многоугольниках иметь 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 (недостаток 3).

### **Задание многоугольников с помощью указателей на вершины**

Каждый узел 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. Например:

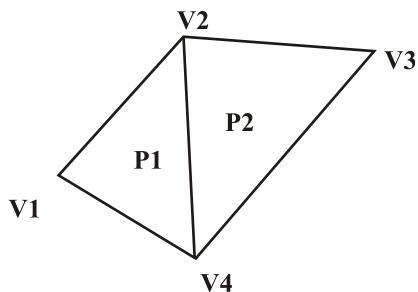


Рис. 8.20

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

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

$$P_2 = (v_4, v_2, v_3).$$

Общие ребра рисуются дважды (недостаток).

### **Явное задание ребер**

В этом представлении есть список вершин:

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

и список ребер, где каждое ребро указывает:

$$E = (V_1, V_2, P_1, P_2),$$

на две вершины в списке вершин, определяющие это ребра, а также на один или два многоугольника, которым это ребро принадлежит. Если ребро принадле-

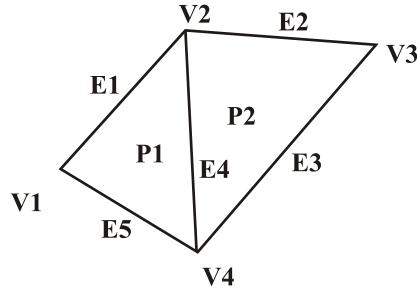


Рис. 8.21

жит одному многоугольнику, то либо  $P_1$ , либо  $P_2$  — пусто.

$$\begin{aligned}
 V = (V_1, V_2, V_3, V_4) &= ((x_1, y_1, z_1), \dots, (x_4, y_4, z_4)), \\
 E_1 &= (v_1, v_2, P_1, \lambda), \\
 E_2 &= (v_2, v_3, P_2, \lambda), \\
 E_3 &= (v_3, v_4, P_2, \lambda), \\
 E_4 &= (v_4, v_2, P_1, P_2), \\
 E_5 &= (v_4, v_1, P_1, \lambda), \\
 P_1 &= (e_1, e_4, e_5), \\
 P_2 &= (e_2, e_3, e_4).
 \end{aligned}$$

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

### Октаантные деревья

Октаантные деревья занимаются визуализацией наборов данных, изначально организованных как трехмерные массивы значений какого-либо параметра. Чаще всего такая информация приходит с различных сканирующих устройств (например, медицина) или в результате численного эксперимента (например, динамика жидких сред).

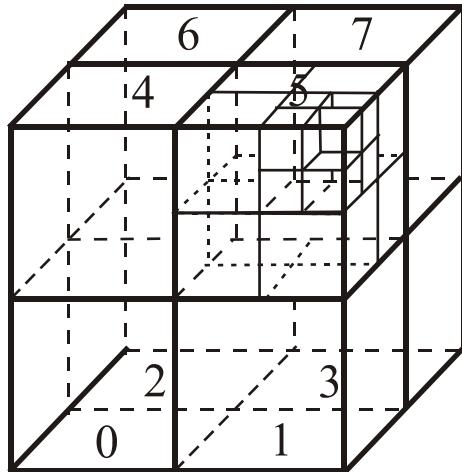


Рис. 8.22 Иерархическое октантное разбиение пространства.

Модель строится из трехмерного массива элементов объема — вокселий (voxels). Все воксели имеют форму куба и одинаковый размер по трем координатным осям. Октантное дерево организует иерархическую структуру группировки вокселий. Весь объект хранится в боксе. Этот бокс делится на восемь подбоксов плоскостями, перпендикулярными осям координат. Эти подбоксы, в свою очередь снова делятся на подбоксы и т.д. (см. рис 8.22). Этот процесс заканчивается на уровне вокселий — элементарных неделимых единиц объема. При такой схеме кодирования воксельного пространства октантное дерево может представлять кубический объем пространства из  $(2^N)^3$  вокселий, где  $N$  — глубина дерева.

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

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

Такая модель данных не позволяет эффективно представлять трехмерные плотностные модели (3D density fields) в необработанном виде. Однако после предобработки (называемой также двоичной квантизацией [9]) визуализировать такие данные (компьютерная томография и магнитно-резонансное сканирование) не представляет большой сложности.

Для того, чтобы хранить информацию только о поверхности объекта, каждый непустой подбокс имеет один байт специальной информации (инфобайт). Каждый бит этого байта говорит, содержит соответствующий ему подбокс воксели поверхности или нет. Только те подбоксы, которые содержат 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 вексель содержит информацию, а для хранения всего пространства требуются гигантские объемы памяти, октантные деревья являются существенно менее требовательными. Например, для хранения объектов, состоящих из  $256^3$  вокселей, требуется до 98 Мбайт: каждым вексель содержит 3 байта для хранения цвета поверхности + 3 байта для хранения нормали + 1/8 байта на тип вокселя ("воздух"—"кожа"), итого 6.125 байт/воксель. Прямое перемножение  $256^3$  на 6.125 дает 98 Мбайт информации. Применение октантных деревьев позволяет сократить этот объем до 1-2.5 Мбайт.

## 9. ОПИСАНИЕ И ХАРАКТЕРИСТИКА ПОВЕРХНОСТЕЙ.

### 9.1. Описание поверхностей

#### 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 запишется:

$$\begin{cases} x = a \sin \theta \cos \varphi \\ y = b \sin \theta \sin \varphi \\ z = c \cos \theta \end{cases}, \quad \theta \text{ — долгота, } \varphi \text{ — широта.}$$

Нормаль к поверхности эллипсоида:

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

Важны в геометрическом моделировании 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. Они 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 метода трассировки лучей. Это ведет к ухудшению изображений: отсутствие теней, прозрачности и зеркального отражения соседних объектов.

Достоинства параметрического описания:

- Возможность передачи геометрической формы очень сложных поверхностей, например винтообразной улитки. Она представляет собой сумму 3-х векторов: первый - вокруг которого завивается улитка, конец которого 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”;  
 $u$  изменяется в пределе  $[0, 2\pi]$ ;  
 $\theta$  изменяется в пределе  $[0, 2\pi]$ .

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 вытачивать 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  $r$ :

---


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

### 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 1-ой и 2-ой степенью. Поверхности, описываемые функциями 3-ей и 4-ой степени, незначительно расширяют возможности геометрической имитации форм, а вычисления резко увеличиваются.

Поверхности 1-го порядка:

$AX + BY + CZ + D = 0$  - описывают плоскости.

Поверхности 2-го порядка:

$AX^2 + BY^2 + CZ^2 + 2DXY + 2EYZ + 2FZX + 2GX + 2HY + 2JZ + K = 0$  - могут описывать 2 плоскости, конусы, гиперболоиды, параболоиды и эллипсоиды.

### Достоинства:

— удобна 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 числа точек, такая модель обеспечивает непрерывную форму описания.

Поточечное описание применяют в случаях, когда поверхность очень сложна, 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 3-хмерных координат точек. Они определяются автоматизированными методами на стереоприборах. Часто используется сопоставление стереопар. Надо учитывать требуемую частоту расположения точек.

### Недостатки:

- 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**

### **Поверхности 1-го порядка**

Поверхности вида:

$$f_1(X, Y, Z) = AX + BY + CZ + D = 0$$

в матричном виде имеют вид:

$$\begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \cdot P = 0, \text{ где } P = [A \ B \ C \ D]^T.$$

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 невыпуклой фигурой, могут

быть сведены к первому путем аппроксимации или разбиения на выпуклые подфигуры. Рассмотрим крыло самолета.

Границные точки многоугольника однозначно оцифровывают и записывают их координаты в матрицу:

$$\Lambda = \begin{bmatrix} X_1 & Y_1 & Z_1 \\ \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n \end{bmatrix}, \quad \text{где } n \geq 3.$$

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  $\vec{N}$ :  $N = iA + jB + kC$ .

Направлена нормаль 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).

Описывающая функция обладает непрерывностью, а производная имеет разрывы в местах стыка участков.

Достоинства:

— простота обработки.

## Поверхности 2-го порядка

$$f_2(X, Y, Z) = AX^2 + BY^2 + CZ^2 + 2DXY + 2EYZ + 2FZX + 2GX + 2HY + 2JZ + K = 0$$

Все поверхности кроме эллипсоида бесконечны. Поэтому только эллипсоид может самостоятельно образовывать объемный примитив, все другие требуют ограничения в пространстве.

Квадратичная функция в матричном виде:

$$f_2(X, Y, Z) = [X \ Y \ Z \ 1] \cdot P \cdot [X \ Y \ Z \ 1]^T, \quad \text{где } P = \begin{bmatrix} A & D & F & G \\ D & B & E & H \\ F & E & C & J \\ G & H & J & K \end{bmatrix}.$$

Нормаль  $\vec{N}$  к поверхности в точке  $(X, Y, Z)$  определяется:

$$N = \Omega \Psi, \quad \text{где} \quad \Psi = \begin{bmatrix} AX + DY + EZ + G \\ BY + DX + EZ + H \\ CZ + FX + EY + J \end{bmatrix}, \quad \Omega = [i \ j \ k]$$

$i, j, k$  – орты осей  $OX, OY, OZ$ . Направлена нормаль  $\vec{N}$  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 обратного трассирования лучей. При прямом трассировании используют параметрическую форму:

$$X = X(u, v); \quad Y = Y(u, v); \quad Z = Z(u, v).$$

Например, для эллипсоида:

$$\begin{cases} X = a \sin u \cos v \\ Y = b \sin u \sin v \\ Z = c \cos u \end{cases}.$$

Для переноса квадратичной поверхности используется преобразование:

$$[X' \ Y' \ Z' \ 1] = [X \ Y \ Z \ 1] \cdot F,$$

где  $(X', Y', Z')$  - точка в другой СК;

$F$  - матрица размером  $[4 \times 4]$ .

Новая матрица  $P'$ :

$$P' = F^{-1} P (F^{-1})^T.$$

Для перехода в другую СК параметрически заданной поверхности:

$$[X'(u, v) \ Y'(u, v) \ Z'(u, v) \ 1] = [X(u, v) \ Y(u, v) \ Z(u, v) \ 1] \cdot F.$$

Рассмотрим пример преобразования шара, описываемого формулой  $f(X, Y, Z) = X^2 + Y^2 + Z^2 - 1 = 0$  в СК  $(X', Y', Z')$  путем сдвига на 5 единиц по всем осям.

Описание шара в матричном виде:

$$[X \ Y \ Z \ 1] \cdot P \cdot [X \ Y \ Z \ 1]^T = 0.$$

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \quad F = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -5 & -5 & -5 & -1 \end{bmatrix}.$$

Координаты новой точки:

$$P' = F^{-1} P (F^{-1})^T$$

$$f(X', Y', Z') = [X \ Y \ Z \ 1] \cdot P \cdot [X \ Y \ Z \ 1]^T = (x + 5)^2 + (y + 5)^2 + (z + 5)^2 - 1 = 0$$

### Поверхности типа экструзий

(Extrusion - выдавливание). Это и металлические профили, выдавленные из расплава, и керамические пустотельные кирпичи, выдавленные из глины. К ним относятся и поверхности вращения, которые вырезаны резцом из заготовки. Широких класс машиностроительных деталей, предметов быта, архитектурных форм могут быть представлен как результат вращения кривой относительно оси. Кривую аппроксимируют ломаной линией. Описание конуса может быть неявным или параметрическим.

Другой тип поверхностей – поверхности сдвига. Кривую аппроксимируют ломаной линией, а поверхность – множеством смежных четырехугольников. Две стороны каждого четырехугольника параллельны направляющей прямой, а две остальные параллельны соответствующему отрезку ломаной (как на рисунке а и б). Направляющая линия тоже аппроксимируется ломаной (рисунок г).

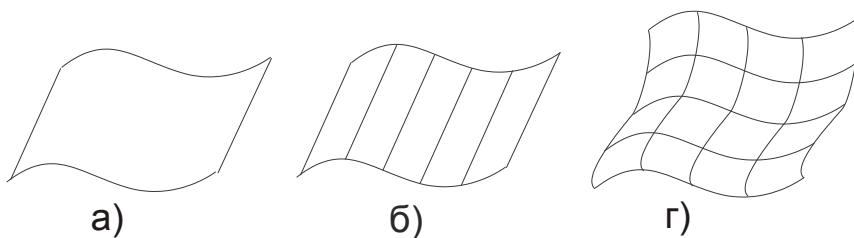


Рис. 9.1

## **Фрактальные поверхности**

Естественные сцены часто не может быть описаны на основе традиционных приемов, базирующихся на использовании непрерывных функций. Но можно заметить, что большинство природных сцен статически родственны. Проведя анализ нерегулярных изображений береговой линии, полученных при съемке с высот 10; 3 км и при наблюдении с поверхности 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 к более высоким уровням разрешения – построен метод использования фрактальных поверхностей.

В переводе (англ.) “фрактальный” обозначает состоящий из частей. Такими поверхностями называется класс нерегулярных геометрических форм, задаваемых вероятностным образом на основе исходного описания низкого разрешения. Случайный закон, по которому 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. Вначале горный массив описывают очень приближенно полигональной сеткой из четырехугольников. Каждый четырехугольник разбивают с помощью случайной функции на 4 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. Изображения, созданные на основе фрактальных поверхностей, только статистически идентичны реальным объектам и не обладают идеальной точностью.



Рис. 9.2

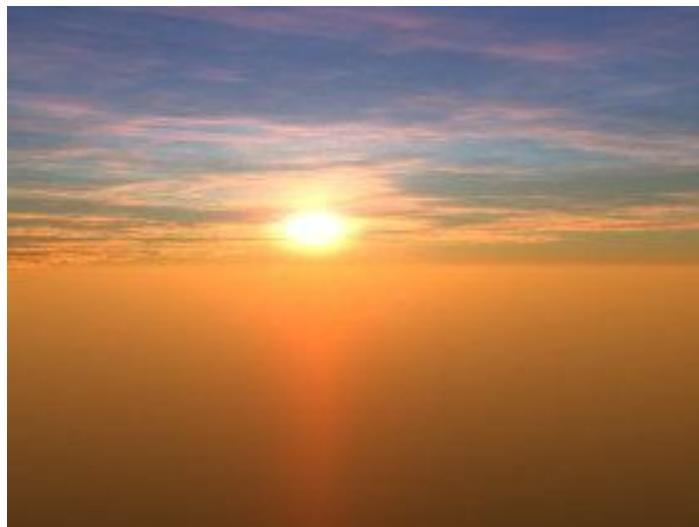


Рис. 9.3

### **9.3. Моделирование деформации трехмерных полигональных поверхностей в режиме реального времени**

Деформация (или морфинг) трехмерных объектов — это большая область компьютерной графики, в которой решаются задачи преобразования геометрических характеристик объектов. Примеры задач из этой области:

- «превращение» трехмерных объектов друг в друга;
- анимация — построение промежуточных модификаций объекта (морфопоследовательности) по двум крайним, причем каждая модификация может отличаться как пространственным положением, так и формой;
- произвольная деформация объекта с целью получения новых геометрических форм;
- моделирование неупругого взаимодействия физических тел;
- моделирование роста биологических объектов и т.д.

Конкретный алгоритм, по которому осуществляется деформация, сильно зависит от способа задания объекта.

К настоящему времени разработано большое количество методов трехмерной деформации, отличающихся по способу задания деформируемых объектов, степени физической достоверности моделируемого изменения формы (здесь имеется в виду соответствие наблюдаемой картины физическим законам) и, конечно, по производительности или по скорости работы. Как правило, деформация тел на основе физических законов требует больших вычислительных затрат, чем свободная деформация, т. е. рассчитываемая без учета физических свойств объекта и влияния других тел.

### **Метод деформации на основе использования неявного задания поверхности объекта**

Разработан метод деформации на основе использования так называемого неявного задания поверхности объекта. Основа метода — моделирование физического тела как набора связанных частиц и ассоциированной с ними неявной поверхности. На частицы могут действовать силы притяжения / отталкивания  $F_{int}$ , а также силы жидкого трения  $F_{fr}$ .

Каждая частица является источником поля  $f_i$ , напряженность которого убывает с расстоянием до нуля. Поверхность тела определяется множеством точек  $P_i$  для которых

$$f_1(P) + \dots + f_n(P) = const$$

Вместе с техникой контроля изменения объема при столкновении этот метод позволяет моделировать сложнейшие эффекты взаимодействия аморфных тел с поверхностями (протекание сквозь отверстие, растекание по поверхности) и между собой, но вычислительные затраты при его применении чрезвычайно высоки.

### **Метод деформации плоских протяженных объектов**

В случае моделирования деформации плоских протяженных объектов (одежды, поверхности человеческого лица) часто используется классический способ представления трехмерной поверхности в виде связанных треугольных граней. В этой работе рассматриваются 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 неупругая. Этот метод вместе с предложенным авторами алгоритмом отслеживания столкновения дает впечатляющие по своей реалистичности результаты при анимации одежды на движущемся человеке, флага на ветру, ряби на поверхности воды, но весьма трудоемок.

### **Деформация тела, заданного полигональной сеткой**

Был создан алгоритм деформации трехмерного тела (в общем случае полигональной поверхности), который совместил в себе высокую скорость работы и приемлемое качество моделирования реакции тела (поверхности) на внешнее воздействие. При разработке алгоритма деформации авторы исходили 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 представляет собой набор плоских граней, грань — набор вершин, соединенных ребрами, причем между любыми двумя вершинами должен существовать путь по ребрам, т. е. поверхность должна быть «связной». С каждой вершиной ассоциируется ее «масса», а с каждым ребром — некий «коэффициент жесткости». Шаги алгоритма распространения деформации:

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. Схема распространения показана на рис. 1.

Рассмотрим зависимость вектора смещения вершины от векторов смещений ее вершин-родителей. Пусть у вершины  $v$  имеется  $N$  вершин-родителей  $v_1, \dots, v_N$  со смещениями  $\vec{d}_1, \dots, \vec{d}_N$ , начальными длинами ребер, соединяющих вершину  $v$  и ее родителей,  $v_1, \dots, v_N$  равными  $l_1, \dots, l_N$ , и коэффициентами жесткости этих ребер  $k_1, \dots, k_N$ . Предлагается следующая формула для вычисления смещения вершины  $v$ :

$$dV(v_1, \dots, v_N, d_1, \dots, d_N, m, l_1, \dots, l_N, k_1, \dots, k_N) = \frac{1}{(m+1)N} \sum_{i=1}^N k_i (|v_i + d_i - v| - l_i) e_i$$

где  $m$  — масса вершины  $v$ ,  $m > 0$ ;  $\nu_1, \dots, \nu_N$  — радиус-векторы соответствующих вершин;  $e_1, \dots, e_N$  — единичные векторы в направлении разности векторов  $\nu_i + d_i$  и  $\nu$  (рис. 2),  $0 < k < 1$ .

При выводении формулы для вычисления смещения вершины в авторы рассмотрели случай двух вершин (родителя и потомка) и руководствовались следующим:

- модуль смещения вершины должен быть убывающей функцией ее массы (по аналогии с инерционностью тела в физике);
- модуль смещения вершины также должен быть возрастающей функцией жесткости соединяющего вершины ребра: условно говоря, оно может «растягиваться» и «сжиматься»;

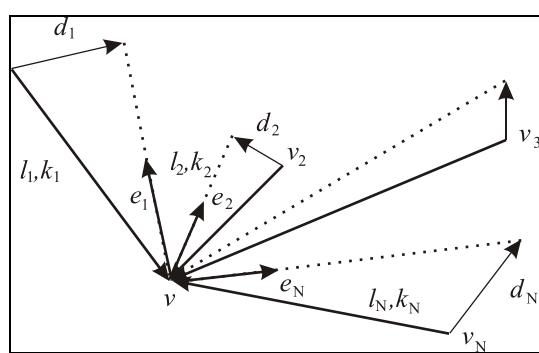


Рис. 9.4. Вычисление смещения вершины

- для каждой вершины при вычислении ее смещения необходимо учитывать только вклад вершин из списка ее родителей;
- формула должна быть достаточно простой для использования ее при вычислениях в реальном времени.

На основе этого и был выбран конкретный вид упомянутых функций. Зависимость от массы, как видно из формулы, — обратная пропорциональность (слагаемое 1 введено для избежания деления на нуль); коэффициенты жесткости входят в члены,

$$|\nu_i + d_i - \nu|$$

соответствующие вершинам-родителям, как множители.

Рассмотрим теперь остальную часть формулы. Каждое слагаемое суммы есть произведение следующих величин:  $|\nu_i + d_i - \nu| - l_i$ , — знаковое изменение длины ребра при допущении, что вершина  $v$  остается на месте;  $e_i$  — единичный вектор в направлении, в котором бы двигалась вершина  $v$  в отсутствие всех родителей, за исключением 1-го. Линия смещения есть линия, проведенная через вершину  $v$  (до смещения) и конечное положение вершины  $v$ . Конечный же вектор смещения вершины  $v$  есть результат суперпозиции (т. е. суммы) влияний всех родителей. Множитель  $k$  введен для усреднения этого влияния, так как если радиу-

сы-векторы всех родителей, их векторы смещений и жесткости ребер соответственно равны, то смещение вершины в интуитивно должно быть таким же, как если бы у  $v$  был всего один родитель с таким же смещением, радиус-вектором и жесткостью соединяющего ребра.

По классификации, приведенной во введении, данный алгоритм может быть использован для решения следующих задач:

1) произвольная деформация объекта с целью получения новых геометрических форм;

2) моделирование неупругого взаимодействия физических тел. Отсюда следует, что область возможного применения алгоритма весьма широка — 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. Рассмотрим некоторые возможные его модификации и дополнения, которые приадут процессу деформации большую зрелищность, а самому алгоритму — большую универсальность.

1. Введение порога растяжения/сжатия отдельного ребра. Под растяжением ребра здесь понимается отношение его длин до и после смещения вершины-потомка. Если это отношение больше/меньше определенного порогового значения, то это ребро можно разбить на два (или больше) и считать новую вершину родителем для старого потомка. Для реализации такого дополнения, конечно, потребуется генерация новых граней и ребер, смежных с вновь созданной вершиной. Пример подобной деформации куба показан на рис. 3.

2. «Закрепление» некоторых выбранных вершин, т. е. такие вершины не будут менять свои координаты при смещениях остальных «свободных вершин». В данном случае даже не потребуется менять сам алгоритм: этого можно достичь путем присваивания вершинам такого рода бесконечно большой массы (реально — максимального числа вещественного типа).

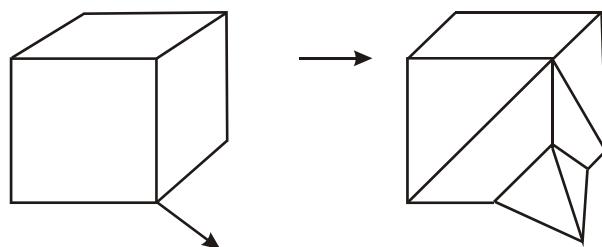


Рис. 9.5. Эффект увеличения числа граней

3. Деформирующее воздействие сразу на несколько вершин. Изменения в алгоритме в этом случае незначительны. Меняются лишь начальные условия: он стартует не с нулевого, а с первого поколения, как будто бы у необрабатываемой вершины нулевого поколения есть  $N$  потомков, где  $N$  — число первоначально смещающихся вершин.

4. «Квантование» внешнего воздействия, т. е. смещение первоначальных вершин под его влиянием, должно осуществляться в несколько проходов алго-

ритма: на каждом проходе вершины смещаются на векторы некоторой характерной длины, в сумме дающие полное смещение. Для получения «гладкости» наблюданной деформации эта длина должна быть достаточно мала по сравнению со средней длиной ребра деформируемого объекта или с какой-либо другой величиной, характеризующей его пространственный масштаб.

## 9.4. Триангуляция поверхностей

Проблема визуализации поверхности, заданной различными способами возникает во многих областях математики, физики, медицины: поверхность при этом часто заданна в виде набора точек, либо функцией.

### Ячеичные методы (cell-based)

В методах такого типа происходит разбиение области триангуляции на ячейки - параллелепипеды или треугольные пирамиды. Далее производится триангуляция поверхности в каждой ячейке отдельно.

Для применения методов этого типа необходимо задать допустимую ошибку аппроксимации, на основе которой выбрать размер ячейки - куба или тетраэдра (если быть точным - то треугольной пирамиды, т.к. тетраэдрами нельзя «замостить» пространство без пропусков и наложений.)

Наиболее известные ячеичные алгоритмы: метод Канейро, метод, предложенный Гуэзеком, метод Скалы, метод марширующих кубов.

### Алгоритм «Марширующие кубы»

Его можно разбить на два этапа:

- Разбиение области  $G$  пространства  $R^3$  на конечное множество ячеек, поиск ячеек пересекаемых искомой поверхностью.
- Аппроксимация поверхности в найденных ячейках.

Две эти подзадачи являются независимыми. Рассмотрим их подробнее.

Основные проблемы первого этапа заключается в следующем:

- Разбить область  $G$  на ячейки.
- Выбрать ячейки, которые пересекаются с искомой поверхностью.

После того как область  $G$  будет разбита на ячейки, значения функции, в общем случае, задающей поверхность будут известны только в вершинах этих ячеек. Таким образом, ячейка является главной структурной единицей во всех алгоритмах, на этом этапе.

В тех задачах, в которых функция, задающая поверхность задана таблицей на регулярной сетке, проблема разбиения области  $G$  на ячейки сразу отпадает, ввиду однозначности ее решения - ячейка должна быть параллелепипедом - для того, чтобы знать значения функции в вершинах ячейки. Если же функция задана явно, то ячейку можно выбрать произвольной формы и размера. Однако следует учесть некоторые проблемы связанные с аппроксимацией искомой поверхности в ячейке. Если размер ячейки будет очень большим, то возможна большая потеря точности.

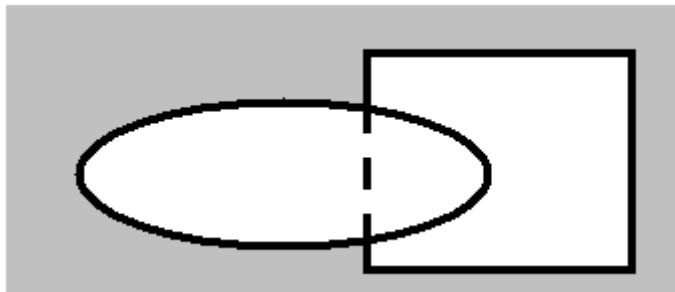


Рис. 9.6. На схеме квадрат обозначает ячейку, овал - некий изгиб искомой поверхности.

Как видно из рис.9.6, при большом размере ячейки некоторые части искомой поверхности просто не будут видны. Однако выбирать ячейки очень маленького размера не очень хорошо с точки зрения быстродействия. Поэтому размер ячейки надо выбирать не меньше допустимой погрешности построения искомой поверхности.

Работает алгоритм следующим образом:

Параллелепипед, внутри которого заведомо находится изоповерхность (или та ее часть, которую мы хотим нарисовать), разбивается сеткой, как бы разрезается на несколько меньших параллелепипедов. Затем считаются значения функции (поля) в узлах сетки, то есть в вершинах этих самых маленьких параллелепипедов. После рассматриваются все кубики. Если значения функции в вершинах кубика больше (или меньше) изоуровня - значит, кубик находится целиком над (или под) изоповерхностью, внутри этого кубика поверхности нет и он просто отбрасывается. Если же часть больше, а часть меньше, то некоторые ребра кубика пересекаются с изоповерхностью. Линейной интерполяцией приближаются эти точки пересечения и в зависимости от того, какие вершины находятся над изоповерхностью, а какие под, генерируются несколько треугольных граней. Все вершины этих граней - это как раз точки пересечения поверхности с ребрами. Как генерировать грани и пересечения каких ребер с поверхностью считать, определяется по таблице, это зависит лишь от того, какие вершины находятся над поверхностью, а какие - под. Вершин восемь, состояний два - над и под. Это дает нам 256 возможных расположений, так что таблица не такая уж и большая. Индекс в таблице тоже генерируется совсем просто: если вершина находится над поверхностью, устанавливается соответствующий этой вершине бит индекса, иначе - сбрасывается.

### Алгоритм Канейро

Алгоритм Канейро [2], основанный на разбиении пространства на треугольные пирамиды, как и алгоритм «Марширующие кубы», состоит из двух этапов:

- Разбиение пространства на конечное множество ячеек, поиск ячеек пересекаемых искомой поверхностью.
- Аппроксимация поверхности в найденных ячейках.

Как уже было сказано, алгоритм использует в качестве ячеек треугольные пирамиды. Для этого пространство разбивается на параллелепипеды в соответствии с сеткой, на которой задана функция, а затем каждый параллелепипед разбивается на треугольные пирамиды. Такой же подход применяется в алгоритмах Скалы и МТ6. Разбиение параллелепипеда на треугольные пирамиды по методу Канейро показано на рис.9.7.

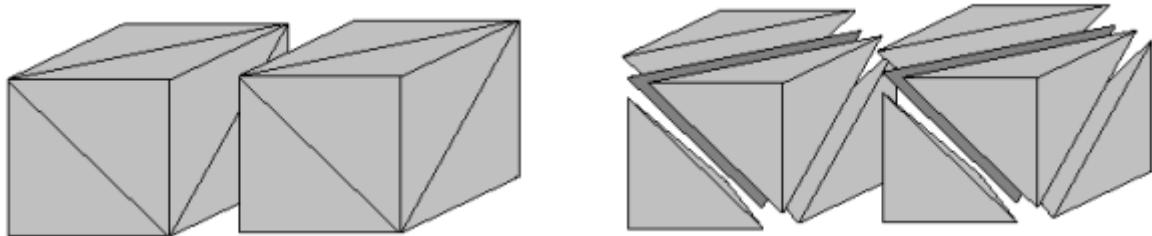


Рис. 9.7. Разбиение параллелепипеда на треугольные пирамиды по методу Канейро.

Однако при подобном разбиении «швы» «разрезов» не совпадают. Другими словами, стороны треугольников, полученных в результате триангуляции соседних ячеек, не будут совпадать, что повлечет за собой появление «дырок». Для решения этой проблемы предлагается разбивать параллелепипеды в «шахматном порядке» - по очереди меняя шаблон разбиения: с показанного на рис.9.7 на зеркальный, как показано на рис. 9.8

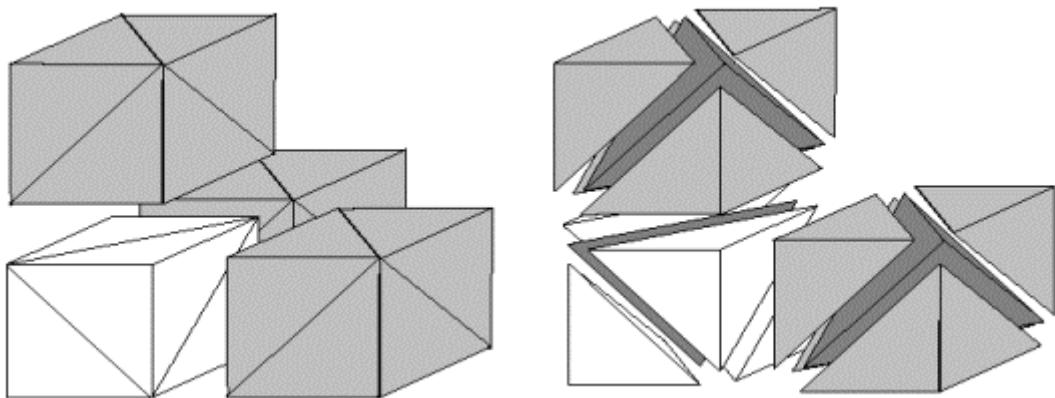


Рис. 9.8. Разбиение параллелепипеда на треугольные пирамиды со сменой шаблона.

Задача второго этапа - аппроксимация поверхности в ячейке. Для алгоритмов Канейро, Скалы и алгоритма МТ6, второй этап один и тот же - производится триангуляция треугольной пирамиды в соответствии со значениями функции в вершинах.

Посчитаем, сколько способов триангуляции имеет треугольная пирамида. Пусть имеется 4-битовый индекс. Тогда сопоставим каждой вершине один бит в индексе, таким же образом, как и для параллелепипеда. Тогда количество разных типов триангуляции будет  $2^4=16$ . Однако, используя симметрию и вращение, 16 способов можно свести к 3.

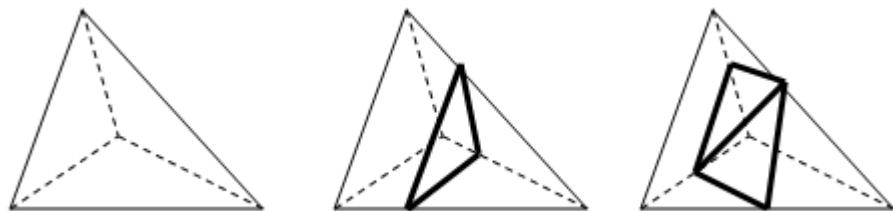


Рис. 9.9

### Алгоритм «МТ6»

Алгоритм «МТ6» был предложен Гузеком как альтернатива алгоритму Канейро. Основное отличие этих двух методов в том, что для алгоритма «МТ6» отпадает необходимость смены шаблонов с прямого на зеркальный и обратно. Это достигается путем симметричного разбиения параллелепипеда, показанного на рис.9.10.

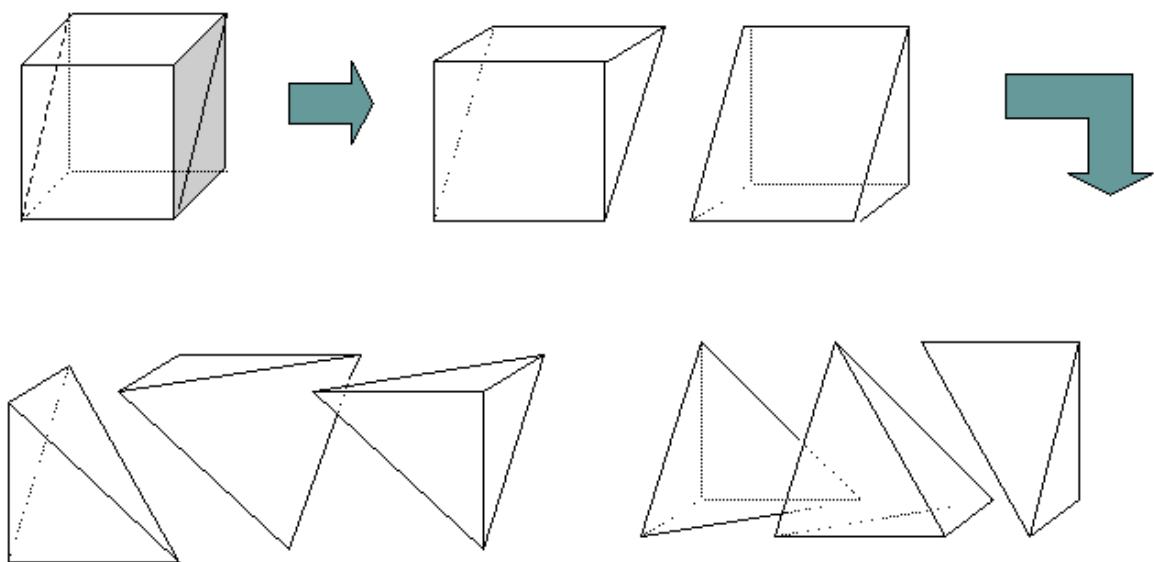


Рис. 9.10. Симметричное разбиение параллелепипеда.

### Алгоритм Скалы

Алгоритм Скалы, относящийся к разряду ячеичных методов, был разработан, для визуализации трехмерных скалярных полей, заданных с помощью функции, определенной в каждой точке пространства. Однако, метод разбиения пространства на ячейки таков, что дает возможность использовать этот алгоритм для визуализации скалярных полей заданных на регулярной сетке.

Для разбиения пространства на ячейки метод Скалы использует узлы регулярной сетки, находящиеся в вершинах параллелепипеда, полученного тем же способом, что и в предыдущих рассмотренных методах, и дополнительную точку, находящуюся на пересечении диагоналей этого параллелепипеда. Значение функции в этой точке предлагается считать как линейную интерполяцию значений функции в вершинах параллелепипеда. Для каждого параллелепипеда, полученного из узлов регулярной сетки, строится ячейка способом, показанным на рис. 10. При таком разбиении для каждой ячейки используются «срединные» точки «соседних» параллелепипедов. На рис.10 это точки I,K,D. Итог этого разбиения - 12 треугольных пирамид (DEAC, DABC, DFBC, DFEC, IEAC, IAHC, IGHC, IEGC, KAHC, KHJC, KJBC, KABC).

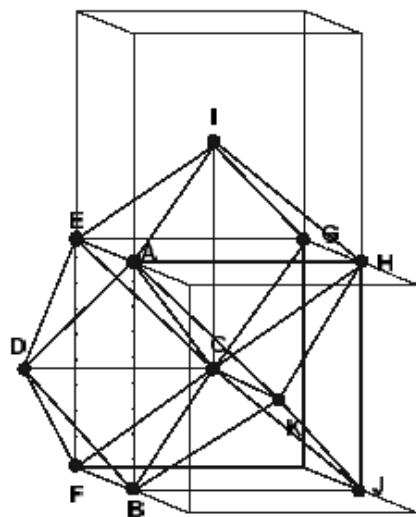


Рис. 9.11. Способ построения ячейки.

### **Методы предиктора-корректора (predictor-corrector)**

Методы из этого класса основаны на добавлении к уже имеющемуся множеству точек триангуляции ещё одной, лежащей на касательной плоскости к заданной функции (это положение предиктора - предсказанное) и затем передвижению её до визуализируемой поверхности (это положение корректора - скорректированное).

При использовании методов из этого класса, необходимо знать значение функции во всех точках пространства и найти хотя бы одну точку, принадлежащую искомой поверхности. Метод заключается в «наращивании» треугольников - на каждой итерации метода к уже существующему множеству треугольников добавляется еще один, построенный на ребре крайнего треугольника и предсказанной (а затем скорректированной по кривизне поверхности) точки на поверхности.

### **Мозаичные методы (pre-tessellation methods & particle-based methods)**

Суть таких методов заключается в разбиении искомой поверхности на части дальнейшей их триангуляции. Разбиение на части в pre-tessellation методах подра-

зумевает разбиение поверхности на «примитивные» поверхности - фрагменты сфер и плоскостей. Разбиение на части в методах из плеяды particle-based - менее «интеллектуально» - ищутся только фрагменты плоскостей. При этом возникает проблема «соединения» уже «протриангулированных» частей. Чаще всего этот процесс сводится к подбору локальных по Делоне треугольников, соединяющих части искомой поверхности.

Определение: треугольник локален по Делоне, если его самая маленькая сфера ограничения не содержит никакую другую точку триангуляции, которая имеет ту же самую поверхностную ориентацию.

### **Достоинства и недостатки методов триангуляции**

Основа методов первого типа - независимая триангуляция каждой ячейки с помощью таблиц триангуляции - является одновременно их сильной и слабой стороной. Высокая скорость работы этих методов делает их наиболее привлекательными по отношению к другим методам и дает возможность использовать их в интерактивных приложениях, но большим минусом считается невозможность правильно визуализировать локальные искривления - масштаб треугольников всегда пропорционален размеру ячейки. Такие методы идеально подходят для визуализации трехмерных скалярных полей, заданных на регулярной сетке.

Методы второго и третьего типа применимы только при визуализации полей определенных в каждой точке той части пространства, которое нас интересует. Большим плюсом таких методов можно считать их зависимость от локального искривления функции - в таких методах мелкие детали не «пропадут». Несмотря на сильную потерю в скорости по сравнению с методами первой группы и ограничениями на дифференцируемость функции и связность поверхности, привлекают высоким «качеством» получаемой поверхности.

Заметное различие между ячеичными методами и методами второго и третьего типа заключается еще и в том, что методы первого типа часто довольно просты в реализации, и предоставляют возможность визуализации «нетривиально» заданных скалярных полей. Так, к примеру, создать регулярную сетку на основе нерегулярной значительно проще, нежели восстановить функцию в каждой точке пространства. Это же относится и к проблеме восстановления поверхности по «срезам», возникающей в томографии.

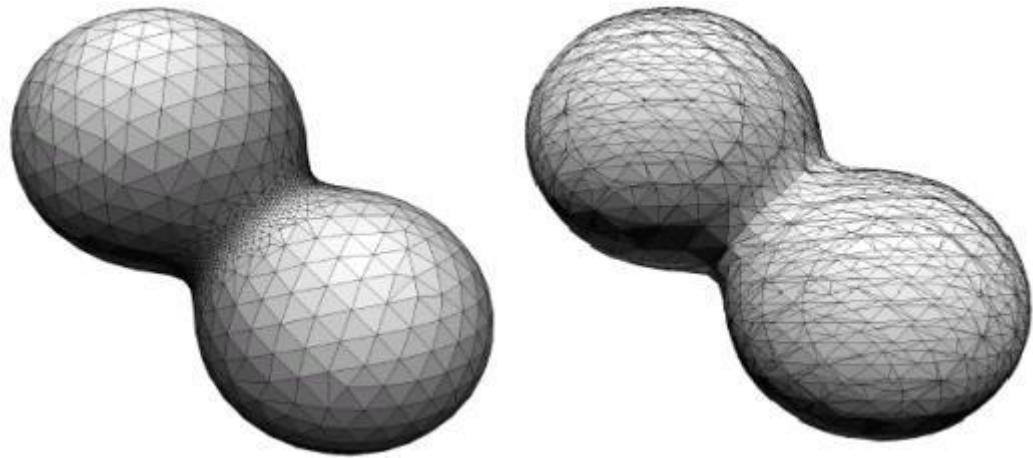


Рис. 9.12. Слева - триангуляция с учетом локальных искривлений, справа - без. (Поверхность справа получена с помощью метода «Марширующих кубов»(cell-based), слева - с помощью метода Стюарта(predictor-corrector))

Среди методов триангуляции для конечного набора точек, которые задают поверхность, широко используют метод Делоне. Метод предполагает соединение между собой набора точек непересекающимися отрезками прямых линий таким образом, чтобы сформированные треугольники стремились к равноугольности. Триангулированное изображение не может быть отнесено к триангуляции Делоне.

### Level Of Detail (LOD)

Уровень детализации (level of detail) - это метод снижения сложности рендеринга кадра, уменьшения общего количества полигонов, текстур и иных ресурсов в сцене, общее снижение её сложности. Простой пример: основная модель персонажа состоит из 10000 полигонов. В тех случаях, когда в обрабатываемой сцене он расположен близко к камере, важно, чтобы использовались все полигоны, но на очень большом расстоянии от камеры в итоговом изображении он будет занимать лишь несколько пикселей, и смысла в обработке всех 10000 полигонов нет никакого. Возможно, в этом случае будет достаточно сотни полигонов, а то и пары штук и специально подготовленной текстуры для примерно такого же отображения модели.

Метод LOD обычно используется при моделировании и рендеринге трехмерных сцен, с использованием нескольких уровней сложности для объектов, пропорционально расстоянию от них до камеры. 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.

Метод особенно эффективен, если количество объектов в сцене велико, и они расположены на разных расстояниях от камеры.

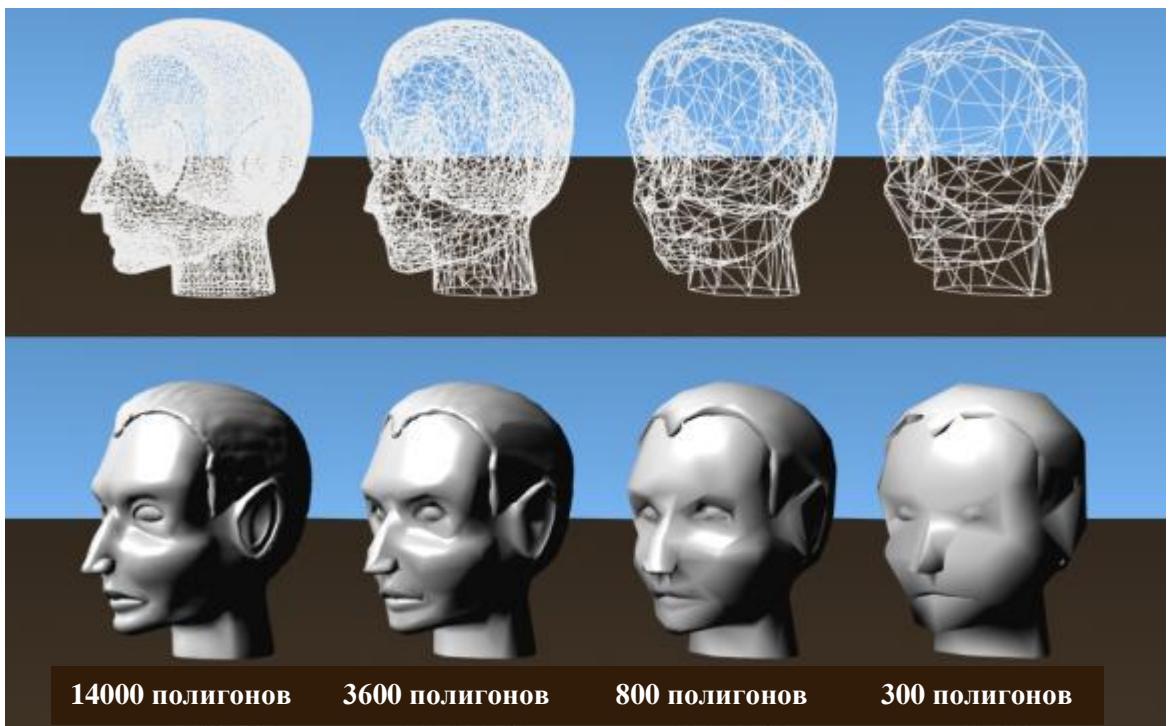


Рис. 9.13

Кроме расстояния от камеры, для LOD могут иметь значение и другие факторы - общее количество объектов на экране (когда один-два персонажа в кадре, то используются сложные модели, а когда 10-20, они переключаются на более простые) или количество кадров в секунду. Другие возможные факторы, влияющие на уровень детализации - скорость перемещения объекта (ракету в движении вы рассмотреть вряд ли успеете, а вот улитку - запросто), важность объекта. Главное - не переборщить, частые и заметные изменения уровня детализации раздражают.



Рис. 9.14

Но на данном примере все же видно, что ближайшая модель автомобиля имеет максимальную детализацию, следующие две-три машины тоже очень близки к этому уровню, а все дальние имеют видимые упрощения, вот лишь самые значительные: отсутствуют зеркала заднего вида, номерные знаки, стеклоочистители и дополнительная светотехника. А от самой дальней модели нет даже тени на дороге. Это и есть алгоритм level of detail в действии.

## 10. ПОЛУЧЕНИЕ РЕАЛИСТИЧНЫХ ИЗОБРАЖЕНИЙ

### 10.1. Методы создания реалистических изображений

Главная трудность на пути получения изображения объекта состоит в том, что все устройства вывода является двумерными. Трехмерные объекты приходится проецировать на плоскость, что приводит к существенным потерям информации, а иногда и к неопределенности изображения. Мы рассмотрим методы, которые используются для восстановления информации, которая теряется при проецировании (пример с кубом).

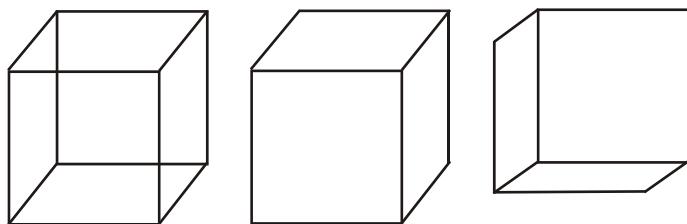


Рис. 10.1

С середины 60-х годов ведутся поиски способов и средств создания реалистичных изображений, чтобы наблюдатель принимал изображение за реальный объект, а не за синтетический объект, существующий только в памяти ЭВМ. Это особенно важно при моделировании, проектировании, организации досуга.

Конструкторам автомобилей, самолетов, машин и т.д. хотелось бы заранее знать, как будет выглядеть их проект. Формирование реалистичных изображений, генерированных на ЭВМ, во многих случаях представляет собой легкий, дешевый и более эффективный способ наблюдения предварительных результатов, чем изготовление моделей и опытных образцов, кроме того, он позволяет рассмотреть большее число вариантов проекта.

Основной трудностью создания 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, вы-

ступы на стенах). Сочетаясь в нашем сознании, они образуют «реальный» визуальный опыт.

Главная трудность при изображении пространственных отношений — практически все устройства вывода являются двумерными. Следовательно, трехмерные объекты приходится проецировать на плоскость, что приводит к существенным потерям информации, а иногда к неопределенности изображения.

Рассмотрим методы, которые используются для восстановления информации, чтобы присущие человеку механизмы восприятия глубины могли соответствующим образом ликвидировать неясности.

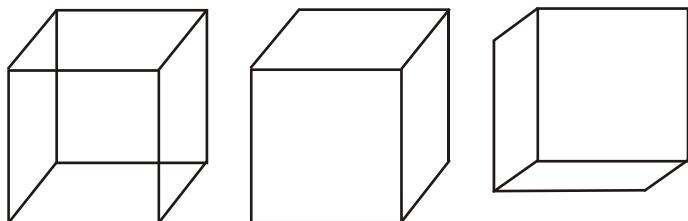


Рис. 10.2

### Перспективные проекции

Размеры объектов обратно пропорциональны их расстоянию от наблюдателя. Меньший объект расположен дальше. Но перспектива подходит не всегда. Особенно эффективно ее использование для объектов с большим числом параллельных линий, так как на изображении они будут сходиться в точке схода. Фактически это схождение линий лучше передает глубину, чем уменьшение размеров. Для сложных объектов (молекулярные структуры), где отсутствуют параллельные линии, перспективные изображения мало пригодны.

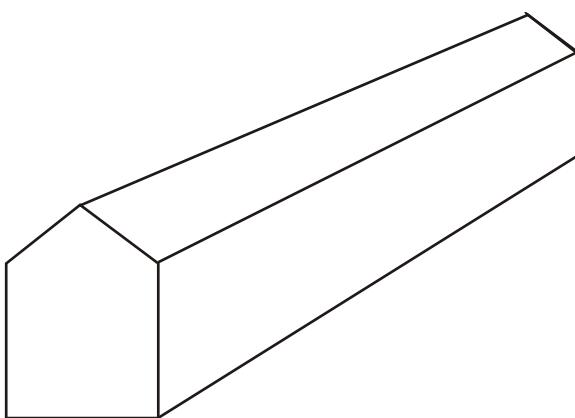


Рис. 10.3

### Передача глубины яркостью

Глубину объекта (расстояние до него) можно представить путем изменения уровня яркости: объекты, находящиеся ближе к наблюдателю, должны выводиться с увеличенной яркостью. Для реализации этого принципа необходима инфор-

мация 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.

Аналогия с реальным зрением.

### **Отсечение по глубине**

Выводимый объект пересекается плоскостью, отсекающей его удаленную часть. Удобно динамически изменять положение отсекающей плоскости.

### **Динамические проекции**

Если серию проекций объекта выводить быстро с разных точек зрения, расположенных недалеко друг от друга, то создается впечатление вращения объекта. Метод эффективен, но надо специальное аппаратное обеспечение.

### **Удаление скрытых линий и поверхностей**

Эти алгоритмы требуют больших затрат машинного времени и не удовлетворяют требованиям относительно времени ответа в машинной графике. Поэтому они используются часто как дополнительная возможность, и пользователь применяет ее очень экономно.

### **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. 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.

Используется шлем с двумя ЭЛТ. При движении головы может изменяться и изображение (датчики, фиксирующие движение).

## **10.2. Перспективные изображения**

В IV веке до нашей эры в Древней Греции существовала Сиклонская 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.

«Если зрение не может достигнуть предметов по прямым линиям, то этих предметов нельзя увидеть, так как зрение не воспринимает ничего по кривым линиям» (учебник великого немецкого художника А. Дюрера).

Долгий путь, измеряемый многими тысячелетиями, прошло развитие рисунка, прежде чем были открыты основные законы реалистического 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 нашей эры). В совершенстве овладел искусством перспективы и светотени Леонардо да Винчи. Он писал в своем «Трактате о живописи»: «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 на изображении непараллельные.

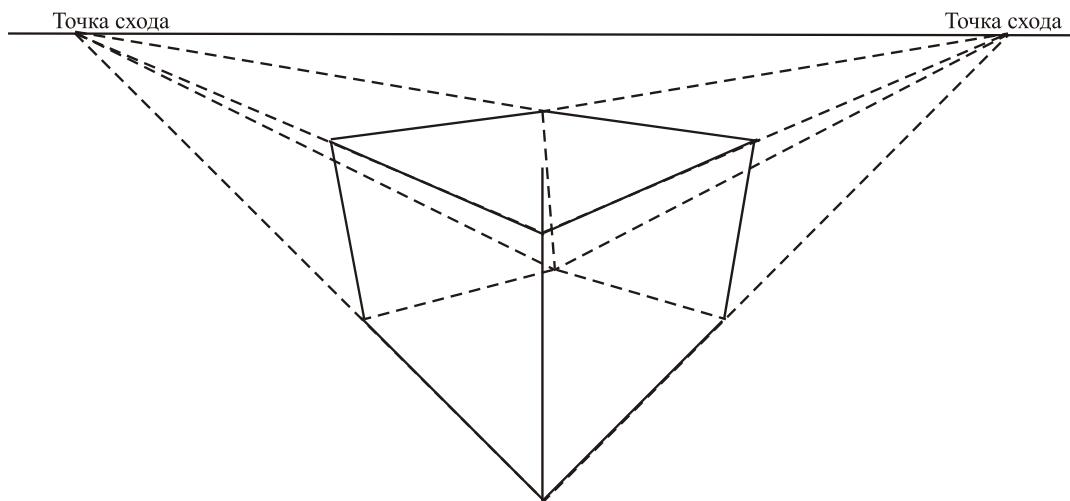


Рис. 10.4

## 11. ПРОЕЦИРОВАНИЕ

### 11.1. Основные виды проекций

В общем случае проекции преобразуют точки в СК размерностью  $n$  в точки СК размерностью  $m$ , где  $m < n$ . Мы рассмотрим преобразование 3-хмерного пространства в 2-мерное.

Проектирование 3-хмерного объекта осуществляется при помощи прямых проецирующих лучей, которые называются проекторами и которые выходят из центра проекции, проходят через каждую точку объекта и, пересекая картинную плоскость, образуют проекцию. Т.к. проекция отрезка сама является отрезком, то достаточно спроектировать лишь конечные точки.

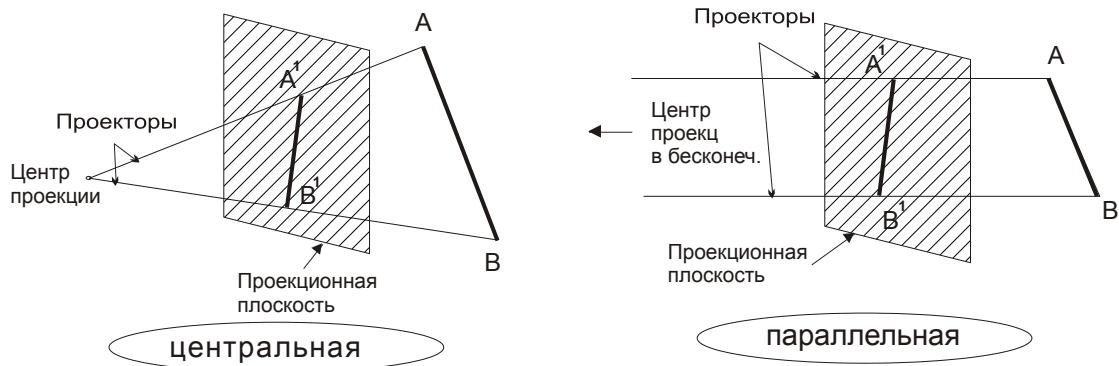


Рис. 11.1

Определенные т.о. проекции являются плоскими геометрическими проекциями (проецирование на плоскость прямыми линиями).

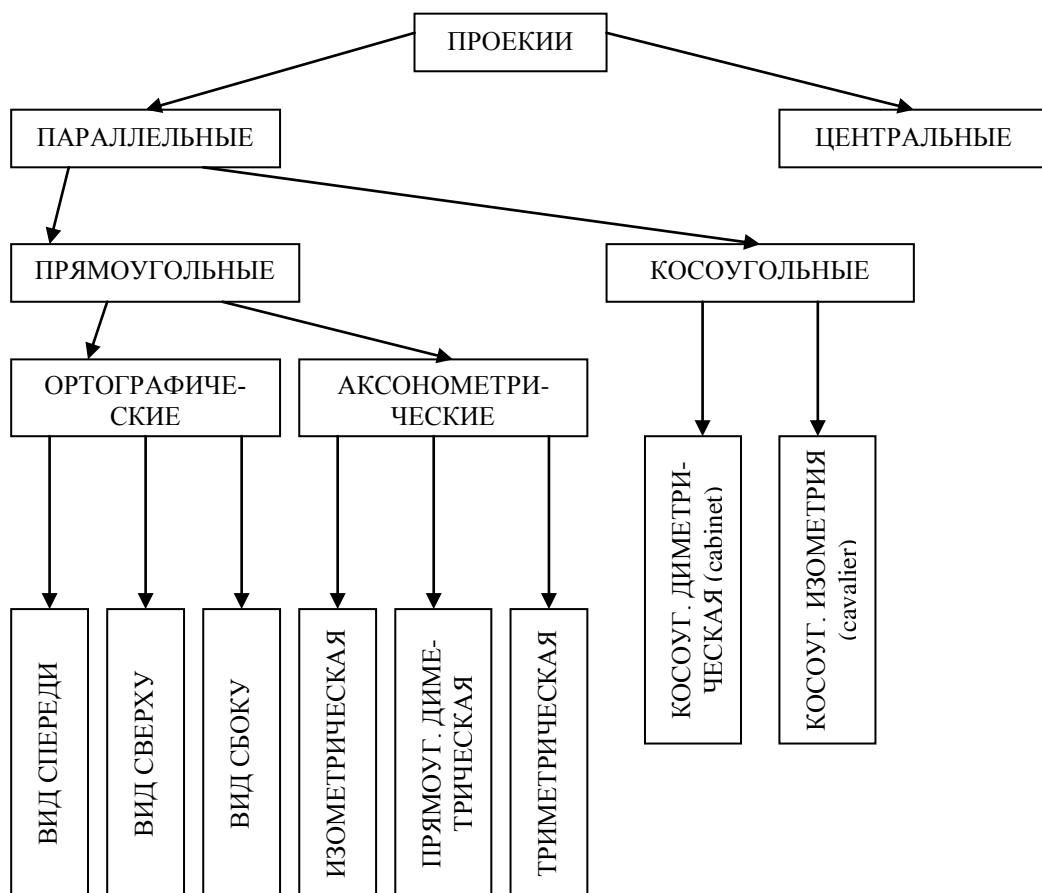


Рис. 11.2

Если расстояние между центром проекции и плоскостью проекции конечно, то проекция называется центральной, если же бесконечно, проекция – параллельная.

При описании центральной проекции (ЦПр) задается центр проекции, а при описании параллельной проекции (ППр) – направление проецирования.

## Параллельные проекции

ПП делятся на 2 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  $\vec{l}$ .

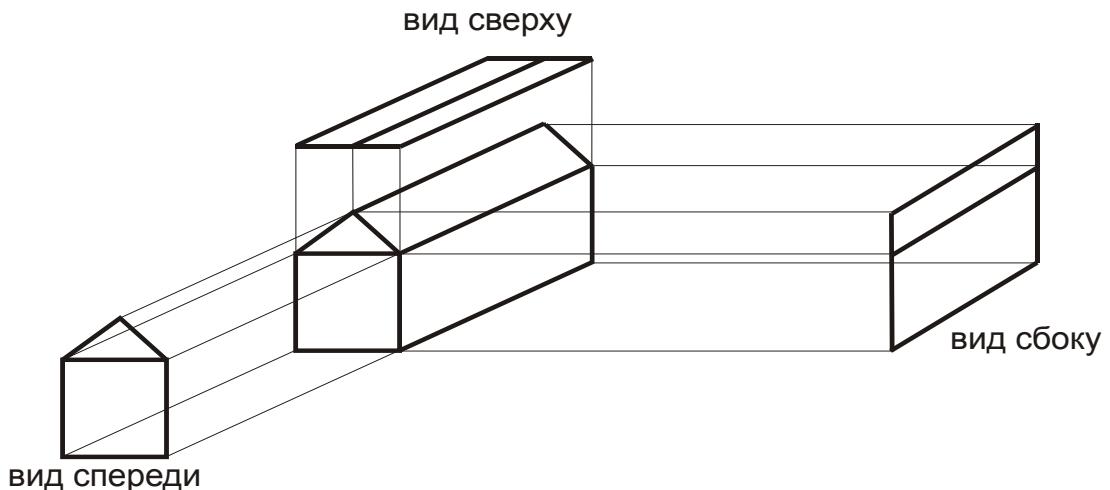


Рис. 11.3

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  $|a|=|b|=|c|$ .

Существует 4 различные изометрические проекции: (a,a,a); (-a,a,a); (a,-a,a); (a,a,-a). (Остальные 4 варианта дублируют эти, т.к. (a,a,a)=(-a,-a,-a)).

Свойство изометрии: все 3 главные координатные оси одинаково укорачиваются.

В диметрии нормаль к проекционной плоскости составляет равные углы с двумя координатными осями.

В триметрии нормаль Пр-ой 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.

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, т.е. укорачивания нет.

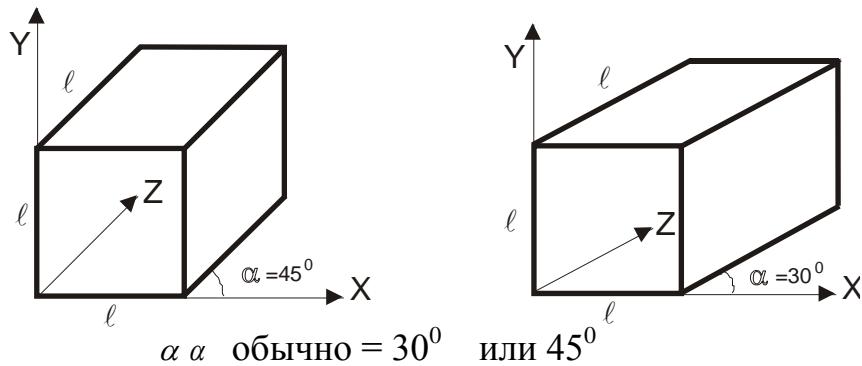


Рис. 11.4

В косоугольной диметрии направление проецирования составляет с проекционной плоскостью угол  $\arctg(2)$ . Отрезки, перпендикулярные проекционной плоскости, после проецирования составляют  $1/2$  их действительной длины.

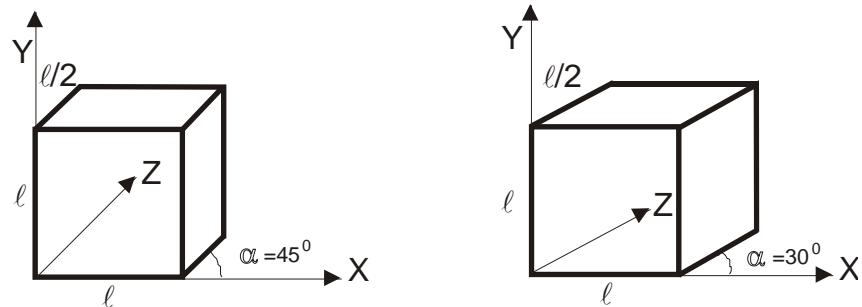


Рис. 11.5

Эта проекция более реалистична, т.к. укорачивание с  $k = 1/2$  больше согласуется с нашим визуальным опытом.

### Центральные проекции

Основное свойство — более удаленные предметы изображаются в меньших масштабах.

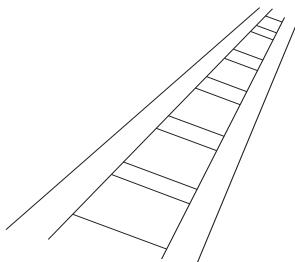


Рис. 11.6

Параллельные прямые в общем случае на изображении не параллельны.

Если совокупность прямых параллельна одной из главных координатных осей, то их т. схода называется главной т. схода. Имеется 3 такие точки.

Если проекционная плоскость перпендикулярна оси Z, то лишь на этой оси будет лежать главная т. схода.

Центральные проекции классифицируются в зависимости от числа главных точек схода, которыми они обладают.

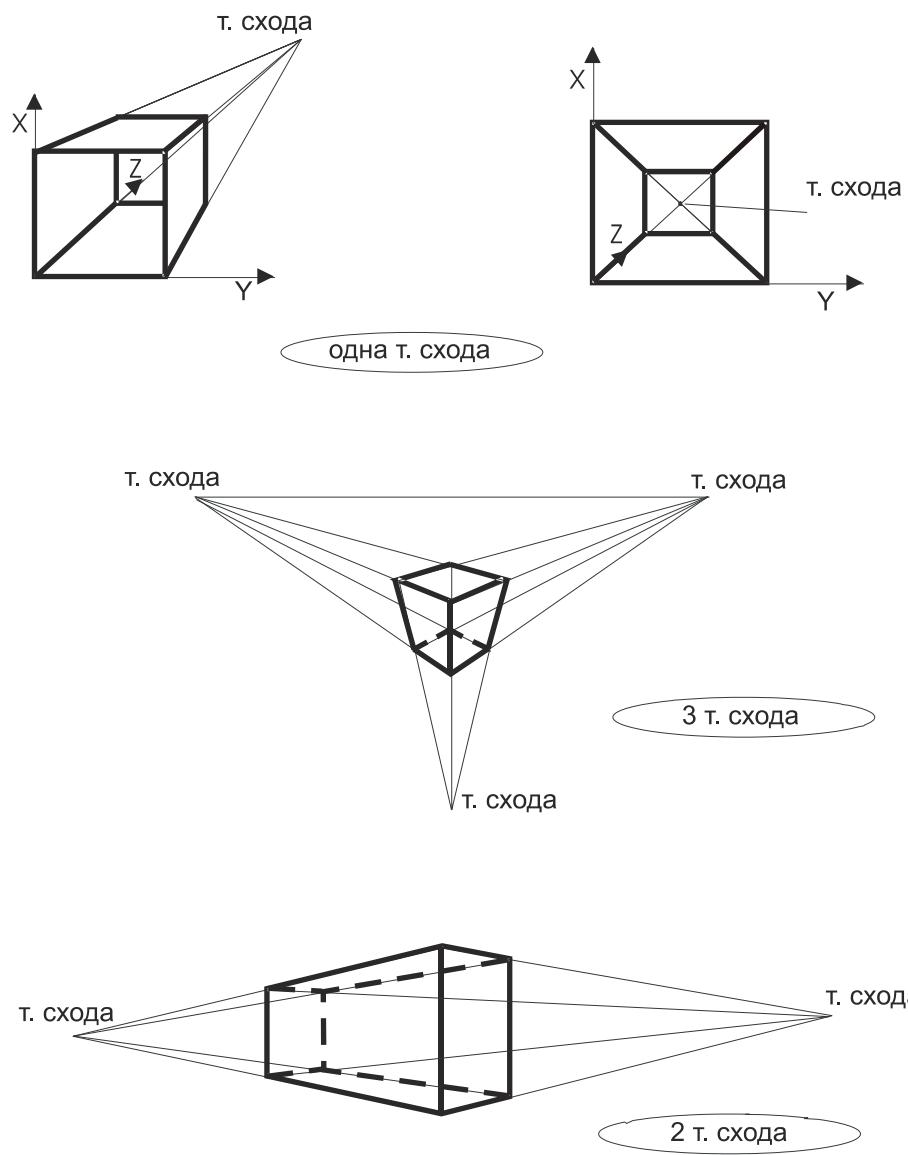


Рис. 11.7

**2-точечная** центральная проекция широко применяется в архитектурном, инженерном проектировании и в рекламных изображениях, в которых вертикальные прямые проецируются как параллельные.

**3-точечные** центральные Пр почти совсем не используются, т. к. 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 т. схода, линии явно не параллельны). Если глаз расположен далеко, то параллельные линии объекта будут казаться параллельными и на картинке.

## 11.2. Математическое описание прямоугольных проекций

**Ортографические проекции** получаются достаточно просто. Если проекционная плоскость -  $Z=0$ , направление проецирования = оси  $Z$ , то точка  $P$

$$X_p = X, Y_p = Y, Z_p = 0.$$

Матрица проецирования:

$$M_{opm_z} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ — фронтальная проекция}$$

Аналогично для 2-х других проекций:

$$M_{opm_x} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ — профильная проекция}$$

$$M_{opm_y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ — горизонтальная проекция}$$

### Аксонометрическая проекция

Получается при повороте на угол  $\psi$  относительно оси У и на угол  $\varphi$  относительно оси Х с последующим проецированием вдоль оси  $Z$ .

Матрица проецирования:

$$M_{a\alpha c} = \begin{bmatrix} \cos\psi & \sin\varphi \sin\psi & 0 & 0 \\ 0 & \cos\varphi & 0 & 0 \\ \sin\psi & -\sin\varphi \cos\psi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\psi & 0 & -\sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Покажем, как при это преобразуется единичные орты координатных осей X, Y, Z:

$$(1 \ 0 \ 0 \ 1) \cdot M = (\cos\psi \ \sin\varphi \sin\psi \ 0 \ 1)$$

$$(0 \ 1 \ 0 \ 1) \cdot M = (0 \ \cos\varphi \ 0 \ 1)$$

$$(0 \ 0 \ 1 \ 1) \cdot M = (\sin\psi \ -\sin\varphi \cos\psi \ 0 \ 1)$$

Для изомерии:

$$\begin{cases} \cos^2\psi + \sin^2\varphi \sin^2\psi = \cos^2\varphi \\ \sin^2\psi + \sin^2\varphi \cos^2\psi = \cos^2\varphi \end{cases} \Rightarrow \sin^2\varphi = \frac{1}{3}, \quad \sin^2\psi = \frac{1}{2}$$

Для диметрии:

$$\cos^2\psi + \sin^2\varphi \sin^2\psi = \cos^2\varphi \Rightarrow \sin^2\psi = \tan^2\varphi$$

Изометрическая Пр:

$$\begin{bmatrix} X_{a'} \\ Y_{a'} \\ Z_{a'} \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix} + m \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} X_a \\ Y_a \\ Z_a \end{bmatrix}$$

Диметрическая Пр:

$$\begin{bmatrix} X_{a'} \\ Y_{a'} \\ Z_{a'} \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix} + m \cdot \begin{bmatrix} 1 & 0 & -\frac{\sqrt{2}}{4} \\ 1 & 1 & \frac{\sqrt{2}}{4} \\ 0 & \frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{3} \end{bmatrix} \begin{bmatrix} X_a \\ Y_a \\ Z_a \end{bmatrix},$$

где  $X_a, Y_a, Z_a$  - координаты т. объекта,  $X_{a'}, Y_{a'}, Z_{a'}$  - координаты т. изображения.

### 11.3. Математическое описание косоугольных проекций

Матрица проецирования может быть записана исходя из значение  $\alpha$  и  $\ell$ . Рассмотрим единичный куб, спроектированный на плоскость XY. Точка  $P$  принадлежит объекту, а точка  $P'$  - изображению т.  $P$  на рисунке.

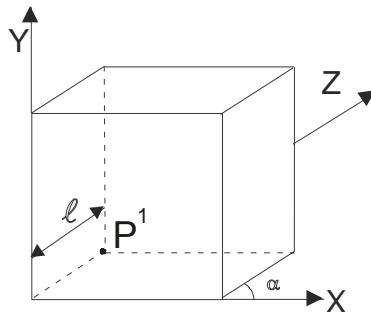


Рис. 11.8

Проекцией точки  $P(0,0,1)$  является т.  $P'(\ell \cos \alpha, \ell \sin \alpha, 0)$ , принадлежащая плоскости XY. По определению это означает, что направление проецирования совпадает с отрезком  $\overline{PP'}$ . Это направление есть:

$$P' - P = (\ell \cos \alpha, \ell \sin \alpha, -1).$$

Направление проецирования составляет угол  $\beta$  с плоскостью XY.

Рассмотрим произвольную т.  $(x, y, z)$  и определим ее косоугольную проекцию  $(x_p, y_p)$  на плоскость XY.

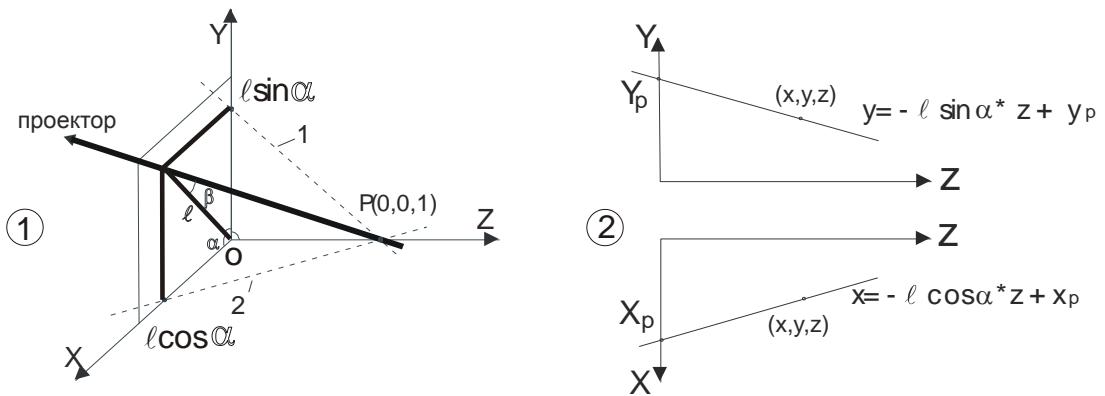


Рис. 11.9

Уравнение для  $x$  и  $y$  — координат проектора как функций  $Z$  имеют вид  
 $y = mz + b$

$$y = \underbrace{-\ell \sin \alpha}_m z + \underbrace{\ell \sin \alpha}_b \quad \text{— (прямая 1, рис.1)}$$

$$x = \underbrace{-\ell \cos \alpha}_m z + \underbrace{\ell \cos \alpha}_b \quad \text{— (прямая 2, рис.1)}$$

На рис. 11.9 показаны 2 изображения т. и проектор, параллельный проектору на рис.11.9. Уравнения для  $x$  и  $y$  — координат проектора:

$$\begin{cases} y = -\ell \sin \alpha \cdot z + y_p \\ x = -\ell \cos \alpha \cdot z + x_p \end{cases}$$

Найдем  $x_p$ ,  $y_p$ :

$$\begin{cases} x_p = x + z \cdot \ell \cos \alpha \\ y_p = y + z \cdot \ell \sin \alpha \end{cases}$$

Матрица, которая выполняет эти действия, а следовательно описывает косоугольную проекцию:

$$M_{koc} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \ell \cos \alpha & \ell \sin \alpha & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Применение матрицы  $M_{koc}$  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 проецируется на плос-

кость  $z=0$ . Сдвиг сохраняет параллельность прямых, а также углы и расстояния в плоскостях, параллельных оси  $z$ .

Для изометрической косоугольной проекции  $\ell = 1$  (см. рис.11.9)

$$\operatorname{tg} \frac{OP}{\ell} = \frac{1}{1} \Rightarrow \beta = 45^\circ$$

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  $\ell = 0$ ,  $\beta = 90^\circ$ .

#### 11.4. Математическое описание перспективной проекции

Для простоты примем, что плоскость проецирования перпендикулярна оси  $z$  и совпадает с плоскостью  $z=d$ .

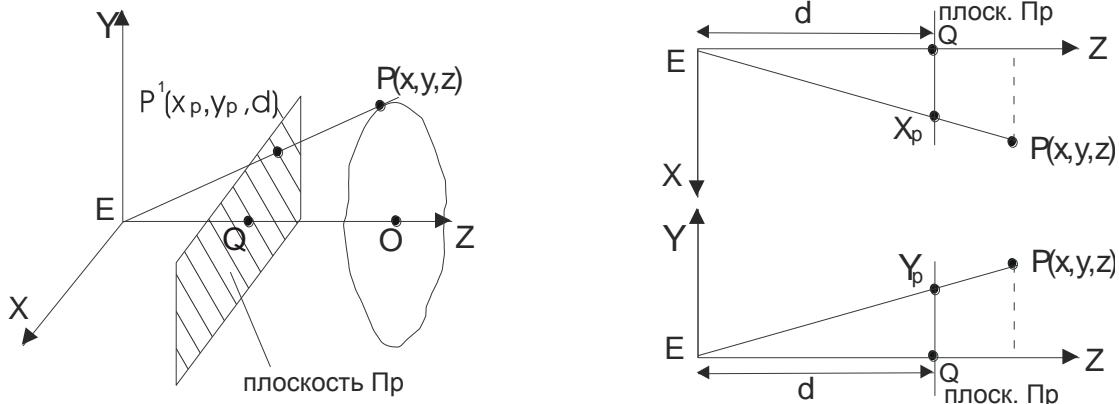


Рис. 11.10

Для каждой т.  $P$  объекта т. изображения  $P'$  определяется как т. пересечения прямой линии  $OP$  и экрана.

Рассмотрим подобные треугольники:

$$\frac{x_p}{d} = \frac{x}{z} \quad \frac{y_p}{d} = \frac{y}{z} \quad \Rightarrow \quad x_p = \frac{d \cdot x}{z} = x \cdot \frac{d}{z} \quad y_p = \frac{d \cdot y}{z} = y \cdot \frac{d}{z}$$

Расстояние  $d$  является в данном случае масштабом  $k$ . Фактором, приводящим к тому, что удаленные объекты выглядят мельче, является деление на  $z$ . Допустимы все значения  $z$ , кроме  $z=0$ .

$$M_{nepcn} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/d \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$P' = P \cdot M_{nepcn} = [x \ y \ z \ 1] \cdot M_{nepcn} = [x \ y \ z \ z/d] = \left[ \frac{x}{z/d} \quad \frac{y}{z/d} \quad d \quad 1 \right]$$

**Замечание 1:** Т.к. т.  $O$  ( $\in$  оси  $z$ )  $\approx$  находится в центре объекта, а ось  $z$  совпадает с  $[OE]$ , то т.  $Q$  тоже будет находиться приблизительно в центре изображения объекта. Если расположить начало координат экрана (плоскости проецирования) в нижнем левом углу, а размеры экрана:  $a$  – по горизонтали,  $b$  – по вертикали, то

$$\begin{aligned} x_p &= x \cdot \frac{d}{z} + \frac{a}{2} \\ y_p &= y \cdot \frac{d}{z} + \frac{b}{2}. \end{aligned}$$

**Замечание 2:** Необходимо определить расстояние между т. наблюдения  $E$  и экраном –  $d$ .

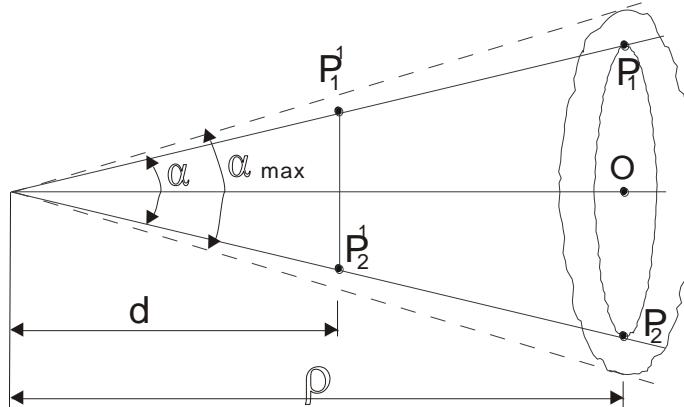


Рис. 11.11

$$\frac{\text{размер\_картинки}}{d} = \frac{\text{размер\_объекта}}{\rho}$$

$$d = \rho \frac{\text{размер\_картинки}}{\text{размер\_объекта}}$$

Это выражение равно применимо для горизонтальных и вертикальных размеров. Его надо использовать лишь для приблизительной оценки  $d$ , т.к. 3-

хмерный объект может иметь сложную форму и не всегда ясно, какие размеры надо включать в это уравнение.

**Замечание 3:** Особенность нашего глаза такова, что мы можем видеть только точки, расположенные внутри определенного конуса, ось которого совпадает с направлением взгляда ОЕ. Очень важный параметр этого конуса – угол  $\alpha_{\max}$ .

Глаз, как и камера допускает только такие значения угла  $\alpha$ , которые не  $> \alpha_{\max}$ . При выборе  $\alpha$  рекомендуется пользоваться формулой:

$$\operatorname{tg} \alpha = \frac{0,5 \cdot p - p_{\text{объекта}}}{\rho}.$$

Выбор слишком малого  $\rho$  может привести к трудностям. Если же  $\rho$  будет слишком большим, то  $\alpha$  будет мало, эффект перспективы уменьшится.

Оптимальный вариант  $\alpha = 40^{\circ} \div 60^{\circ}$ .

**Замечание 4:** Рассмотрим случай, когда объект слишком длинный в направлении оси X (балка 200\*2\*2). Где лучше выбрать т. O? До сих пор мы выбирали ее в середине объекта. Всегда ли она будет и в центре объекта?

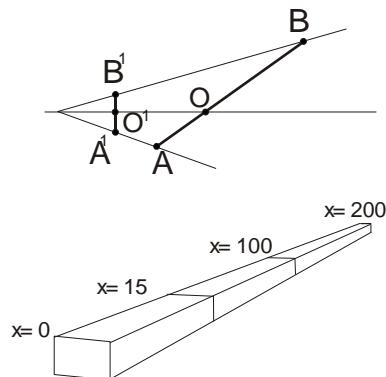


Рис. 11.12

Точка  $O'$ , которая находится в центре изображения, не лежит на середине объекта.

Очевидно, что точка  $O$  надо выбирать не на середине балки, а ближе к глазу.

### 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 относительно объекта произвольным образом. По сути дела задача сводится к преобразованию СК.

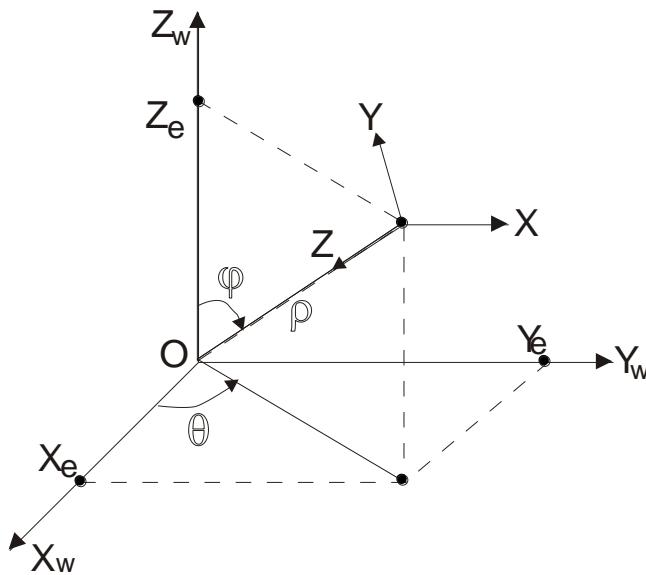


Рис. 11.13

Пусть объект задан в МСК, причем ее начало находится  $\approx$  в центре объекта (т.  $O$ ). Объект наблюдается от точки  $E$  к точке  $O$ . В точке  $E$  расположим вторую СК – видовую СК. Вектор  $EO$  определяет направление наблюдения. Пусть точка  $E$  задана в сферических координатах  $\rho, \theta, \varphi$  по отношению к мировым. Т.е. ортогональные координаты  $x_e, y_e, z_e$  могут быть вычислены:

$$\begin{cases} x_e = \rho \sin \varphi \cos \theta \\ y_e = \rho \sin \varphi \sin \theta \\ z_e = \rho \cos \varphi \end{cases}$$

Ось  $z$  ВСК расположена по линии наблюдения,  $x$  – вправо;  $y$  – вверх.  
 МСК – правосторонняя;                    ВСК – левосторонняя  
 (обычно так выбирается)                    (позволит определить картины оси в тех же направлениях)



Рис. 11.14

Видовое преобразование:

$$[x_e \quad y_e \quad z_e \quad 1] = [x_w \quad y_w \quad z_w \quad 1] \cdot V,$$

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 СК 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 СК 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 СК вокруг оси  $X$  на угол  $(\pi - \varphi)$  в положительном направлении, чтобы ось  $z$  совпадала по направлению с  $OE$ . Этот поворот соответствует повороту т. на угол  $-(\pi - \varphi) = \varphi - \pi$ .

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\varphi - \pi) & \sin(\varphi - \pi) & 0 \\ 0 & -\sin(\varphi - \pi) & \cos(\varphi - \pi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### Изменение направления оси $X$

После 3-х преобразований оси  $Y$  и  $Z$  имеют правильную ориентацию, а ось  $X$  д.б. 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.

## 12. АЛГОРИТМЫ УДАЛЕНИЯ СКРЫТЫХ ЛИНИЙ И ПОВЕРХНОСТЕЙ

### 12.1. Общие сведения об удалении скрытых линий и поверхностей

Существует большое число способов решения этой задачи. Многие из них ориентированы на специализированное применение. Наилучшего решения задачи удаления скрытых линий и поверхностей не существует. Почти все алгоритмы удаления скрытых линий и поверхностей включают в себя сортировку по геометрическому 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 пов-тей можно разделить на два типа:

1. Алгоритмы, работающие в объектном пространстве (ОП) (каждая из  $n$  граней сравнивается с оставшимися  $n-1$ );
  - они имеют дело с физической СК, в которой описаны эти объекты;
  - весьма точны (полученные изображения можно легко увеличить в несколько раз);
  - объем вычислений теоретически  $n^2$ ;
2. Алгоритмы, работающие в пространстве изображения (ПИ) (надо определить, какая из  $n$  граней видна в каждой точке разрешения экрана), т.е. каждый объект сравнивается с каждым пикселем экрана:

- имеют дело с СК экрана, на котором изображается;
- точность ограничена разрешающей способностью экрана (если полученные результаты потом увеличивать во много раз, они не будут соответствовать исходному изображению);
- объем вычислений теоретически —  $Nn$  ( $N=4000000—10000000$ ) пикселей.

## 12.2. Алгоритм сортировки по глубине

*Принцип:* все объекты сортируются по глубине и выводятся на экран в обратном порядке, и таким образом, более близкие объекты затирают более дальние (алгоритм художника).

*Шаги:*

- Упорядочение всех объектов в соответствии с их большими z-координатами.

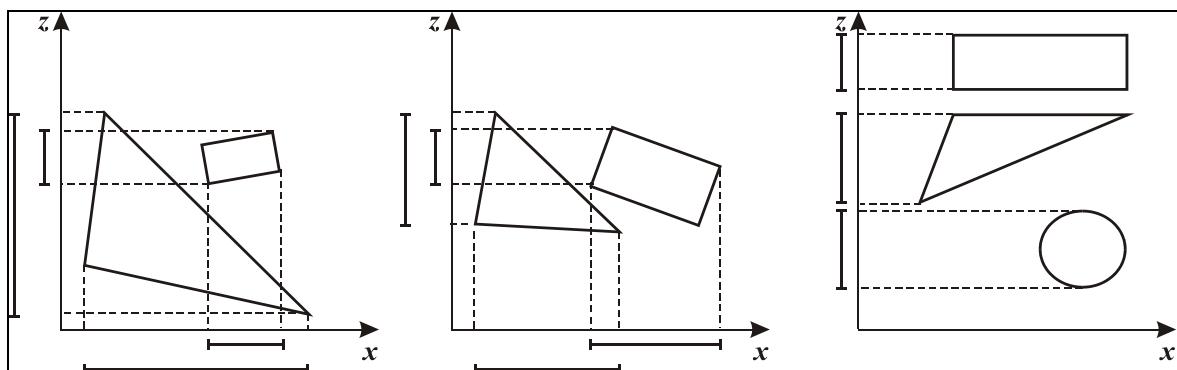


Рис. 12.1

- Разрешение всех неопределенностей, которые возникают при перекрытии z-оболочек (исследуется x-оболочка и у-оболочка).
- Преобразование каждого из объектов в растровую форму в порядке уменьшения z-координаты.

## 12.3. Алгоритм, использующий z-буфер

*Принцип:* используются два буфера: регенерации, в котором хранятся значения  $\bar{p}$ , z-буфер который хранит z-координаты.

Буфер регенерации заполняется значениями при параллельном анализе z-координаты со значениями z-буфера.

*Шаги:*

- В z-буфере заносятся максимально возможные значения z;
- Буфер регенерации заполняется значениями фона;
- Каждый объект раскладывается в растр;  
если  $z(x, y)$  меньше значения z-буфера в элементе  $(x, y)$ , то:

- a)  $z(x, y)$  заносится в элемент  $(x, y)$  z-буфера;
  - b) значение  $\bar{p}$  помещается в элемент  $(x, y)$  буфера регенерации.
- Сортировка не нужна.

### Достоинства:

- простота реализации;
- нет сортировки.

### Недостатки:

- нужен большой объем памяти для хранения z-буфера.

*Объем памяти:* информация о значении  $\bar{p}$  — 24 бита ( $8 \times 3$ ), информация о глубине 20 бит.

буфер регенерации =  $512 \times 512 \times 24$  бит

z-буфер =  $512 \times 512 \times 20$  бит

$$\} = 1,5 \text{ Мбайт}$$

- большая стоимость устранения лестничного эффекта.

### Расчет координаты $z$ :

Уравнение плоскости:

$$Ax + By + Cz + D = 0, z = \frac{-D - Ax - By}{C}.$$

Если в точке  $(x_i, y_i) \rightarrow z_i$ , то в точке  $(x_{i+1}, y_i)$ ,

где  $x_{i+1} = x_i + \Delta x \rightarrow z_{i+1} = z_i - \frac{A}{C}(\Delta x)$ ,  $\Delta x = 1 \Rightarrow z_{i+1} = z_i - \frac{A}{C}$

*Пример.*

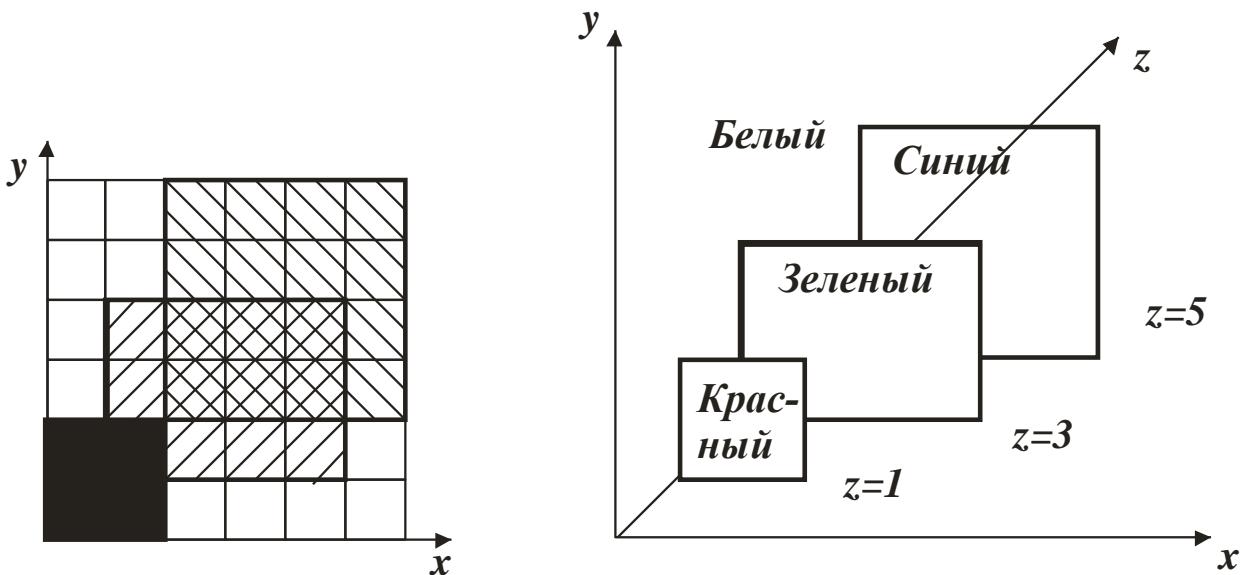


Рис. 12.2

Z—Б I	БР	Z—Б II	БР
5 5 5 5 5 5	б б б б б б	5 5 5 5 5 5	б б б б б б
5 5 5 5 5 5	б б б б б б	5 5 5 5 5 5	б б б б б б
5 5 5 5 5 5	б б б б б б	5 5 5 5 5 5	б б б б б б
5 5 5 5 5 5	б б б б б б	5 5 5 5 5 5	б б б б б б
5 5 5 5 5 5	б б б б б б	1 1 5 5 5 5	к к б б б б
5 5 5 5 5 5	б б б б б б	1 1 5 5 5 5	к к б б б б

Z—Б III	БР	Z—Б 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 б с с с с
5 3 3 3 3 5	б з з з з б	5 3 3 3 3 5	б з 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 з с
1 1 3 3 3 5	к к з з з б	1 1 3 3 3 5	к к з з з б
1 1 5 5 5 5	к к б б б б	1 1 5 5 5 5	к к б б б б

## 12.4. Алгоритм построчного сканирования

*Принцип:* расширение алгоритма преобразования многоугольника в растровую форму; разница в том, что имеем дело не с одним многоугольником, а со всеми сразу.

*Шаги:*

- Создается таблица ребер (ТР). Она содержит все ребра многоугольников, отсортированные по меньшей у-координате.

Описание ребра содержит:

$x(y_{\min})$	$y_{\max}$	$\frac{1}{m}$	Указание на многоугольник
---------------	------------	---------------	------------------------------

- Создается таблица многоугольников (ТМ).

Описание многоугольников содержит:

Коэффициенты уравнения плоско- сти ( $A, B, C, D$ )	Цвет	Флаг = $\begin{cases} 0 & \text{вне} \\ 1 & \text{внутри} \end{cases}$
-----------------------------------------------------------	------	------------------------------------------------------------------------

- Создается ТАР.

Содержит все активные ребра на текущей сканирующей строке. Ребра упорядочены по возрастанию х-координаты.

$x_i$	$y_{\max}$	$\frac{1}{m}$	Указание на многоугольник
-------	------------	---------------	---------------------------

*Пример.*

Строка  $\alpha$  — четыре ребра в ТАР.

$[LM]$ , флаг 1 = 1, флаг 2 = 0, извлекаем цвет 1.

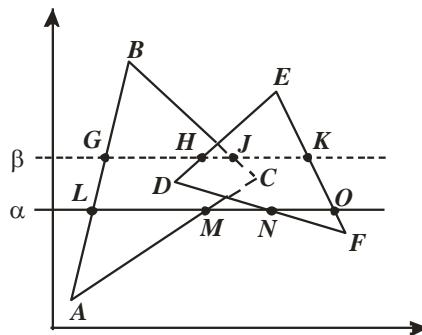


Рис. 12.3

- |                     |                          |
|---------------------|--------------------------|
| $[MN] \dashv$       | Флаг 2 = 0               |
|                     | Флаг 1 = 0, цвет = фон   |
| $[NO] \dashv$       | Флаг 2 = 1, цвет = цвет2 |
|                     | Флаг 1 = 0;              |
| $[O^\infty] \dashv$ | Флаг 2 = 0, цвет = фон   |
|                     | Флаг 1 = 0;              |

Строка  $\beta$  — четыре ребра в ТАР.

- |                     |                                                                                           |
|---------------------|-------------------------------------------------------------------------------------------|
| $[GH] \dashv$       | Флаг 1 = 1, цвет = цвет1                                                                  |
|                     | Флаг 2 = 0;                                                                               |
| $[HJ] \dashv$       | Флаг 1 = 1,<br>Флаг 2 = 1, вычисляется $z_1$ и $z_2$ из уравнений плоскости цвет = цвет 2 |
| $[JK] \dashv$       | Флаг 1 = 0,<br>Флаг 2 = 1, цвет = цвет2                                                   |
| $[K^\infty] \dashv$ | Флаг 1 = 0, цвет = фон<br>Флаг 2 = 0;                                                     |

Алгоритм немного усложняется, если многоугольники могут проникать друг в друга. Тогда находят линию пересечения и многоугольник разбивается на несколько.

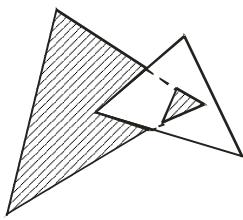


Рис. 12.4

Удобно использовать *принцип когерентности по глубине*: если при переходе к следующей сканируемой строке ребра остаются те же и в том же порядке, то соотношения глубин остаются те же и их не надо вычислять.

## 12.5. Алгоритм разбиения области

В основе лежит гипотеза о способе обработки информации глазом и мозгом. Области, более насыщенные информацией, дальше приковывают к себе взгляд.

Широко используется когерентность (однородность смежных областей).

*Принцип*: область разбивается на окна и в каждом окне решается вопрос о том, пусто ли оно или достаточно просто для визуализации; если это не так, то окно разбивается дальше до тех пор, пока не станет простым или его размер не достигнет размера  $\bar{p}$ .

При  $512 \times 512 (2^9)$  надо максимально 9 разбиений.

Конкретная реализация алгоритма зависит от метода разбиения и критерия определения простоты изображения в окне.

*Вариант 1:*

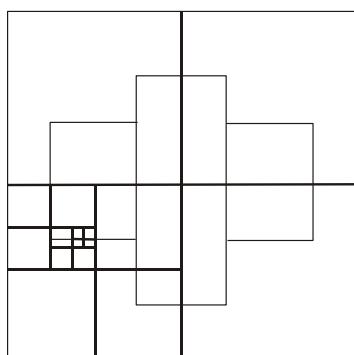


Рис. 12.5

- область разбивается последовательно на четыре равные прямоугольные части;
- критерий простоты — объекты не попадают в область.

*Вариант 2:*

- критерий простоты — одна из четырех ситуаций:

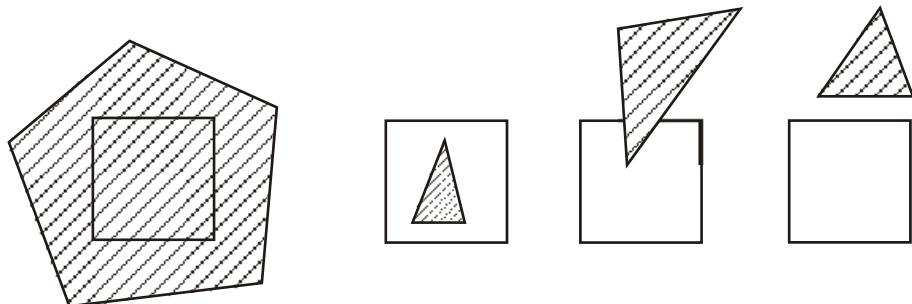


Рис. 12.6

1. Не один многоугольник не пересекает область (цвет равен цвету фона);
2. Есть один внутренний или пересекающий многоугольник:
  - цвет равен цвету фона;
  - многоугольник раскладывается в растр со своим цветом;
3. Есть один охватывающий многоугольник (цвет равен цвету многоугольника);
4. Есть несколько внутренних, пересекающих многоугольников, и как минимум один охватывающий, расположенный ближе всех (цвет равен цвету охватывающего многоугольника).

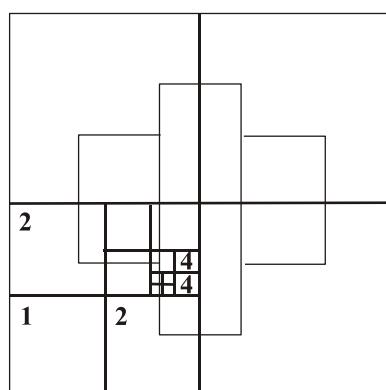


Рис. 12.7

Меньше разбиений, но больше расчетов.

*Вариант 3:* область разбивается относительно вершин многоугольника.

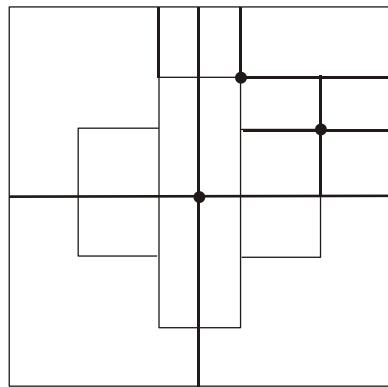


Рис. 12.8

## 12.6. Сравнительная характеристика алгоритмов

Таблица 12.1

Алгоритм	Число граней		
	100	2500	60000
Сортировка по глубине	1	10	507
z-буфер	54	54	54
Построчное сканирование	5	21	100
Разбиение области	11	64	307

## 12.7. Алгоритм плавающего горизонта

Используется чаще всего для удаления невидимых линий трехмерного представления функций, описывающих поверхности 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  $z$ .

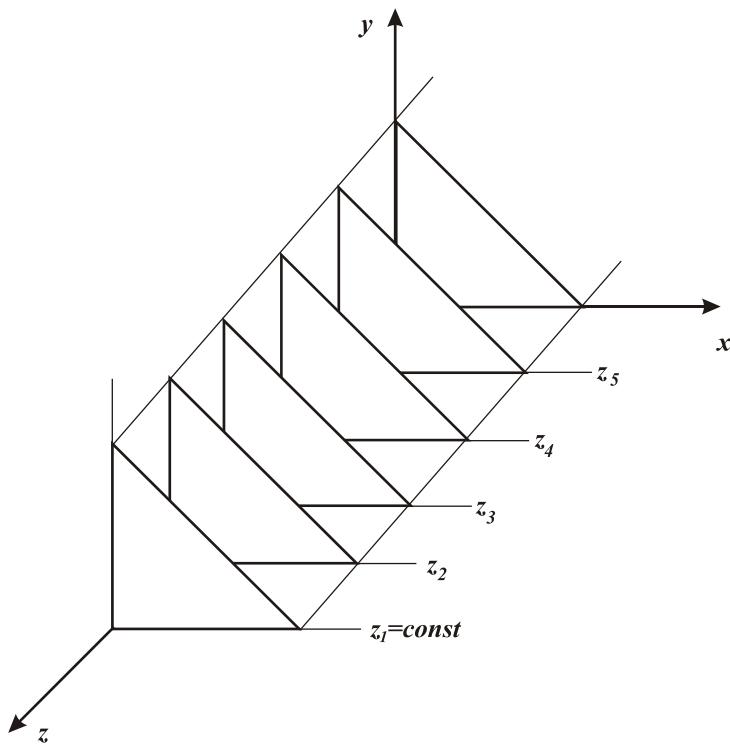


Рис. 12.9

На рисунке показаны параллельные плоскости с  $z=const$ . 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  $z = const$  на каждой из заданных параллельных плоскостей.

Поверхность теперь складывается из последовательности кривых, лежащих в каждой из этих плоскостей (см. рис.).

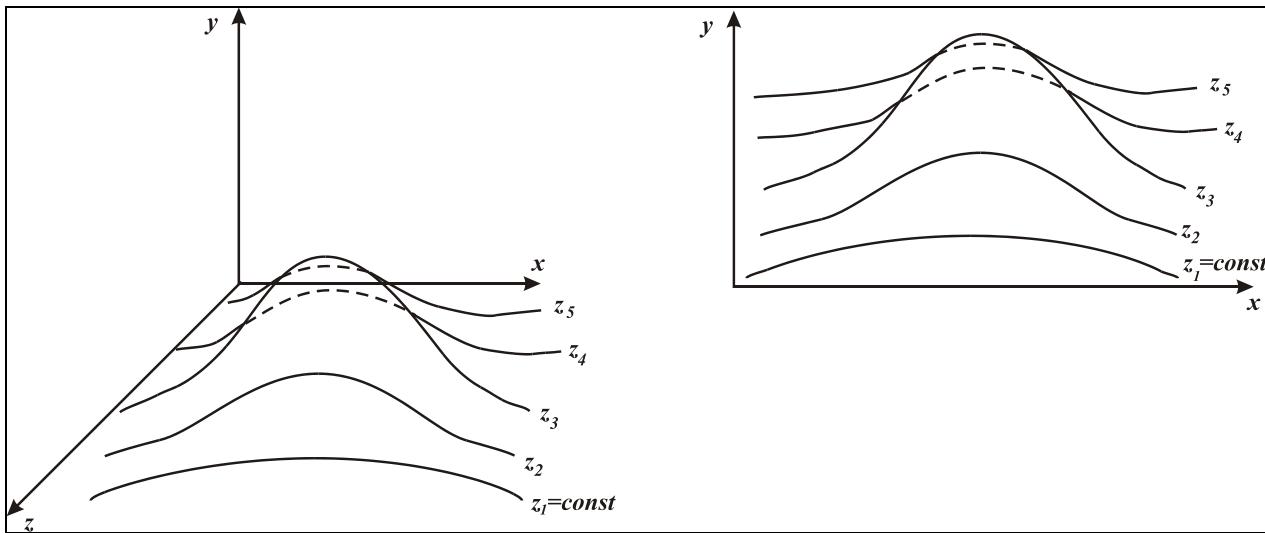


Рис. 12.10

При проецировании кривых на плоскость  $z=0$  (см. рис. 12.10) становится ясна *идея алгоритма*.

- В начале происходит упорядочивание плоскостей  $z=const$  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  $y$ .
- 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.

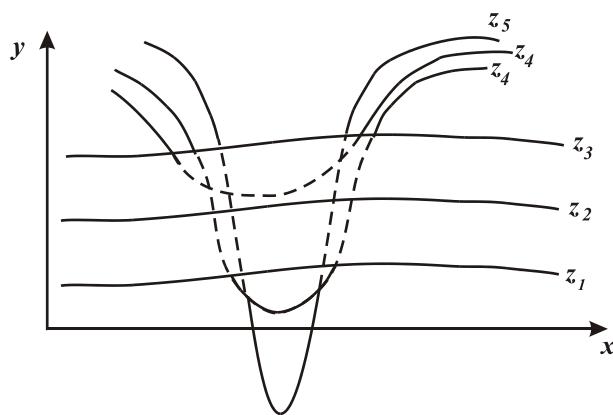


Рис. 12.11

Реализация алгоритма достаточно проста. Для хранения максимальных и минимальных значений  $y$  при каждом значении  $x$  используется 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 этот горизонт «всплывает».

Фактически этот алгоритм работает каждый раз с одной линией.

## 12.8. Алгоритм Робертса

Это математически элегантный алгоритм.

Вначале удаляются из каждого тела те ребра или грани, которые экранируются самим телом. Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части экранируются этими телами.

Трудоемкость —  $n^2$ .

Это в сочетании с распространением растровых дисплеев привело к снижению интереса к этому алгоритму. Но математические методы, используемые в нем, просты, мощны и точны. В последующем введение предварительной сортировки по z снижает трудоемкость.

Необходимо, чтобы все изображаемые тела были выпуклыми. Невыпуклые тела разбиваются на выпуклые части. Выпуклое тело с плоскими гранями представляются набором пересекающихся плоскостей. Уравнение плоскости в пространстве:

$$ax + by + cz + d = 0.$$

В матричной форме:  $[x \ y \ z \ 1] \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = 0$ , или

$[x \ y \ z \ 1] \cdot [p]^T = 0$ ,  $[p]^T = [a \ b \ c \ d]$  — представляет собой плоскость.

Поэтому любое выпуклое тело 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 точка внутри тела — знак «+», если вне — «-».

Если точка  $S$  не лежит на плоскости, то знак этого скалярного произведения показывает по какую сторону от плоскости расположена точка (точка внутри тела — знак «+», точка вне тела — знак «-»).

## 12.9. Алгоритм трассировки лучей

При рассмотрении этого алгоритма 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 определяются какие именно объекты сцены пересекаются с лучом,
- вычисляются и упорядочиваются по  $Z$  координаты точек 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  $R_x \times R_y \times N$  раз (здесь  $R_x, R_y$  — 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. Последнее обеспечивается использованием геометрически простой оболочки, объемлющей 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 есть пересечение луча с оболочкой (см. рис. 12.12)

При использовании сферической оболочки для определения пересечения луча со сферой достаточно сосчитать 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:

$$P(t) = P1 + (P2 - P1) \times t$$

Минимальное расстояние от точки центра сферы  $P0(x_0, y_0, z_0)$  до луча равно:

$$d^2 = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2$$

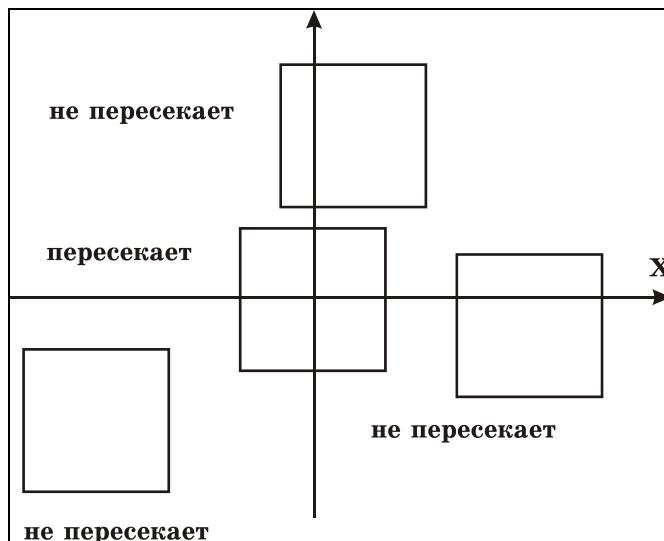


Рис. 12.12. Определение пересечения луча и оболочки

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,ложенная 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 оболочки вложенные в нее.

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.

## 12.10. Иерархический Z—буфер

Всевозрастающая сложность геометрических сцен в компьютерной графике дает прекрасную почву для исследования и разработки различных алгоритмов, по определению видимости объектов. Представим себе виртуальную прогулку по максимально детализированному геометрическому набору объектов, представляющему собой целый город 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. И пройдет еще немало времени, прежде чем будет создано то «железо», которое будет способно в одиночку справиться с данной задачей. Поэтому, чтобы выжать максимум из имеющегося в нашем распоряжении аппаратного обеспечения, мы нуждаемся в создании быстрых алгоритмов, способных значительно ускорить просчет видимых частей и отброс всей невидимой части сцены.

Существует, по крайней мере, три типа когерентных связей, присущих процессам просчета видимости объектов, умелое применение которых позволит резко увеличить скорость работы алгоритмов просчета видимой и отсечения невидимой геометрии.

1 Когерентность в объектном пространстве: во многих случаях однократное вычисление позволит определить видимость целого набора рядом расположенных объектов.

2. Когерентность в отображаемом двумерном пространстве: очень часто однократное вычисление позволяет определить видимость объекта, покрывающего определенный набор пикселей на экране.

3. Переходная когерентность — информация о видимости объектов в одном кадре, зачастую может быть использована для ускорения просчетов в следующем кадре.

Далее мы представляем алгоритм просчета видимости объектов, реализующий все три типа когерентных связей и позволяющий на несколько порядков опередить традиционные технологии.

Доминирующими технологиями для просчета видимости являются Z—buffer scan conversion и ray tracing. Т.к. алгоритмы, использующие Z—буфер, не очень эффективно работают с частично прозрачными поверхностями, поэтому мы ограничим обсуждение модельями, состоящими из непрозрачных поверхностей. В

случае применения таких моделей для определения видимости нам достаточно луча, направленного из камеры до поверхности, таким образом, мы остановимся только на алгоритмах Z—буферизации и ray casting (тот же ray tracing, только без учета вторичного луча).

Традиционный Z—буфинг достаточно эффективно использует когерентные связи в отображаемом пространстве по ходу scan conversion. При реализации алгоритма обычно сначала делаются базовые вычисления для каждого полигона, а затем производится и обновление данных по каждому пикслю полигона. Но вот проблема традиционного Z—buffer состоит в том, что этот подход 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 и сверхвысокой геометрической сложностью, как, например, модель города, данный подход очень неэффективен. Традиционный алгоритм будет, например, 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 луча (ray tracing и ray casting), наоборот, используют когерентные связи в объектном пространстве, реализуя некоторого рода пространственное деление. Луч из глаз наблюдателя (или камеры) 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, мы исключаем из дальнейшей обработки значительное количество геометрии. В этом контексте мы получили значительное преимущество по сравнению с традиционным Z—буфером, но не использовали когерентность в отображаемом пространстве и переходные взаимосвязи между кадрами. Здесь нужно заметить, что в алгоритмах основанных на трассировке лучей все же вполне возможно утилизовать переходные когерентные связи, но реализовать когерентность в отображаемом пространстве (image space coherence) представляется очень и очень сложной задачей.

В нашем случае мы представляем алгоритм, который объединяет в себе силы сразу двух (ray casting и Z—buffering) алгоритмов. Для utiлизации когерентных связей в объектном пространстве мы будем использовать рекурсивное деление пространства на 8 подпространств. Реализацию когерентности в пространстве изображения возложим на Z—buffer scan conversion, усовершенствованный при помощи Z—пирамиды, которая позволит нам очень эффективно отсекать невидимые части геометрии сцены. И наконец, для использования переходной когерентности, в качестве стартовой точки начала всего алгоритма для каждого кадра мы будем использовать уже просчитанную видимую геометрию из предыдущего кадра. Представляемый алгоритм не сложен для реализации и применим для полигонных наборов любой сложности. И чем сложнее использованная геометрия в сцене и чем выше разрешения конечного изображения, тем заметнее будет разница в скорости просчетов по сравнению с традиционными алгоритмами.

Уже были неоднократные попытки ускорить работу традиционных алгоритмов. Каждая из этих попыток представляла собой работу по использованию некоторых аспектов когерентности, присущих самой идее просчета видимости. Но ни одна из этих попыток не давала возможность совместного использования всех трех видов когерентности.

Основная масса наработок в области ray tracing расширяла возможности использования только объектной когерентности. Переходная когерентность очень редко использовалась на практике, но для ряда случаев все же есть несколько методик. Если все объекты являются выпуклыми и остаются неподвижными во время движения самой камеры, то существуют некоторые закономерности в процессе изменения видимости самих объектов. Алгоритм трассирования в состоянии отследить и использовать эти закономерности. С другой стороны, если камера является неподвижной, то тогда лучи, которые не попадают под влияние движущихся объектов, могут быть легко определены, и их можно взять из процесса обработки предыдущего кадра. Но мы по-прежнему не видим ни одного метода, способного извлечь выгоду из когерентности изображения, ввиду того, что каждый пиксель просчитывается независимо от состояния соседних. Существуют, однако, алгоритмы эвристического предсказания результатов ray tracing на конкретный пиксель по текущему состоянию соседних, однако мы и здесь не получаем гарантированного решения по использованию когерентности изображения при ray tracing.

При реализации алгоритмов Z-buffer (и алгоритмов scan conversion вообще) возникает совершенно противоположная проблема. Обычная растеризация с применением Z-буфера в подавляющем большинстве случаев связана с необходимостью предварительного просчета каждого полигона, и только после этого можно непосредственно перейти к самой растеризации посредством scan conversion, при которой последовательно обновляются все задействованные пиксели. Эта схема, сама по себе, прекрасно утилизирует когерентность изображения, поэтому необходимо задуматься только над объектной и переходной когерентностью.

Для того, чтобы быть максимально понятными, для начала введем несколько соглашений.

1) Относительно Z-буфера, полигон будет считаться невидимым, если ни один пиксель полигона не будет лежать ближе значения Z, уже записанного в соответствующую позицию Z-буфера.

2) Куб в пространстве будет считаться невидимым, если три его грани, обращенные к наблюдателю, будут невидимыми полигонами.

3) Все дочерние ветви дерева (octree), полученного при делении пространства, будут считаться невидимыми, если мы определим, что кубическое подпространство, ассоциируемое с данными ветвями, будет считаться невидимым. С учетом этих определений, мы можем сформулировать следующее условие, позволяющее нам комбинировать 2D Z-буфер с делением пространства на кубические субпространства: если куб считается невидимым на основе значений в Z-буфере, то все полигоны, которые полностью находятся в данном кубическом субпространстве, тоже будут невидимыми. Что нам это дает? А то, что если мы методом

*scan conversion* просчитаем грани определенного субпространства из состава *octree* и определим, что все пиксели этого куба находятся позади соответствующих значений в Z—буфере, то можем смело игнорировать всю геометрию находящуюся внутри данного куба.

Исходя из вышесказанного, базовый алгоритм легко сконструировать. Для начала вся геометрическое пространство последовательно размещается в структуре дерева (*octree*), ассоциируя каждый примитив с наименьшим кубическим пространством дерева, в котором примитив помещается полностью.

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 (*root*) и соответствовать началу (корню) дерева. Дальнейшая процедура является рекурсивной по сути и начинается с проверки содержащихся в данном кубе примитивов, и если их количество меньше определенного порогового значения, то процедура ассоциирует данный примитив с этим кубом и заканчивает рекурсию. Если нет, то с кубом ассоциируется любой примитив, который пересекает хотя бы одну из трех плоскостей, виртуально делящих куб на восемь равных подпространств, и производится деление куба этими тремя плоскостями. В результате этого мы получим 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), означающих собой появление новых, дочерних ветвей у родительской ветви и т.д. 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.

Закончив с формированием дерева, мы рекурсивно выполняем следующую процедуру: начиная с базового (*root*) куба, мы проверяем, попадает ли данный куб в поле зрения (*viewing frustum*), если НЕТ — на этом и закончим, если же ДА, то методом *scan conversion* для граней определяем видимость данного куба. Если куб не видим — заканчиваем, если видим — то рендеризуем геометрию, ассоциированную с данным кубом, а затем переходим к его дочерним ветвям (меньшим по размерам кубам) и последовательно выполняем вышеприведенную процедуру, но уже для этих ветвей.

Базовый алгоритм имеет несколько характеристик, на которые надо обратить особое внимание. Первое: он рендеризует 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 мы их будем называть исходя из следующих соображений: 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. Это значительное преимущество по отношению к обычному усечению геометрии под поле зрения камеры

(culling to the viewing frustum). В дополнение к этому, наш алгоритм не тратит время на ненужные ветви дерева разбиений, так как он посещает только те ветви, родительские структуры которых — видимы. Что еще более важно, наш алгоритм никогда не посещает одни и те же ветви дважды. А этот недостаток присущ алгоритмам ray tracing, где корень дерева посещается каждый раз для каждого нового пикселя, а другие ветви могут повторно посещены десятки тысяч раз. Как результат, наш базовый алгоритм осуществляет culling гораздо более эффективно.

Однако у этого алгоритма есть и слабые места. Например, алгоритм ассоциирует некоторые небольшие примитивы с большим кубом в том случае, если примитив пересекает плоскости, делящие куб на дочерние кубы. Маленький треугольник, который пересекает центр корневого (базового) куба, например, будет рендеризоваться каждый раз, пока модель является видимой. Для исключения подобного поведения есть два варианта: первый — подрезать (clip) проблематичный полигон так, чтобы он полностью уместился в значительно более меньших дочерних подпространствах (кубах). Но это неизбежно приведет к увеличению количества полигонов в структуре данных. Второй вариант — ассоциировать этот полигон сразу с несколькими ветвями. Этот вариант мы и выберем. Для его реализации нам необходимо несколько модифицировать базовый алгоритм построения дерева (octree). Если мы обнаружим, что некоторый примитив пересекается с плоскостями деления куба, но значительно меньше размеров самого куба, мы не станем ассоциировать его с этим кубом. Вместо этого мы ассоциируем его сразу с несколькими дочерними кубами, которые он пересекает. В результате, так как он ассоциирован с нескольким подпространствами сразу, при рендеризации мы встретим его несколько раз. Поэтому, как только мы встретим его первый раз, пометим его как "отрендеренный" в структуре данных и таким образом мы исключим его повторную растеризацию в текущем кадре.

Дерево делений объектного пространства (object space octree) позволяет нам отсечь значительную долю невидимой геометрии со скоростью, присущей scan conversion для граней кубических подпространств, из состава octree. Так как кубические пространства могут занимать значительную площадь в пересчете на пиксели, то такой вариант применения scan conversion может оказаться очень "дорогим" удовольствием.

Чтобы понизить стоимость процедуры определения видимости кубических пространств, мы будем использовать Z—пирамиду. В большинстве случаев для больших полигонов, Z—pyramid позволяет очень быстро определить, видим он или нет, исключая при этом попиксельные операции.

По сути, Z—пирамида очень напоминает собой пирамиду текстур с mip-levels.

Смысл Z—пирамиды — это использование базового (стандартного) 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 (тайлу). 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 мы обновляем значение базового Z—буфера, мы последовательно продвигаем это значение по более грубым уровням и остановимся в тот момент, когда встретим ту запись, значение которой находится так же далеко, как новое значение Z.

Проверка на видимость при помощи Z—пирамиды осуществляется следующим образом. Мы находим ту запись в пирамиде, которая отображает минимально возможную квадратную площадь экрана полностью содержащей исследуемый полигон. И если Z значение ближайшей вершины полигона будет дальше значения, содержащегося в этой записи, мы немедленно определим, что полигон невидим. Этим методом мы будем пользоваться для определения видимости как ветвей octree, так и для определения видимости полигонов самой модели.

Несмотря на то, что пирамида позволяет нам достаточно быстро определить видимость полигонов и отсечь невидимые, этот базовый алгоритм страдает от тех же недостатков, что и в случае с octree. Исходя из структуры пирамиды, где каждая запись соответствует определенной площади на экране (называемой квадрантом, или тайлом), маленький полигон, расположенный в центре экрана, будет сравниваться с самым грубым уровнем, находящимся в вершине пирамиды. Несмотря на то, что результат будет все равно аккуратным и в этом случае, мы теряем в производительности, так как на определение видимости такого незначительного полигона понадобится самый максимальный рекурсивный цикл.

Окончательный алгоритм определения видимости реализуется путем рекурсивного применения базового алгоритма через всю пирамиду, начиная с вершины. Если базовый алгоритм не может определить, что полигон является невидимым, мы переходим к следующему, более точному уровню пирамиды. Здесь мы попытаемся определить, что полигон является невидимым во всех четырех тайлах, которые он пересекает. Для каждого из этих квадрантов мы сравним ближайшее значение Z—полигона со значением, находящимся в Z—пирамиде. Если значение из пирамиды находится ближе, мы знаем, что полигон невидим в данном квадранте.

Если мы не можем определить, что полигон невидим хотя бы в одном из квадрантов, мы переходим на следующий, еще более точный уровень пирамиды для этого квадранта и пытаемся снова. В конце концов, мы сможем сказать, что полигон или полностью невидим, или мы дойдем до самого точного уровня пирамиды и найдем единичный видимый пиксель. Т.е. мы или отбросим полигон, или "нарисуем" один из пикселей этого полигона. Когда мы таким образом найдем все видимые пиксели, мы скажем, что произвели hierarchical scan conversion.

Потенциальная проблема с окончательным алгоритмом определения заключается в том, что может быть очень "дорого" вычислять ближайшее значение Z полигона для конкретного квадранта.

Альтернативой может быть сравнение значения Z из пирамиды с ближайшим Z целого полигона на всех стадиях рекурсии. В этом случае алгоритм гораздо шустрее, и его легче реализовать, но теперь он не является окончательным. Потому что итогом такого подхода становится либо утверждение, что полигон невидим, либо мы дойдем до уровня пикселя и не сможем сказать, видим он или нет. При возникновении такой ситуации с неопределенностью алгоритм должен переключиться на проверку данного пикселя стандартным вариантом scan conversion. На испытаниях в нашей лаборатории мы реализовали именно такой подход.

### **Переходная когерентность**

Зачастую, когда мы просчитываем проекцию сложной модели на двумерную плоскость с использованием object space octree, только малая часть кубических подпространств из состава octree остается видимой. Если мы начинаем просчитывать 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). С помощью нашего иерархического алгоритма мы реализуем и эту зависимость. Создав первый кадр, мы создадим и сохраним перечень видимых кубов из него в виде списка (temporal coherence list). Переходя к формированию следующего, перед тем, как с самого начала запустить наш иерархический алгоритм для нового кадра, мы проведем рендеризацию геометрии, содержащейся в подпространствах из нашего списка. Кубы, геометрию из которых мы уже просчитали, пометим как "rendered". На базе получившегося в результате этой операции Z-буфера создадим начальную Z-пирамиду. Теперь, запустив в работу наш алгоритм, при достаточной когерентности между кадрами мы затратим значительно меньше времени на работу алгоритма, т.к. большая часть геометрии уже будет рендеризована. Потребуется значительно меньше циклов рекурсий на выявление невидимых кубов из состава octree и полигонов геометрии. В заключение нам необходимо обновить список видимых кубов в соответствии с новым кадром.

Разумеется, что, при малой когерентности между кадрами или ее отсутствии, такой подход вынудит впустую затратить время на предварительную рендеризацию геометрии из списка, т.к. 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.

## **13. СВЕТ**

### **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 об окружающей нас действительности. Младенец берет предмет, глядит на него мгновение, затем тащит в рот. Его язык - это прекрасный датчик, и может определять форму и вид поверхности предмета практически так же, как и глаз, а иногда и лучше. Ребенок учится ассоциировать то, что он видит с той формой, которую ему описал 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 медицина вернула 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 вы можете видеть, еще не означает, что вы можете понять то, что видите.

В этом и заключается разница между Данными (Data) и Информацией (Information). Данные - это световой образ, формирующийся на сетчатке глаза. Информация — это интерпретация этого образа нашим мозгом.

Создавая изображение любого вида, вы пытаетесь сформировать световой образ на сетчатке глаза таким образом, чтобы он интерпретировался мозгом как предмет, который отображает это изображение. Тренированный мозг может извлечь огромное количество информации из изображения. Благодаря этому в голове мы можем получить полное трехмерное представление сцены, изображенной на двухмерной картинке. Чтобы получить это, 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. Когда вы, как программист, будете писать фрагменты кода, отвечающего за графику, вам не следует думать: "Я пишу процедуру затенения по Фонгу", вместо этого вам следует рассуждать так: "Я использую визуальный трюк для корректной интерпретации мозгом".

Человеческий мозг может извлечь и интерпретировать 4 информационных ресурса из потока видимых данных.

1) 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; (к месту сказать, что многие компьютерные программы для обработки изображений используют алго-

ритмы, позволяющие получать улучшения четкости, подобные тем, какие производит глаз человека.)

## 2) Оттенки

Блики и тени. Тон и структура поверхностей.

## 3) Цвет

Три цвета могут быть обнаружены человеческим глазом — красный, зеленый и синий.

## 4) Движение.

Мозг человека особенно восприимчив к движению объектов. Прекрасно "камуфлированное" животное мгновенно будет обнаружено, если оно пошевелитесь. Очень часто, если вы потеряли курсор на экране монитора, лучший способ найти его - двинуть мышкой.

Специальные отделы головного мозга отвечают за обработку этих четырех информационных ресурса. Это было неоднократно доказано в случаях анализа черепно-мозговых травм, получаемых человеком. Как только человек получает травму и лишается отдела головного мозга, отвечающего за любой из вышеперечисленных ресурсов, то он сразу утрачивает способность к восприятию этой информации. Например, в одном случае женщина потеряла способность ощущать движение. Она могла видеть так же, как все, за исключением способности чутко определять движение объектов. Например, она могла видеть автомобили на дороге, но никогда не могла сказать с первого взгляда - движутся они или нет.

Способность к восприятию принимается человеком как само собой разумеющееся. Принято считать, если вы можете видеть, то, значит, вы 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, мы имеем дело с гигантскими объемами информации. Было бы просто невозможным проанализировать и запомнить все сведения до мельчайших деталей. Да это и не нужно. Большая часть сведений (данных), поступающих нам через зрение, не обладают какой-либо ценностью. Мозг автоматически производит фильтрацию этого "мусора", позволяя нам сконцентрироваться на более значимой информации. Что еще более важно, мозг также добавляет недостающую информацию. Человеческое зрение имеет "мертвые зоны", но, тем не менее, мы этого не замечаем, потому что пробелы будут всегда 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. Ваша картина может быть значительно упрощена. Вот, например, в фильме "Возвращение Джедая" из знаменитых "Звездных Войн" один из космических кораблей в пространстве - это обыкновенный ботинок. Но никто этого не заметил, потому что ожидали видеть космический корабль, и в том месте действительно был объект, напоминающий его своей формой, поэтому все и видели именно космический корабль.

Вы можете еще более упростить свое конечное изображение, если сцена находится в движении. Нажмите паузу на видеомагнитофоне и посмотрите на не-

подвижное изображение, оно выглядит никуда негодным, но мы этого не замечаем, когда оно в движении.

Цель программиста, отвечающего за вывод графики в реальном времени, - обеспечить такие процедуры аппроксимации в визуализирующих фрагментах кода, которые улучшают реализм и точно передают атмосферу, дух создаваемого вами мира. Остальное пусть делает мозг. Цель программиста фотoreалистичной графики - попытаться смоделировать взаимодействие света с объектами сцены настолько аккуратно, чтобы оно могло выдержать скрупулезную проверку человеческим мозгом.

Так же надо учитывать две особенности глаза:

- 1) Глаз приспосабливается к "средней" яркости сцены; поэтому область с постоянной яркостью на темном фоне кажется ярче или светлее, чем на светлом.
- 2) Еще одно важное свойство: границы областей постоянной яркости кажутся более яркими.

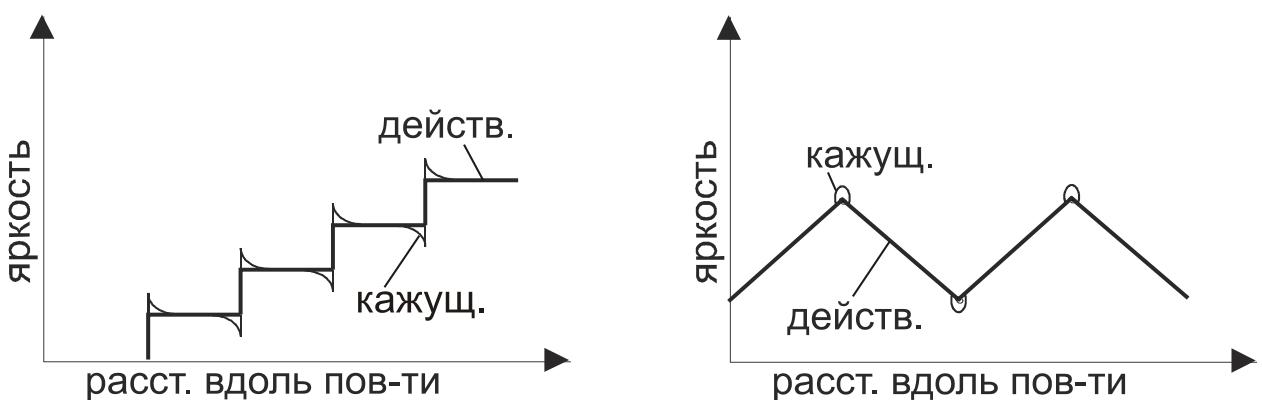


Рис. 13.1

Этот 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. показаны действительные и кажущиеся изменения яркости вдоль поверхности, вызванные литеральным 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 <затормаживаются> своими соседями с более темной стороны. И наоборот, рецепторы, расположенные на границе с более темной стороны, подвергаются меньшему воздействию, чем находящиеся дальше от границы. Причина в том, что они подвергаются более сильному торможению от соседей с яркой стороны границы.

Эффект полос Маха мешает глазу создавать сглаженное изображение сцены. Увеличивая количество полигональных граней, его можно ослабить, но полностью уничтожить нельзя.

### 13.2. Модель освещения.

Световая энергия, падающая на поверхность, может быть:

- поглощена (превращаться 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

Отражающие свойства объектов описываются коэффициентами отражения, коэффициентом яркости и индикатором отражения.

В основу классификации объектов по характеру отражения падающего света положено пространственное распределение отраженного света. Определяющее влияние на характер распределения оказывает структура поверхности объекта.

#### 4 типа поверхностей:

1. Ортотропные поверхности отражают падающий свет равномерно (диффузно) по всем направлениям. Их называют диффузными (ламбертовскими). Эти поверхности доминируют среди естественных и искусственных объектов – пески, рыхлый снег, сухой асфальт, грунт. Отличительная особенность – независимость яркости от положения наблюдателя.

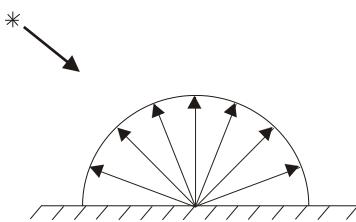


Рис. 13.2

2. Зеркальные поверхности отражают падающий свет преимущественно под углом, равным углу падения. К ним относятся чистые стеклянные поверхности,

пластики, металлические поверхности, лед, камни сухие, 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 реальных зеркальных отражений угол отражения = углу падения (идеально-отражающие поверхности - зеркало). При этом падающий свет рассеивается в некотором телесном угле относительно направления тах.

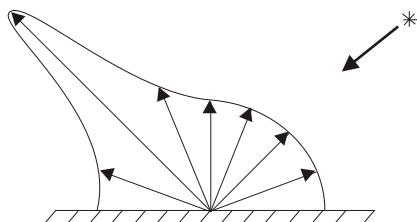


Рис. 13.3

3. Обратно отражающие поверхности отражают свет преимущественно к источнику. Их называют изрытыми, антизеркальными, световозвращающими. Такое отражение характерно для сельскохозяйственных культур, лугов и другой растительности.

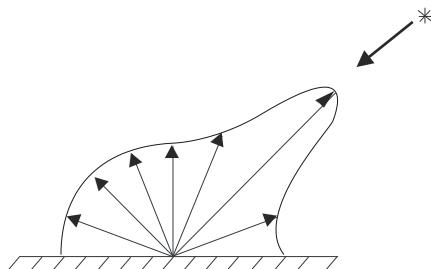


Рис. 13.4

4. Смешенное отражение. Для таких поверхностей характерно наличие 2-х или 3-х типов отражения. В общем случае можно выделить диффузную, зеркальную и обратную составляющие, а индикатриса имеет 2 тах. Такое отражение наблюдается у рисовых полей, лугов, покрытых росой и др. аналогичных объектов.

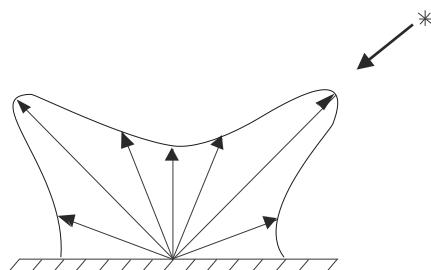


Рис. 13.5

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).

### Отражение диффузное

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, т.к. 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:

$$I_d = I_L k_d \cos \theta$$

$I_d$  - 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.

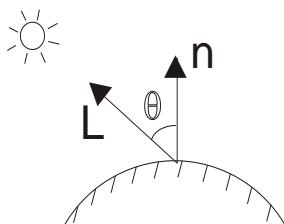


Рис. 13.6

Предметы, освещенные одним точечным источником света, выглядят контрастными (предмет в темной комнате при фотосъемке). Отсутствует рассеянный свет, как в реальной ситуации, когда на объекты падает еще и свет, отраженный от окружающей обстановки, например, от стен комнаты, других предметов.

Поэтому введем коэффициент рассеяния (const):

$$I = I_a k_a + I_L k_d \cos \theta$$

$I_a$  - интенсивность отраженного света;

$k_a$  - коэффициент диффузного отражения рассеянного света ( $0 \leq k_a \leq 1$ ).

Если есть 2 объекта, одинаково ориентированные относительно источника, но расположенные на разном расстоянии, то их интенсивность ( $I$ ) по данной

формуле будет одинакова. А ведь  $I$  д.б. обратно пропорциональна расстоянию до объекта.

Тогда модель освещения примет вид:

$$I = I_a k_a + \frac{I_L k_d \cos \theta}{d + K}$$

$d$  - расстояние до объекта от точечного источника;

$K$  - произвольная const.

Если предполагается, что точка наблюдения находится в  $\infty$ , то  $d$  определяется положением объекта, ближайшего к точке наблюдения.

Для цветных поверхностей модель освещения применяется к каждому из 3-х основных цветов.

### Зеркальное отражение

Что означает термин «идеальное зеркало»? Будем полагать, что 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 остается правило: один луч падает — один отражается. Можно рассматривать также «неидеальное зеркало». Это будет означать, 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 можно получить от любой блестящей поверхности. Осветим ярким светом яблоко — световой блик возникнет в результате зеркального отражения, а свет, отраженный от остальной части яблока — диффузный. В месте светового блика яблоко кажется не красным, а белым, т.е. окрашенным в цвет падающего света. Т.к. 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 эта 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) = углу падения; в любом другом положении наблюдатель не видит зеркально отраженный свет ( $\alpha = 0$ ).

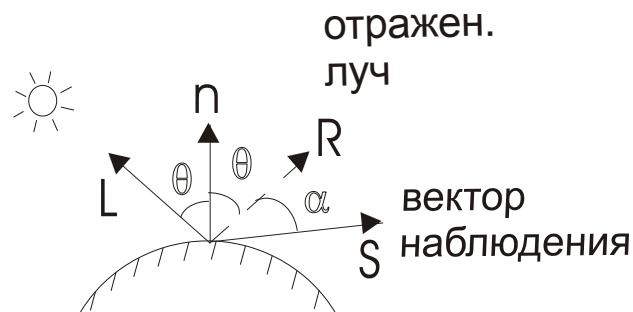


Рис. 13.7

Для неидеально отраженных поверхностей (яблоко) интенсивность отраженного света резко падает с увеличением  $\alpha$ . У гладких поверхностей распределение узкое, сфокусированное, у шероховатых – более широкое.

Эмпирическая модель Фонга:

$$I_s = I_L \cdot \omega(\theta, \lambda) \cos^n \alpha$$

$\omega(\theta, \lambda)$  - кривая отражения, представляет собой отношение зеркально отраженного света к падающему, как функцию угла падения  $\theta$  и длины волны  $\lambda$ .

Большие значения  $n$  дают сфокусированные распределения характеристик металлов и др. блестящих поверхностей, а малые – более широкие распределения для малоблестящих поверхностей.

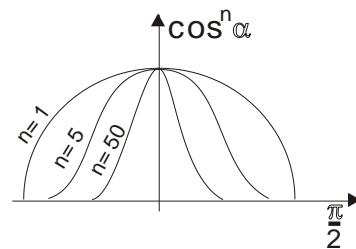


Рис. 13.8

Коэффициент отражения для металлов ( $n$ ) может быть 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.

$k_s$  обычно одинакова для всех 3-х основных цветов.

Модель освещения (функция закраски):

$$I = I_a k_a + \frac{I_L}{d + K} (k_d \cos \theta + k_s \cos^n \alpha_j)$$

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 направленно и диффузно пропускать свет. Направленное пропускание света происходит сквозь прозрачные вещества (стекло). Через них хорошо видны предметы, несмотря на то, что лучи света, как правило, преломляются, т.е. 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). Преломление рассчитывается по закону Снеллиуса: падающий и преломляющий лучи лежат в одной плоскости, а углы падения и преломления определяются:

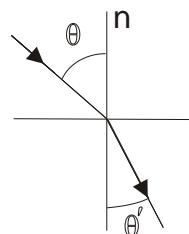


Рис. 13.9

$$\eta_1 \sin \theta = \eta_2 \sin \theta'$$

$\eta_1, \eta_2$  - показатели преломления двух сред.

Моделирование пропускания света осуществлялось несколькими способами. В простейшем из них преломление не учитывалось совсем и световые лучи пересекают поверхность без изменения направления. Т.о. 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. Без учета преломления виден предмет В, с преломлением – А. На 1-ый взгляд достаточно знать угловые соотношения в точках пересечения луча с объектом. Но это не так, т.к. длина пути луча в объекте тоже меняется,  $\Rightarrow$  1) не совпадают т. выхода луча из объекта; 2) меняется количество поглощенного объектом света, поэтому исходящий луч имеет другую интенсивность.

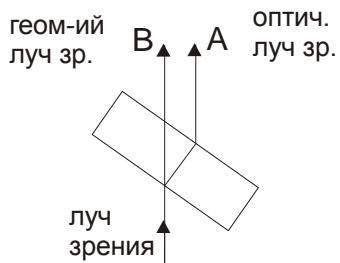


Рис. 13.10

Простое пропускание света можно встроить в любой алгоритм удаления невидимых поверхностей, кроме  $z$  – буфера, т.к. поверхности в нем обрабатываются в произвольном порядке. Если используется алгоритм построчного сканирования и передний многоугольник оказывается прозрачным, определяется ближайший из др. многоугольников, внутри которых находится сканирующая строка. Уровень закраски определяется как взвешенная сумма уровней, вычисленных для каждого из двух многоугольников:

$$I = kI_1 + (1 - k)I_2$$

$I_1$  — интенсивность видимой поверхности,

$I_2$  — интенсивность поверхности за видимой,

$k$  — коэффициент прозрачности поверхности 1 ( $k = 0 \rightarrow$  полная прозрачность

$k = 1 \rightarrow$  полная непрозрачность).

Если  $I_2$  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 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,  $d$  – диффузноотраженный свет,  $s$  - зеркальноотраженный свет,  $t$  – пропущенный свет.

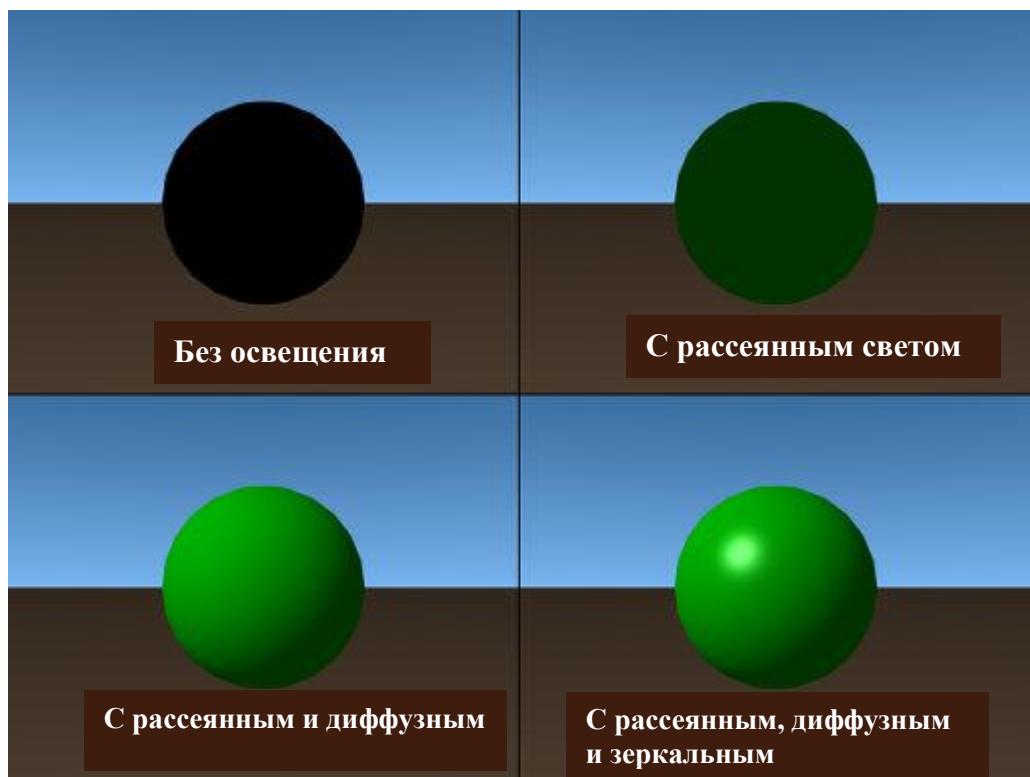


Рис. 13.11

### Специальные модели

Для исследования общих закономерностей отражения поверхностей сложной структуры используют специальные модели:

- 1) Модель Торрэнса-Спэрроу (фацентная модель).

Поверхность представляется в виде совокупности случайно ориентированных микроскопических зеркальных граней. Отражение от каждой микрограмми определяется по формуле, затем методами геометрической оптики учитывается 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) Модель слоистая используется для растительности, покрытой листвой. Каждый слой образован отдельными, в общем случае не перекрывающимися площадками определенных форм и размеров и обладающими ортотропным отражением. Отражение определяется затенением отражающих площадок нижних слоев вышестоящими. Получить аналитическое решение такой 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 машинной графике. Обычно используют приближенные модели. Полагают, что форма индикаторы отражения не зависит от длины волны.

### 13.3. Закраска полигональных сеток.

Существует 3 способа закраски объектов, заданных полигональными сетками:

#### Однотонная закраска

Вычисляется 1 уровень интенсивности, который используется для закраски всего многоугольника. При этом предполагается, что:

- 1) Источник света расположен в бесконечности ( $\cos\theta = \text{const}$  на всей полигональной грани)
- 2) Наблюдатель находится в бесконечности ( $\cos\alpha = \text{const}$  на всей полигональной грани)
- 3) Многоугольник п.с. реальную моделируемую поверхность, а не является аппроксимацией криволинейной поверхности.

Если первое или второе условие неприемлемо, можно использовать усредненное значение  $\cos\theta$ ,  $\cos\alpha$  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, т.к. интенсивность каждой из этих граней отличается от интенсивности соседних граней (эффект полос Маха).

#### Интерполяция интенсивностей (метод Гуро)

Метод Гуро позволяет получать сглаженный объект на этапе визуализации, не внося изменения в геометрическую модель (полигональные сетки). Полосы Маха значительно уменьшаются.

Процесс закраски осуществляется в 4 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.

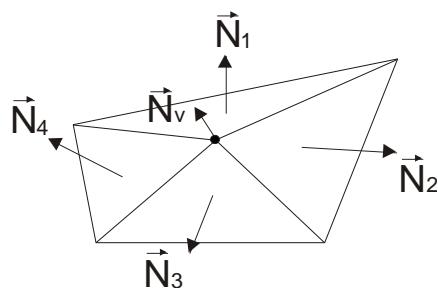


Рис. 13.12

$$\bar{N}_v = (\bar{N}_1 + \bar{N}_2 + \bar{N}_3 + \bar{N}_4) / 4$$

- 3) Используя нормали в вершинах и применяя произвольный метод закраски, вычисляются значения  $i$  в вершинах.

4) Каждый многоугольник закрашивается путем линейной интерполяции значений  $i$  в вершинах сначала вдоль каждого ребра, а затем между ребрами вдоль каждой сканирующей строки:

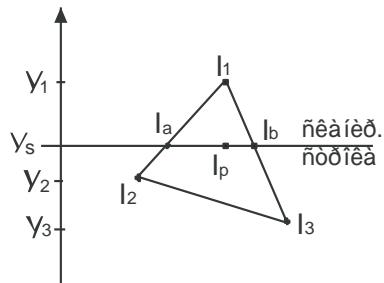


Рис. 13.13

$$I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2}$$

$$I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3}$$

$$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}$$

Интерполяция вдоль ребер легко объединяется с алгоритмом удаления скрытых поверхностей, построенном на принципе построчного сканирования. Для всех ребер запоминается начальное  $i$ , а также изменение  $i$  при каждом единичном шаге по координате  $y$ . Заполнение видимого интервала на сканирующей строке производится путем интерполяции между значениями  $i$  на двух ребрах, ограничивающих интервал. Для цветных объектов отдельно интерполируется каждая из компонент цвета.

### **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  $i$ , а при закраске Фонга – вектор нормали. Затем он используется в модели освещения для вычисления  $\bar{i} \bar{p}$ . При этом достигается лучшая локальная аппроксимация кривизны поверхности, и получается более реалистичное изображение. Особенно правдоподобно выглядят зеркальные блики, которые в методе Гуро сильно размываются.

#### Этапы закраски:

- 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, т.к. аппроксимация нормали осуществляется в каждой точке. Полосы Маха практически исчезают.

### 13.4. Тени.

Изображение с построенными тенями выглядит гораздо реалистичнее, и, 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, при разработке космических аппаратов тени влияют на расчет падающей солнечной энергии, обогрев и кондиционирование воздуха. Если положение наблюдателя и источника света совпадают, то теней не видно, но они появляются, когда наблюдатель перемещается в любую другую точку. Тень стоит из 2-ух частей: полутени и полной тени. Полная тень – центральная, темная, резко очерченная часть, а полутень – окружающая ее более светлая часть. Распределенные источники света создают как 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.

Рассмотрим рисунок. В этом случае тени образуются двояко:

- 1) собственно тень на объекте;
- 2) проекционная тень.

Собственная тень получается тогда, когда сам объект препятствует попаданию света на некоторые его грани. Алгоритм затенения в этом случае идентичен алгоритму удаления скрытых поверхностей. В алгоритме удаления скрытых поверхностей 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 1 алгоритм, последовательно применяя его к точке зрения и к каждому из точечных источников света.

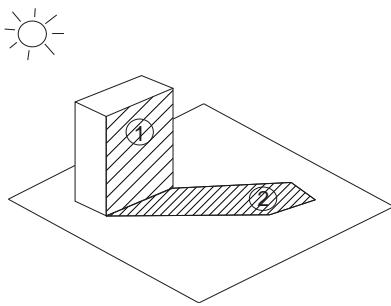


Рис. 13.14

Если один объект препятствует попаданию света на другой, то получается проекционной тень (горизонтальная плоскость на рис.). Чтобы найти такие тени, надо построить проекции граней на сцену. Центр проекции – в источнике света. Т.о. находятся теневые многоугольники для всех граней и 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 надо вычислять тени заново, т.к. они зависят только от положения источника света и не зависят от положения наблюдателя.

### Источник на бесконечности

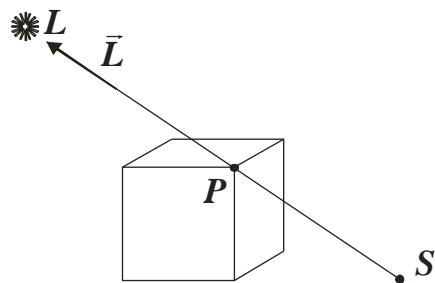


Рис. 13.15

В случае бесконечно удаленного источника света мы предполагаем, что лучи света, приходящие к объекту, полностью параллельны. Это позволит нам решить уравнение проекции только раз и применять полученное решение ко всем вершинам объекта.

### Общая постановка задачи:

Имея точку источника света  $(x_l, y_l, z_l)$  и вершину объекта  $(x_p, y_p, z_p)$ , мы хотим получить проекцию вершины объекта на плоскость  $z=0$ , т.е. точку тени  $(x_s, y_s, z_s)$ .

Из подобных треугольников получаем:

$$\frac{x_p - x_s}{z_p - z_s} = \frac{x_\Delta - x_p}{z_l - z_p} \quad (1.1)$$

Решая это уравнение относительно  $x_s$ , получаем:

$$x_s = x_p - (z_p - z_s) \left( \frac{x_l - x_p}{z_l - z_p} \right) \quad (1.2)$$

Если принять, что  $L$  это вектор из точки  $P$  к источнику света, то точку  $S$  можно выразить как:

$$S = P - \alpha \vec{L} \quad (1.3)$$

Т.к. мы производим проекцию на плоскость  $z=0$ , то уравнение (1.3) можно переписать в следующем виде:

$$0 = z_p - \alpha z_l \quad (1.4)$$

или

$$\alpha = \frac{z_p}{z_l} \quad (1.5).$$

Решая (1.3) относительно  $x_s$  и  $y_s$ , получаем

$$\begin{aligned} x_s &= x_p - \frac{z_p}{z_l} x_l \\ y_s &= y_p - \frac{z_p}{z_l} y_l \end{aligned} \quad (1.6)$$

или в матричной форме:

$$M_T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{x_l}{z_l} & -\frac{y_l}{z_l} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.7)$$

Теперь имея координаты точки  $P$  в мировом координатном пространстве, можно получить ее проекцию на плоскость  $z=0$  просто путем умножения на матрицу  $M_T$

$$S = P \cdot M_T \quad (1.8).$$

### Локальный источник

Уравнение (1.6) для бесконечно удаленного источника света может быть обобщено для случая, когда источник света находится на конечном расстоянии от объекта. В этом случае нам понадобятся дополнительные вычисления на каждую вершину, т.к. каждая вершина имеет, в общем случае, свое собственное направление на источник света. Тем не менее, в этом случае мы тоже можем перенести большую часть вычислений в матрицу  $M_T$ .

Если  $L$  это точка расположения источника света, то (1.3) принимает вид:

$$S = P + \alpha(P - L) \quad (1.9)$$

И снова нам необходимо произвести проекцию на плоскость  $z=0$ , тогда.

$$\alpha = \frac{-z_p}{z_p - z_l} \quad (1.10)$$

и

$$\begin{aligned} x_s &= \frac{x_l z_p - x_p z_l}{z_p - z_l} \\ y_s &= \frac{y_l z_p - y_p z_l}{z_p - z_l} \end{aligned} \quad (1.11)$$

Если использовать гомогенизацию после преобразования, то (1.11) можно записать в виде матрицы:

$$M_T = \begin{bmatrix} -z_l & 0 & 0 & 0 \\ 0 & -z_l & 0 & 0 \\ x_l & y_l & 0 & 1 \\ 0 & 0 & 0 & -z_l \end{bmatrix} \quad (1.12).$$

Опять, имея координаты точки  $P$  в мировом координатном пространстве, можно записать:

$$S = P \cdot M_t \quad (1.13)$$

После чего провести гомогенизацию точки  $S$  для получения проекции точки Р на плоскость  $z=0$ .

### 13.5. Фактура. Нанесение узора.

В машинной графике фактурой называется детализация строения поверхности. Существует 2 вида детализации:

1.Нанесение заданного узора на гладкую поверхность (регулярная и стохастическая текстуры).

2.Создание неровностей на поверхности.

#### **Нанесение узора на поверхность. Регулярная текстура.**

Характерные точки узора из пространства текстуры переносятся в объективное пространство, затем в пространство изображения и определенным образом соединяются отрезками. Главным при этом является отображение, поэтому задача сводится к преобразованию систем координат.

Пусть рисунок узора задан в прямоугольной системе координат  $(u, w)$ , а поверхность – в другой прямоугольной системе координат  $(x, y)$ , то для нанесения узора на поверхность надо найти или задать функцию отображения одного пространства на другое:

$$x = f(u, w), \quad y = g(u, w)$$

или

$$u = h(x, y), \quad w = e(x, y).$$

Обычно предполагается, что функция отображения линейна:

$$\begin{aligned} x &= Au + B \\ y &= Cw + D \end{aligned} \quad \text{после}$$

где коэффициенты А, В, С, Д выводятся из соотношения между двумя известными точками в системах координат.

#### Пример.

Узор на рис. а) надо отобразить на кусок поверхности, заданный десятой частью сферы.

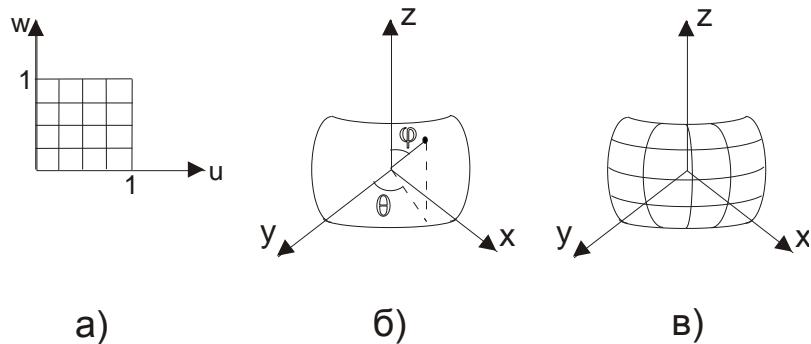


Рис. 13.16

Параметрическое представление куска сферы:

$$\begin{aligned}x &= \sin \theta \sin \varphi & 0 \leq \theta \leq \frac{\pi}{2} \\y &= \cos \theta \sin \varphi \\z &= \cos \varphi & \frac{\pi}{4} \leq \varphi \leq \frac{\pi}{2}\end{aligned}$$

Пусть функция отображения линейна и имеет вид:

$$\theta = Au + B, \quad \varphi = Cw + D.$$

Углы узора переходят в углы куска следующим образом:

$$u = 0, w = 0 \text{ при } \theta = 0, \varphi = \frac{\pi}{2}$$

$$u = 1, w = 0 \text{ при } \theta = \frac{\pi}{2}, \varphi = \frac{\pi}{2}$$

$$u = 0, w = 1 \text{ при } \theta = 0, \varphi = \frac{\pi}{4}$$

$$u = 1, w = 1 \text{ при } \theta = \frac{\pi}{2}, \varphi = \frac{\pi}{4}$$

Отсюда  $A = \frac{\pi}{2}$ ,  $B = 0$ ,  $C = -\frac{\pi}{4}$ ,  $D = \frac{\pi}{2}$ .

Функция отображения:  $\theta = \frac{\pi}{2}u$ ,  $\varphi = \frac{\pi}{2} - \frac{\pi}{4}w$

или обратное преобразование:  $u = \frac{\theta}{\pi/2}$ ,  $w = \frac{\pi/2 - \varphi}{\pi/4}$ .

В таблице приведено отображение одной линии узора из пространства ( $u-w$ ) в пространство ( $\theta-\varphi$ ), а затем в декартовы координаты ( $x,y,z$ ).

Таблица 13.1

$u$	$w$	$\theta$	$\varphi$	$x$	$y$	$z$
$\frac{1}{4}$	0		$\pi/2$	0,38	0,92	0
	$\frac{1}{4}$		$7\pi/16$	0,38	0,90	0,20
	$\frac{1}{2}$	$\pi/2$	$3\pi/8$	0,35	0,85	0,38
	$\frac{3}{4}$		$5\pi/16$	0,32	0,77	0,56
	1		$\pi/4$	0,27	0,65	0,71

### Нанесение узора на поверхность. Стохастическая текстура.

Используется метод обратного трассирования лучей. Центр каждого  $\bar{p}$  изображения проецируется на поверхность объекта и по координатам т. на поверхности определяется соответствующая ей т. в пространстве фактурном. Далее используются процедуры сглаживания для устранения дискретизации.

Рассмотренный узор был задан математически, но он м.б. также нарисован от руки или получен путем сканирования фотографий. Для нанесения рис. на поверхность необходимо:

- отображение объектного пространства (ОП) в пространство изображения (ПИ);
- преобразование из фактурного пространства (ФП) в ОП.

Рассмотрим алгоритм разбиения Кэтмула:

- 1) Кусок поверхности разбивается на фрагменты до тех пор, пока фрагмент не будет покрывать центр только одного  $\bar{p}$ .
- 2) Производится отображение параметрических значений центра фрагмента или  $\bar{p}$  в ФП.
- 3) Находится интенсивность  $\bar{p}$  по узору.

Пример. (Исходные данные те же)

Узор задан на растре  $64 \times 64 \bar{p}$ .

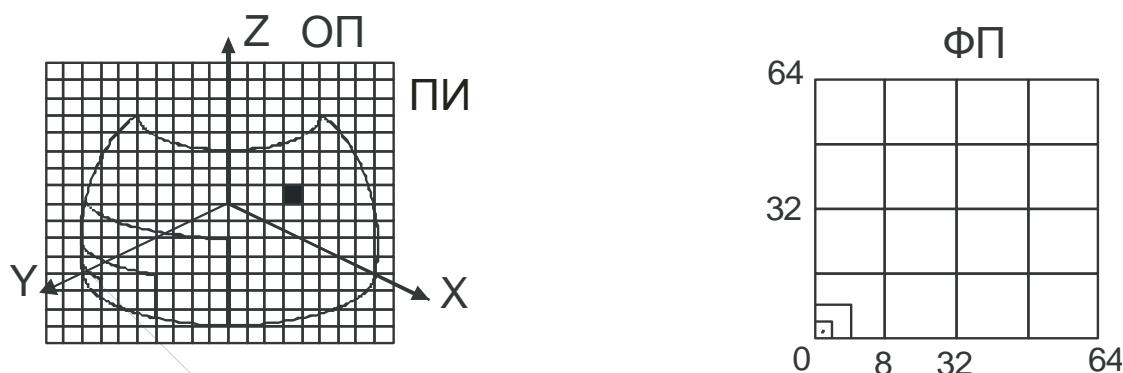


Рис. 13.17

Кусок поверхности разбиваем на фрагменты. Для того, чтобы фрагмент покрывал центр только одного  $\bar{p}$ , надо 4 разбиения. В ПИ этот фрагмент имеет прямоугольную форму. Пределы изменения  $\theta$  и  $\varphi$  в ОП:

$$0 \leq \theta \leq \frac{\pi}{32} \quad \frac{31\pi}{64} \leq \varphi \leq \frac{\pi}{2}$$

С помощью функции обратного отображения из ОП  $(\theta - \varphi)$  в ФП  $(u - w)$ :

$$u = \frac{\theta}{\pi/2}, \quad w = \frac{\pi/2 - \varphi}{\pi/4}$$

получим координаты углов фрагмента в ФП:

$$\begin{aligned} \theta = 0, \varphi = \frac{\pi}{2} &\rightarrow u = 0, \quad w = 0 \\ \theta = 0, \varphi = \frac{31\pi}{64} &\rightarrow u = 0, \quad w = \frac{1}{16} \\ \theta = \frac{\pi}{32}, \varphi = \frac{31\pi}{64} &\rightarrow u = \frac{1}{16}, \quad w = \frac{1}{16} \\ \theta = \frac{\pi}{32}, \varphi = \frac{\pi}{2} &\rightarrow u = \frac{1}{16}, \quad w = 0 \end{aligned}$$

В ФП – это квадрат. На растре  $64 \times 64$  часть  $1/16$  соответствует 4  $\bar{p}$ . Интенсивность  $\bar{p}$  в ПИ определяется путем усреднения интенсивностей  $\bar{p}$  в соответствующей части ФП. Кусок фрагмента  $4 \times 4 \bar{p}$  содержит 7 черных  $\bar{p}$ , поэтому в ПИ интенсивность  $\bar{p} = \frac{7}{16}$ .

Недостаток: такой метод поточечной выборки приводит к сильному лестничному эффекту.

### 13.6. Создание неровностей на поверхности.

Рассматривая рисунок, единственный способ определить, что изображение отображает рельефную поверхность, — это проанализировать изменения яркости отдельных участков изображения. Наш мозг делает это автоматически, незаметно для нас. В результате мы четко определяем, что будет выпуклостью, что впадиной.

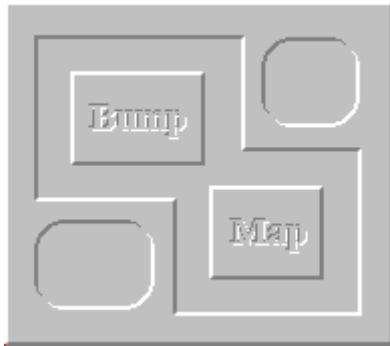


Рис. 13.18

Очень похоже на выдавливание (чеканку). Но, по сути, единственное, что было сделано для придания объемности плоскому изображению, — это правильное наложение ярких и темных участков. Остальное делает наш мозг.

Но как определить, какие биты изображения делать яркими, и наоборот. Очень просто. Большинство людей в течение своей жизни продолжительно находятся в условиях, когда свет исходит сверху. Таким образом, человек привык, что большинство поверхностей сверху ярко освещены, а снизу, наоборот, находятся в тени и будут темнее. Таким образом, если лаз воспринимает светлые и темные области на объекте, то человек воспринимает их как рельеф.

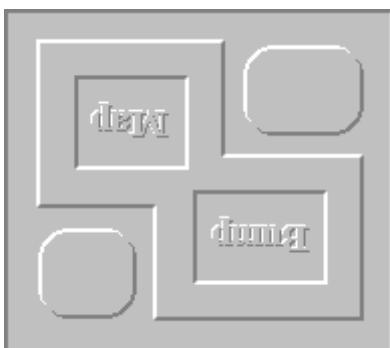


Рис. 13.19

Посмотрев на тот же рисунок, только развернутый на 180 градусов, можно заметить, что он стал похож на полную противоположность предыдущему. То, что раньше казалось выпуклым, стало вогнутым, и наоборот. А ведь это то же самое изображение. Тем не менее, наш мозг все же не настолько глуп. Если вы сможете заставить себя подумать, что свет исходит снизу, мозг воспроизведет информацию так же, как на первом изображении.

Для того, чтобы поверхность казалась шероховатой, можно оцифровать фотографию нерегулярной фактуры и отобразить ее на поверхность. Но при этом будет казаться, что неровности нарисованы на гладкой поверхности.

### **Рельефное текстурирование**

Первый метод основан на использовании рельефных карт (Bump Mapping).

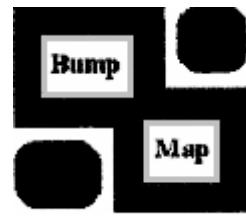


Рис. 13.20

Рельефная карта — это обычная текстура, только в отличие от первой, несущей информацию о цвете определенных участков, рельефная карта несет информацию о неровностях. Самый распространенный способ представить неровности - это применить карту высот. Карта высот — это текстура в оттенках серого, где яркость каждого пикселя представляет, насколько он выдается из базовой поверхности.

Используя карту высот, можно имитировать неровности практически любой поверхности: дерево, камень, шелущающуюся краску и т.д. Конечно, у всего есть свои пределы. Используя bump mapping, нельзя имитировать крупные впадины и возвышенности, но вот для имитации неровностей и шероховатостей на поверхности этот метод подходит идеально.

По сути, это логическое продолжение техники просчета по Фонгу, где интерполируется нормаль к поверхности по всему полигону, и этот вектор используется для дальнейшего определения яркости соответствующего пикселя. Реализуя его, немного меняется направление вектора нормали, основываясь на информации, содержащейся в карте высот. Изменяя положение вектора нормали в конкретной точке полигона, меняется яркость текущего пикселя (закон косинуса из теории света).

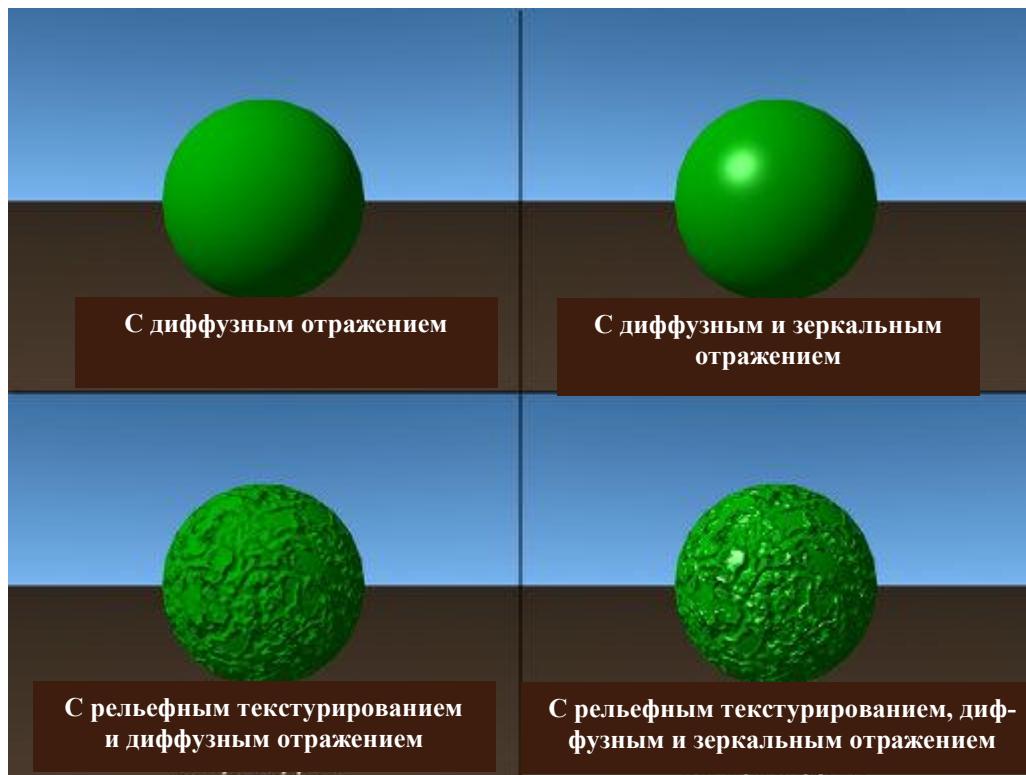


Рис. 13.21

Для того, чтобы этого достичь, существует несколько путей. Вначале необходимо преобразовать информацию о высоте неровностей на карте высот в информацию о величине подстройки вектора нормали. Для этого нужно преобразовать высоты с карты в маленькие векторы. Более светлые квадраты соответствуют более выпуклым участкам. После этого для каждого пикселя рассчитывается вектор, указывающей направление уклона поверхности. Рисунок 13.22 демонстрирует нам это. Маленькие векторы указывают на уменьшение высоты.

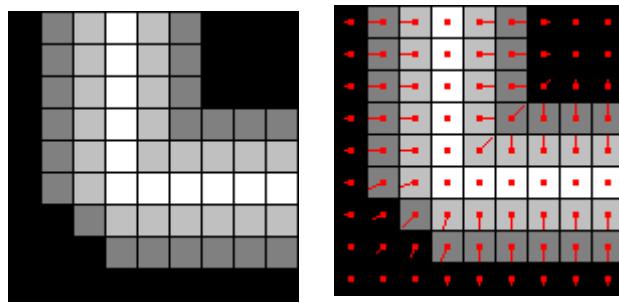


Рис. 13.22

Для определения этих векторов определяется величина градиента для каждого пикселя:

$$x\_gradient = \text{pixel}(x-1, y) - \text{pixel}(x+1, y) \quad y\_gradient = \text{pixel}(x, y-1) - \text{pixel}(x, y+1),$$

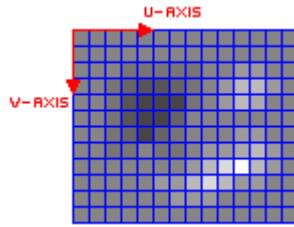


Рис. 13.23

где  $x$  и  $y$  — координаты соответствующего пикселя.

Теперь, имея в руках значения градиентов, можно подкорректировать вектор нормали в соответствующем пикселе.

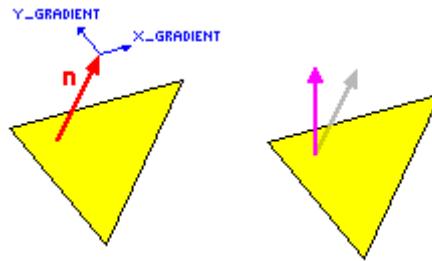


Рис. 13.24

На рис. 13.24 показан полигон с изначальным вектором нормали, обозначенным  $n$ . Также показаны два вектора, которые будут использованы для изменения положения (направления) нормали к пикслю под ним. Оба вектора должны располагаться параллельно осям координат применяемой карты высот.

На рисунке ??? показаны карта высот и полигон. На обоих рисунках показаны направления U и V осей координат карты (текстуры) высот.

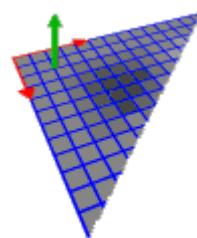


Рис. 13.25

Пересчитать новый вектор нормали легко:

$$\text{New_Normal} = \text{Normal} + (U * \text{x\_gradient}) + (V * \text{y\_gradient})$$

Получив новый вектор нормали, можно просчитать яркость данного пикселя, используя ранее изученную технологию затенения по Фонгу.

Для затенения по Фонгу применялся способ с заранее просчитанной картой, которая представляет собой набор яркостей для всех возможных нормалей на полигоне. Таким образом, быстро выполняемым вариантом рельефного текстурирования будет просчет смещений к карте затенения по Фонгу. Результирующей яркостью пикселя будет сумма значений, полученных с карты затенения и рельефной карты. Используя этот подход, мы одновременно будем производить затенение по Фонгу с учетом рельефности рисунка.

### **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 т. (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 вычисляются методом конечных разностей.

В то время как бампмаппинг всего лишь изменяет существующую нормаль для точек поверхности, нормалмаппинг полностью заменяет нормали при помощи 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 8-битных черно-белых карт высот в бампмаппинге.

В общем, как и бампмаппинг, это тоже "дешевый" метод для добавления детализации к моделям сравнительно низкой геометрической сложности, без использования большего количества реальной геометрии, только более продвинутый. Одно из наиболее интересных применений техники - существенное увеличение детализации низкополигональных моделей при помощи карт нормалей, полученных обработкой такой же модели высокой геометрической сложности. Карты нормалей содержат более подробное описание поверхности, по сравнению с бампмаппингом и позволяют представить более сложные формы.

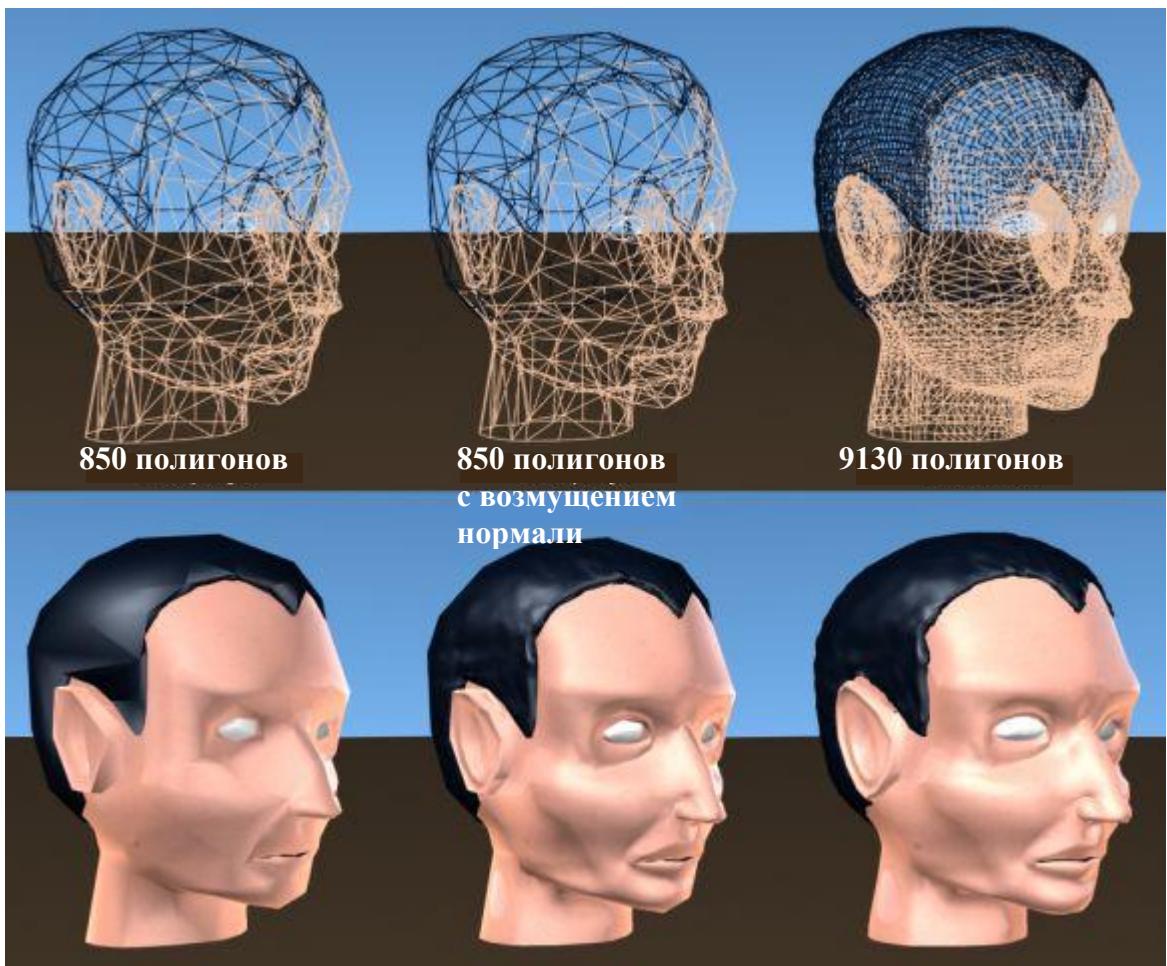


Рис. 13.26

Карты нормалей предоставляют более эффективный способ для хранения подробных данных о поверхностях, по сравнению с простым использованием большого количества полигонов. Единственное серьезное их ограничение в том, что они не очень хорошо подходят для крупных деталей, ведь нормалмаппинг на самом деле не добавляет полигонов и не изменяет форму объекта, он только создает видимость этого. Это всего лишь симуляция деталей, на основе расчета освещения на пиксельном уровне. На крайних полигонах объекта и больших углах наклона поверхности это очень хорошо заметно. Поэтому наиболее разумный способ применения нормалмаппинга 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, при помощи специальных утилит они сравниваются друг с другом, разница рассчитывается и сохраняется в текстуре, называемой картой нормалей. При ее создании дополнительно можно использо-

вать и bump map для очень мелких деталей, которые даже в высокополигональной модели не смоделировать (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 тексель в карте нормалей представлен как нормаль точки поверхности.

Замечание1: Эффект шероховатости зависит от масштаба изображаемого объекта. Если размер объекта в увеличивается в 2 раза, то величина вектора нормали увеличивается в 4 раза, а его возмущения — только в 2 раза. Это приведет к тому, что увеличенный объект будет казаться более гладким. Но масштаб фактуры не зависит на перспективном изображении от перемещения объекта в пространстве по направлению к т. зрения или от нее.

Замечание 2: 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. Надо рассчитывать изображение с разрешением, большим, чем у дисплея, а затем отфильтровать или усреднить его и вывести с более низким разрешением экрана.

### Использование фрактальных поверхностей

Второй 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 (рис.). Для этого можно, например, случайным образом сместить центр и середины сторон многоугольника, причем исходный и полученный многоугольники не обязательно д.б. плоскими.

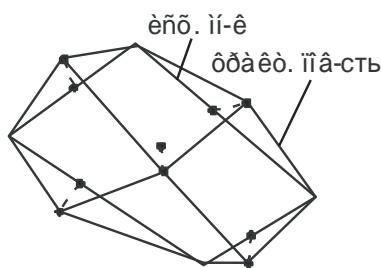


Рис. 13.27

Одно из преимуществ фрактальных поверхностей в том, что их можно разбивать бесконечно и получить 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 со скоростью выше линейной, поэтому между количеством разбиений и уровнем детализации д.б. 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 бампмаппинга, когда картами высот правильно моделируется только 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. Этот метод изменяет положение вершин треугольников, сдвигая их по нормали на величину, исходя из значений в картах смещения. Карта смещения - это обычно черно-белая текстура, и значения в ней используются для определения высоты каждой точки поверхности объекта (значения могут храниться как 8-битные или 16-битные числа), схоже с bumpmap. Часто карты смещения используются (в этом случае они называются и картами высот) для создания земной поверхности с холмами и впадинами. Так как рельеф местности описывается двухмерной картой смещения, его относительно легко деформировать при необходимости, так как это потребует всего лишь модификации карты смещения и рендеринга на ее основе поверхности в следующем кадре.

Наглядно создание ландшафта при помощи наложения карт смещения представлено на картинке. Исходными были 4 вершины и 2 полигона, в итоге получился полноценный кусок ландшафта.

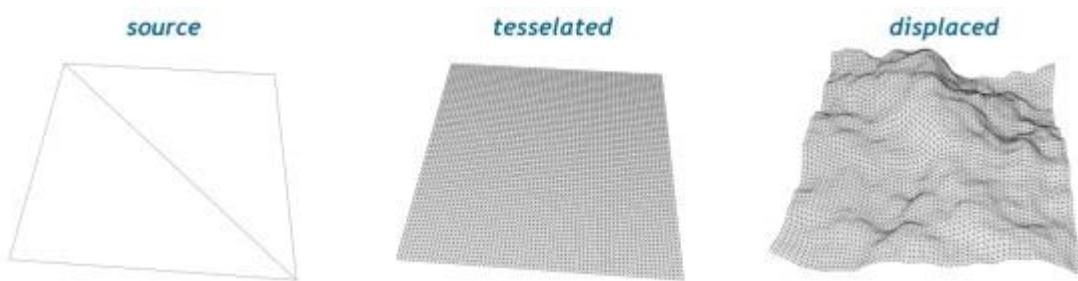


Рис. 13.27

Большим преимуществом наложения карт смещения является не просто возможность добавления деталей к поверхности, а практический полный 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. Получается в итоге сложный 3D объект из простого:



Рис. 13.28

Количество треугольников, должно быть достаточно большим для того, чтобы передать все детали, задаваемые картой смещений. Карты смещения лучше использовать совместно с бампмаппингом для создания мелких деталей, где достаточно правильного попиксельного освещения.

Наложение карт смещения можно по существу считать методом сжатия геометрии, использование карт смещения снижает объем памяти, требуемый для определенной детализации 3D модели. Громоздкие геометрические данные замещаются простыми двухмерными текстурами смещения, обычно 8-битными или 16-битными. Это снижает требования к объему памяти и пропускной способности. Другое преимущество в том, что применение карт смещения превращает сложные полигональные трехмерные сетки в несколько двухмерных 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. Есть также проблемы удобства их применения, это почти всегда требует специализированных утилит, ведь очень сложно напрямую создавать карты смещения (если речь не идет о простых объектах, вроде ландшафта). 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.

Фильтрация решает задачи определения цвета пикселя на базе имеющихся текстелей из текстурного изображения.

Простейший метод наложения текстур называется поточечная выборка (single point-sampling). Суть его в том, что для каждого пикселя, составляющего полигон, выбирается один текстель из текстурного изображения, ближе всех рас-

положенный к центру светового пятна. Совершается ошибка, так как цвет пикселя определяют несколько текстелей, а выбран был только один.

Этот метод очень неточен и результатом его применения является появление неровностей. А именно, всякий раз, когда пиксели больше по размеру, чем текстели, наблюдается эффект мерцания. Этот эффект имеет место, если часть полигона достаточно удалена от точки наблюдения, так, что сразу много текстелей накладываются на пространство, занимаемое одним пикселием. Заметим, что если полигон расположен очень близко к точке наблюдения и текстели больше по размеру, чем пиксели, наблюдается другой тип ухудшения качества изображения. В данном случае, изображение начинает выглядеть блочным. Этот эффект имеет место, когда текстура может быть достаточно большой, но ограничение в виде доступного разрешения экрана не дает возможности правильно представить исходное изображение.

Второй метод - билинейная фильтрация (Bi-Linear Filtering) состоит в использовании интерполяционной техники. Для определения текстелей, которые должны быть задействованы для интерполяции, используется основная форма светового пятна -- круг. По существу, круг аппроксимируется 4 текстелями. Билинейная фильтрация - это техника устранения искажений изображения (фильтрация), таких, как "блочности" текстур при их увеличении. При медленном вращении или движении объекта (приближение/удаление) могут быть заметны "перескакивания" пикселов с одного места на другое, т.е. появляется блочность. Во избежании этого эффекта применяют билинейную фильтрацию, при использовании которой для определения цвета каждого пикселя берется взвешенное среднее значение цвета четырех смежных текстелей и в результате определяется цвет накладываемой текстуры. Результирующий цвет пикселя определяется после осуществления трех операций смещивания: сначала смещиваются цвета двух пар текстелей, а потом смещиваются два полученных цвета.

Главный недостаток билинейной фильтрации в том, что аппроксимация выполняется корректно только для полигонов, которые расположены параллельно экрану или точке наблюдения. Если полигон развернут под углом (а это в 99% случаев), используется неправильная аппроксимация, так как должен аппроксимироваться эллипс.

Ошибки "depth aliasing" возникают в результате того факта, что объекты более удаленные от точки наблюдения, выглядят более маленькими на экране. 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, текстурное изображение, наложенное на объект, становится настолько сжатым, что появляются ошибки визуализации. Эти ошибки визуализации особенно нежелательны в анимации, где такие артефакты во время движения становятся причиной мерцания и эффекта медленного движения в той части изображения, которая должна быть неподвижной и стабильной.

В качестве иллюстрации к описанному эффекту могут служить следующие прямоугольники с билинейным текстурированием:

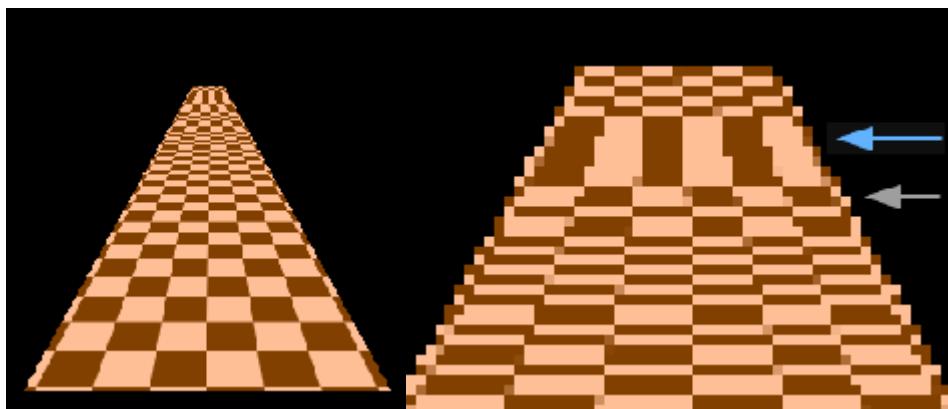


Рис. 13.29. Закраска объекта методом билинейной фильтрации. Появление артефактов "depth-aliasing", выражющихся в том, что несколько квадратов сливаются в один.

Для избежания ошибок и имитации того факта, что объекты на расстоянии выглядят менее детализированными, чем те, что находятся ближе к точке наблюдения, используется техника, известная как mip-mapping. Если говорить кратко, то mip-mapping - наложение текстур, имеющих разную степень или 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-тап, пиксель представляется в виде среднего четырех пикселей из предыдущего уровня с более высоким разрешением. Отсюда, изображение связанное с каждым уровнем mip-текстуры в четыре раза меньше по размеру предыдущего mip-тап уровня.

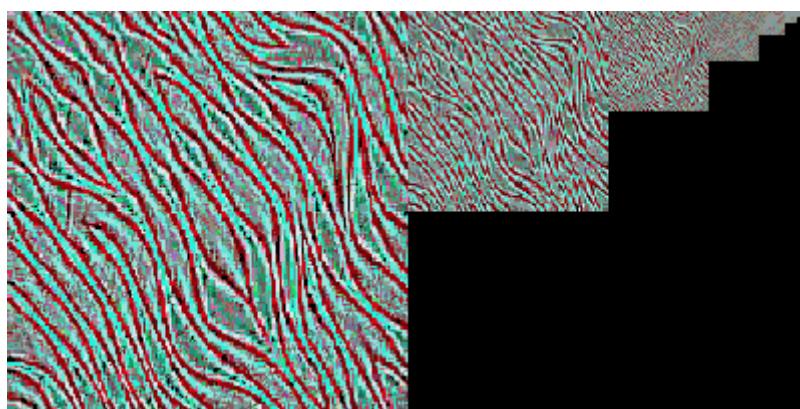


Рис. 13.30. Изображения, связанные с каждым mip-тап уровнем волнообразной текстуры.

Слева направо мы имеем mip-тап 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 серых пикселей.

Степень или уровень детализации - Level of Detail или просто LOD, используются для определения, какой mipmap уровень (или какую степень детализации) следует выбрать для наложения текстуры на объект. LOD должен соответствовать числу текселей накладываемых на пиксель. Например, если текстурирование происходит с соотношением близким к 1:1, то LOD будет 0, а значит и будет использоваться mipmap уровень с самым высоким разрешением. Если 4 текселя накладываются на один пиксель, то LOD будет 1 и будет использоваться следующий mipmap уровень с меньшим разрешением. Обычно, при удалении от точки наблюдения, объект, заслуживающий наибольшего внимания имеет более высокое значение LOD.

В то время, как mipmap-текстурирование решает проблему ошибок "depth-aliasing", его использование может стать причиной появления других артефактов. При удалении объекта все дальше от точки наблюдения, происходит переход от низкого mipmap уровня к высокому. В момент нахождения объекта в переходном состоянии от одного mipmap уровня к другому, появляется особый тип ошибок визуализации, известных под названием "mip-banding" - полосатость или слоенность, т.е. явно различимые границы перехода от одного mipmap уровня к другому.

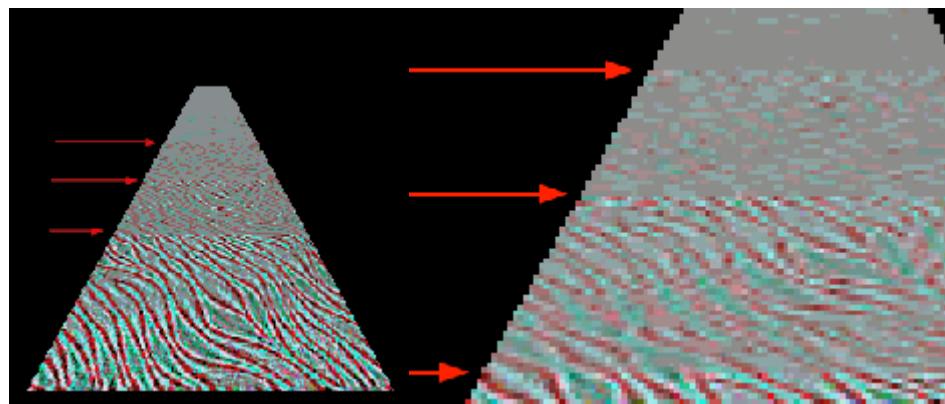


Рис. 13.31. Прямоугольная лента состоит из двух треугольников, текстурированных волнообразным изображением, где "mip-banding" артефакты обозначены красными стрелками.

Особенно остро проблема наличия ошибок "mip-banding" стоит в анимации, за счет 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 "mip-banding", возникающие при использовании mipmap-текстурирования. При трилинейной фильтрации для определения цвета пикселя берется среднее значение цвета восьми текселей, по четыре из двух соседних текстур и в результате семи операций смешивания определяется цвет пикселя. При использовании трилинейной фильтрации возможен вывод на экран текстурированного объекта с плавно выполнеными переходами от одного mipmap уровня

ня к следующему, что достигается за счет определения LOD путем интерполяции двух соседних mip-map уровней. Таким образом решая большинство проблем, связанных с mip-текстурированием и ошибками из-за неправильного расчета глубины сцены ("depth aliasing").

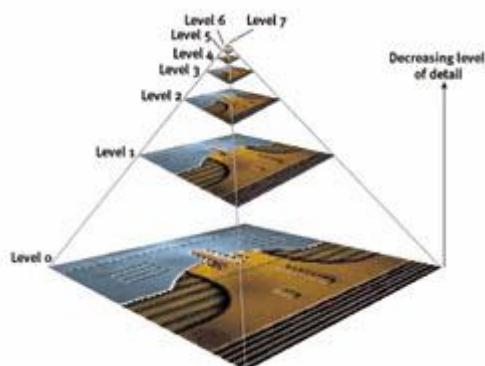


Рис. 13.32. Пирамидальность MIP-мап

Пример использования трилинейной фильтрации приведен ниже. Здесь опять используется все тот же прямоугольник, текстурированный волнообразным изображением, но с плавными переходами от одного mip уровня к следующему за счет использования трилинейной фильтрации. Обратите внимание на отсутствие каких-либо заметных ошибок визуализации.

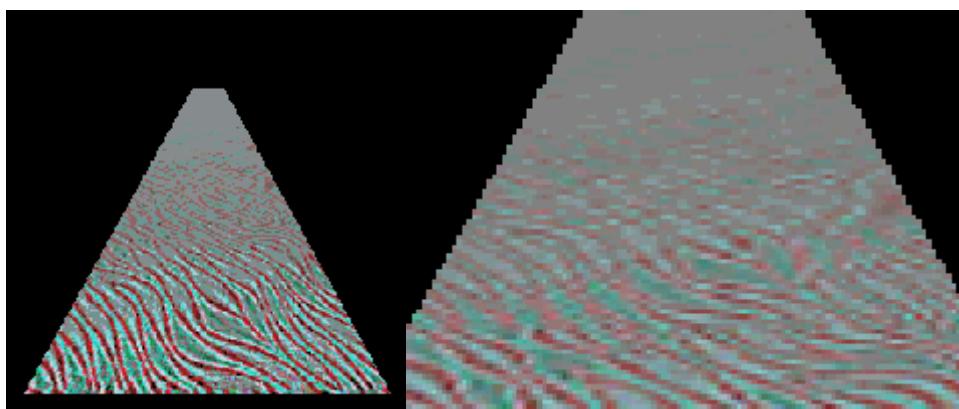


Рис. 13.33. Прямоугольник, текстурированный волнообразным изображением, выведен на экран с использованием mip-текстурирования и трилинейной фильтрации.

Существует несколько способов генерации MIP текстур. Один из них - просто подготовить их заранее, используя графические пакеты типа Adobe PhotoShop. Другой способ - генерация MIP текстур на "лету", т.е. в процессе выполнения программы. Заранее подготовленные MIP текстуры означают дополнительные 30% дискового пространства для текстур в базовой поставке инсталляции игры, но позволяют применять более гибкие методы управления их созданием и позво-

ляют добавлять различные эффекты и дополнительные детали различным MIP уровням.

Получается, что трилинейный mipmapping это лучшее, что может быть?

Нет конечно. Видно, что проблема не только в соотношении размеров пикселя и текселя, но также и в форме каждого из них (или, что бы быть более точными, в соотношениях форм).

Метод mipmap-текстурирования лучше всего работает для полигонов расположенных прямо "лицом к лицу" к точке наблюдения. Однако, полигоны, косо направленные по отношению к точке наблюдения искривляют накладываемую текстуру так, что на пиксели могут накладываться различного вида и квадратичные по форме области текстурного изображения. Метод mipmap-текстурирования не принимает это во внимание и в результате наблюдается эффект слишком сильного размытия текстурного изображения, так, будто использованы неправильно выбранные тексели. Для решения этой проблемы нужно делать выборку из большего количества текселей, составляющих текстуру, и выбирать эти тексели следует принимая во внимание "отображенную" форму пикселя в текстурном пространстве. Этот метод называется анизотропная фильтрация ("anisotropic filtering"). Обычное mipmap-текстурирование называется "isotropic" (изотропное или однородное), потому что мы всегда фильтруем вместе квадратные области, состоящие из текселей. Анизотропная фильтрация означает, что форма области из текселей, которую мы используем меняется в зависимости от обстоятельств.

### 13.8. Полутоновые изображения.

Аппроксимация полутонами – это метод, в котором используется минимальное число уровней интенсивности, обычно белый и черный, для улучшения визуального изображения, т.е. получения нескольких полутонов серого или уровней интенсивности.

Этот метод известен давно. Первоначально он использовался при изготовлении шелковых картин и других текстильных изделий. В 1880 г. Хагеном была изобретена современная полутоновая печать. В этом методе можно получать большое количество фотографических полутонов серого, используя чисто двухуровневую среду: черную краску на белой бумаге. Полутоновая печать – клеточный процесс. Для газетных фотографий из-за низкого качества бумаги применяются клетки 50-90 т/дюйм. Бумага более высокого качества для книг и журналов позволяет использовать клетки 100-300 т/дюйм. Успех метода полутонов зависит от свойства человеческого глаза быть интегратором, т.е. объединять или сглаживать дискретную информацию.

Визуальное изображение машинно-сгенерированных изображений можно улучшить методом конфигурирования. В противоположность полутоновой печати, в которой используются переменные размеры клеток, в данном методе размеры клеток фиксированы. Несколько  $\bar{r}$  объединяются в конфигурации. Происходит ухудшение пространственного разрешения за счет улучшения визуального.

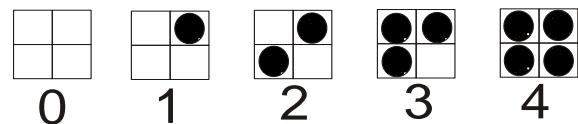


Рис. 13.34

Например, для каждой клетки используется  $4 \bar{p}$ . Т.о. получается 5 уровней серого. В общем случае:

$$n_i = n_{\bar{p}} + 1$$

$n_i$  - число уровней  $\bar{I}$ ,

$n_{\bar{p}}$  - число  $\bar{p}$  в клетке.

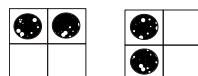


Рис. 13.35

При выборе конфигураций надо проявлять осторожность, т.к. могут возникнуть нежелательные мелкомасштабные структуры. Например, не надо применять следующие конфигурации:

иначе это приведет к появлению нежелательных горизонтальных и вертикальных линий. Число уровней  $\bar{I}$  может увеличиваться с помощью увеличения размера клетки. Клетки конфигурации не обязательно д.б. квадратными. Т.к. 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.

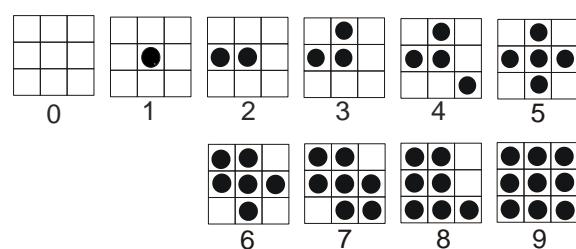


Рис. 13.36

Простейший – применение порогового значения для каждого  $\bar{p}$ . Если интенсивность больше некоторой пороговой величины, то  $\bar{p}$  горит, иначе – не горит.

Пороговая величина ( $T$ ) обычно:

$$T = \frac{1}{2} o_m I_{\max}.$$

$$T \approx \frac{1}{2} I_{\max}$$

если  $I(x, y) > T$  то пиксел горит,

иначе не горит.

При простом пороговом методе наблюдается низкое качество. Будут теряться мелкие детали, а изображение шара, освещенного светом, будет иметь внутри белый круг. Улучшить изображение помогает метод порога с переносом, где ошибка не отбрасывается, а распределяется на следующий  $\bar{p}$ .

Если  $I(x_i, y_i) < T$ , то пиксел не горит

ошибка =  $I(x, y)$

$$\hat{I}(x_{i+1}, y_{i+1}) = I(x_{i+1}, y_{i+1}) + ou$$

иначе горит

ошибка =  $I(x, y) - 2T$

$$\hat{I}(x_{i+1}, y_{i+1}) = I(x_{i+1}, y_{i+1}) + ou$$

Пример.

Пусть  $I \in [0; 29]$

$T = 14,5$

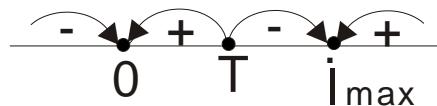


Рис. 13.36

Ряд  $I$ :

Таблица 13.2

	11	13	15	17	19	21	23	23	21	18
Ошибки		(11)	(-5)	(10)	(-2)	(-12)	(9)	(3)	(-3)	(-11)
Простое ограничение	.	.	L	L	L	L	L	L	L	L
Метод порога с переносом	.	L	.	L	L	.	L	L	L	.
14	10	6	5	5	6	8	10	10	10	
(7)	(-8)	(2)	(8)	(13)	(-11)	(-5)	(3)	(13)	(-6)	
.	.	.	.	.	.	.	.	.	.	
L	.	.	.	L	.	.	.	L	.	

## 14. ТРАССИРОВКА ЛУЧЕЙ

### 14.1 Метод прямой трассировки

Теперь рассмотрим то, как формируется изображение некоторой сцены, включающей в себя несколько пространственных объектов. Будем полагать что из точек поверхности (объема) излучающих объектов исходят лучи. Можно назвать такие лучи первичными — они освещают всё остальное. Важным моментом является предположение, что световой луч в свободном пространстве распространяется *вдоль прямой линии* (хотя в специальных разделах физики изучаются также и причины возможного искривления). Но в *геометрической оптике* полагают, что луч света распространяется прямолинейно до тех пор, пока не встретится отражающая поверхность или граница среды преломления. Так будем полагать и мы.

От источников излучения исходит по различным направлениям бесчисленное множество первичных лучей (даже луч лазера невозможно идеально сфокусировать — все равно свет будет распространяться не одной идеально тонкой линией, а конусом, пучком лучей). Некоторые лучи уходят в свободное пространство, а некоторые (их также бесчисленное множество) попадают на другие объекты. Если луч попадает в прозрачный объект, то, преломляясь, он идет дальше, при этом некоторая часть световой энергии поглощается. Подобно этому если на пути луча встречается зеркально отражающая поверхность, то он также изменяет направление, а часть световой энергии поглощается. Если объект зеркальный и одновременно прозрачный (например, обычное стекло), то будет уже два луча — в этом случае говорят, что луч расщепляется.

Можно сказать, что в результате действия на объекты первичных лучей возникают вторичные лучи. Бесчисленное множество вторичных лучей уходит в свободное пространство, но некоторые из них попадают на другие объекты. Так многократно отражаясь и преломляясь, отдельные световые лучи приходят в точку наблюдения — глаз человека или оптическую систему камеры. 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 спектром и интенсивностью первичных лучей источников и интенсивностью, а также поглощением световой энергии в объектах, встретившихся на пути соответствующих лучей.

Непосредственная реализация данной лучевой модели формирования изображения представляется затруднительной. В таком алгоритме необходимо предусмотреть перебор всех первичных лучей и определить те из них, которые попадают в объекты и в камеру. Затем выполнить перебор всех вторичных лучей, и также учесть только те, которые попадают в объекты и в камеру. И так далее. Можно назвать такой метод *прямой трассировкой* лучей. Практическая ценность такого метода вызывает сомнения. В самом деле, как учитывать бесконечное множество лучей, идущих во все стороны? Очевидно, что полный перебор бесконечного числа лучей в принципе невозможен. 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), всё равно остаётся главный недостаток метода — много лишних операций, связанных с расчётом лучей, которые затем не используются. Так, во всяком случае, это представляется в настоящее время.

### Метод обратной трассировки

Метод *обратной трассировки* лучей позволяет значительно сократить перебор световых лучей. Метод разработан в 80-х годах, основополагающими считаются работы Уиттеда и Кея. Согласно этому методу отслеживание лучей производится не от источников света, а в обратном направлении — точки наблюдения. Так учитываются только те лучи, которые вносят вклад в формирование изображения.

Рассмотрим, как можно получить растровое изображение некоторой трёхмерной сцены методом обратной трассировки. Предположим, что плоскость проецирования разбита на множество квадратиков — пикселов. Выберем центральную проекцию с центром схода на некотором расстоянии от плоскости проецирования. Проведем прямую линию из центра схода через середину квадрата (пикселя) плоскости проецирования. Это будет первичный луч обратной трассировки. Если прямая линия этого луча попадает в один или несколько объектов сцены, то выбираем ближайшую точку пересечения. Для определения цвета пикселя изображения нужно учитывать свойства объекта, а также то, какое световое излучение приходится на соответствующую точку объекта.

Если объект зеркальный (хотя бы частично), то строим вторичный луч — луч падения, считая лучом отражения предыдущий, первичный трассируемый луч. Выше мы рассматривали зеркальное отражение и получили формулы для вектора отражённого луча по заданным векторам нормали и луча падения. Но здесь нам известен вектор отражённого луча, а как найти вектор падающего луча? Для этого можно использовать ту же формулу зеркального отражения, но определяя необ-

ходимый вектор луча падения как отражённый луч. То есть отражение наоборот,  $i$ .

Для идеального зеркала достаточно затем проследить лишь очередную точку пресечения вторичного луча с некоторым объектом. Неидеальное зеркало резко усложняет трассировку — нужно проследить не один, а множество падающих лучей, учитывать вклад излучения от других видимых из данной точки объектов.

Если объект прозрачный, то необходимо построить новый луч, такой, который при преломлении давал бы предыдущий трассируемый луч. Здесь также можно воспользоваться обратимостью, которая справедлива 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.

Когда выясняется; что текущий луч обратной трассировки не пересекает какой-либо объект, а уходит в свободное пространство, то на этом трассировка для этого луча заканчивается.

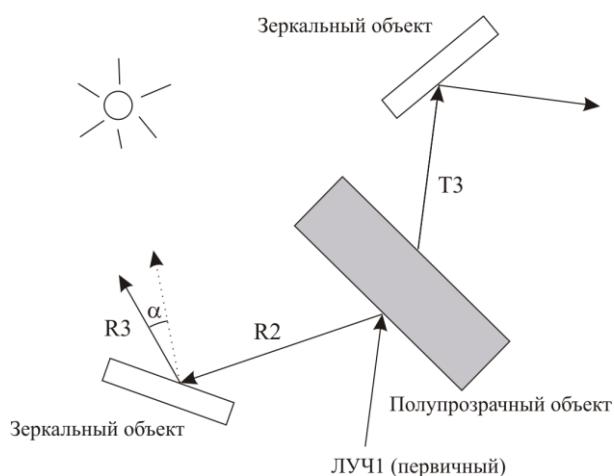


Рис. 14.1 Пример обратной трассировки лучей.

Обратная трассировка лучей в том виде, в котором мы её здесь рассмотрели, хоть и сокращает перебор, но не позволяет избавиться от бесконечности числа 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 бесконечное количество. Так, например, если объект может отражать свет от любого другого объекта, и если эти другие объекты имеют достаточно большие размеры, то какие именно точки излучающих объектов нужно учитывать для построения соответствующих лучей, например, при диффузном отражении? Очевидно, все точки.

### *Принцип работы метода трассировки лучей:*

1. Испускается воображаемый луч из глаза наблюдателя через некоторый пиксель экрана и отслеживается его путь, пока он не пересечет объект.

2. Из первой точки пересечения луча со сферой испускается отраженный луч. Пусть поверхность непрозрачна. Тогда преломленные лучи не рисуем. Обозначаем луч тени от точки пересечения к источнику света. Так как этот луч не пересекает другую непрозрачную поверхность, то источник света непосредственно влияет на интенсивность освещения в данной точке.

3. Пусть отраженный луч пересекает другой объект, на этот раз полупрозрачную сферу, которая отражает и пропускает свет. 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. Пусть точка, в которой луч пересекает сферу, не будет прямо освещена источником, потому что путь теневого луча преграждает непрозрачная поверхность. Если бы сцена содержала несколько источников света, то теневые лучи должны были бы быть пущены в каждый из них.

5. Влияние всех лучей, сгенерированных явно или неявно с помощью начального луча, суммируется и результат определяет RGB-значение данной точки.

### **Реализация метода обратной трассировки**

При практической реализации метода обратной трассировки вводят ограничения. Некоторые из них необходимы, чтобы можно было в принципе решить задачу синтеза изображения, а некоторые ограничения позволяют значительно повысить быстродействие трассировки. Рассмотрим примеры таких ограничений.

1. Среди всех типов объектов выделим некоторые, которые назовём *источниками света*. Источники света могут только излучать свет, но не могут его отражать или преломлять. Будем рассматривать только точечные источники света.

2. Свойства отражающих поверхностей описываются суммой двух компонент — диффузной и зеркальной.

3. В свою очередь, зеркальность тоже описывается двумя составляющими. Первая (*reflection*) учитывает отражения от других объектов, не являющихся источниками света. Строится только один зеркальный луч  $g$  для дальнейшей трассировки. Вторая компонента (*specular*) означает световые блики от источников света. Для этого направляются лучи на все источники света и определяются углы, образуемые этими лучами с зеркально отражённым лучом обратной трассировки ( $r$ ). 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 освеще-

щённой точки поверхности определяется собственным цветом поверхности и цветом источников света.

5. Для прозрачных (*transparent*) объектов обычно не учитывается зависимость коэффициента преломления от длины волны. Иногда прозрачность вообще моделируют без преломления, то есть направление преломленного луча  $t$  совпадает с направлением падающего луча.

6. Для учёта освещённости объектов светом, рассеиваемым другими объектами, вводится фоновая составляющая.

7. Для завершения трассировки вводят некоторое пороговое значение освещённости, которое уже не должно вносить вклад в результирующий цвет, либо ограничивают количество итераций.

Согласно модели Уиттеда цвет некоторой точки объекта определяется суммарной интенсивностью

$$I(\lambda) = K_a(\lambda)I_a(\lambda) + K_d I_d(\lambda)C(\lambda) + K_s I_s(\lambda) + K_t I_t(y)$$

где  $\lambda$  - длина волны,  $C(\lambda)$  - заданный исходный цвет точки объекта,  $K_i$  - коэффициенты, учитывающие свойства конкретного объекта параметрами рассеянного света, диффузного и зеркального отражения, прозрачности.

$I_a$  - интенсивность рассеянного света,

$I_d$  - интенсивность диффузного отражения,

$I_s$  – интенсивность зеркального отражения,

$I_t$  - интенсивность излучения, проходящего по преломленному лучу.

Интенсивность рассеянного света ( $I_a$ ) для некоторого объекта обычно константа. Запишем формулы для остальных интенсивностей. Для диффузного отражения:

$$I_d = \sum_i I_i(\lambda) \cos \Theta_i$$

где  $I(\lambda)$ — интенсивность излучения  $i$ -го источника,  $\Theta_i$ — угол между нормалью к поверхности объекта и направлением на  $i$ -й источник света.

Для зеркального отражения:

$$I_s = \sum_i I_i(\lambda) \cos^p \alpha_i,$$

где  $p$ — показатель степени от единицы до нескольким сотен (согласно модели Фонга),  $\alpha$ ,— угол между отражённым лучом (обратной трассировкой) и направлением на  $i$ -й источник света.

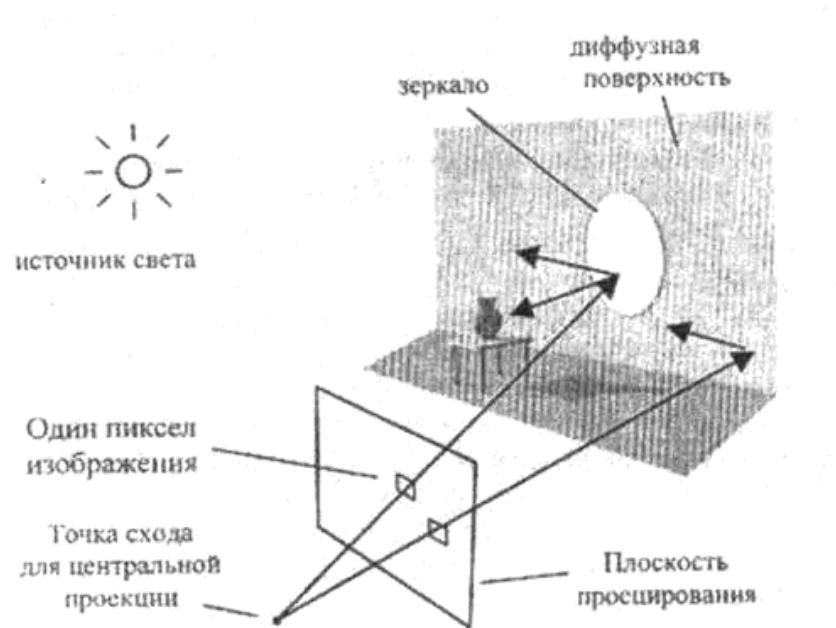


Рис. 14.2

### **Достоинства обратной трассировки лучей:**

1. Универсальность метода, его применимость для синтеза изображений достаточно сложных пространственных схем. Воплощает многие законы геометрической оптики. Просто реализуются разнообразные проекции.
2. Даже усечённые варианты этого метода позволяют получить достаточно реалистичные изображения. Например, если ограничиться только первичными лучами (из точки проецирования), то это даёт удаление невидимых точек. Трассировка уже одного-двух вторичных лучей даёт тени, зеркальность, прозрачность.
3. Все преобразования координат (если таковые есть) линейны, поэтому достаточно просто работать с текстурами.
4. Для одного пикселя растрового изображения можно трассировать несколько близко расположенных лучей, а потом усреднить их цвет для устранения эффекта ступенчатости.
5. Поскольку расчёт отдельной точки изображения выполняется независимо от других точек, то это может быть эффективно использовано при реализации данного метода в параллельных вычислительных системах, в которых лучи могут трассироваться одновременно.

### **Недостатки:**

1. Проблемы с моделированием диффузного отражения и преломления.

2. Для каждой точки изображения необходимо выполнять много вычислительных операций. Трассировка лучей относится к числу самых медленных алгоритмов синтеза изображений.

### Дискретная трассировка лучей в октантных деревьях

Применяется для воксельных моделей. Рис. 2. иллюстрирует основной принцип метода: 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).

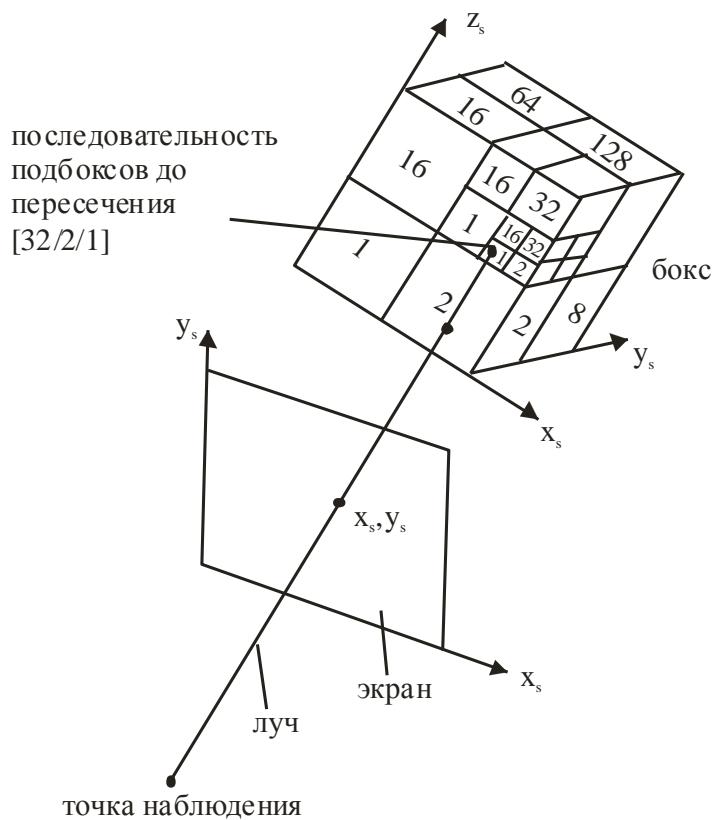


Рис. 14.3. Основная идея метода

В основном, алгоритм работает как многие методы обратной трассировки лучей (backward raytracing). 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 или преломленные лучи, если таковые имеются. Главный вопрос трассировки лучей: "Как найти точку пересечения луча и объекта?". Алгоритм решает эту задачу следующим образом:

- находит точку пересечения луча и бокса, содержащего объект (вычис-

ляются точки входа и выхода луча из бокса);

- используя информацию о точках пересечения, алгоритм отслеживает луч внутри бокса и находит первый пересеченный вексель поверхности по ходу луча;
- используя информацию о свойствах поверхности, содержащуюся в векселе (цвет, нормаль и т.д.), вычисляет цвет пикселя или испускает вторичные лучи.

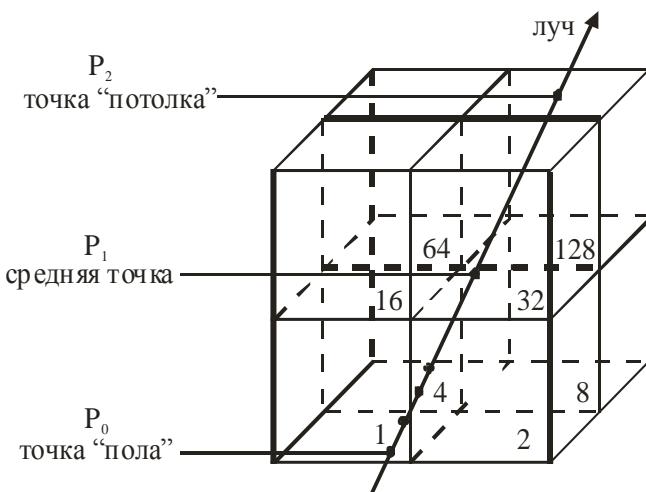


Рис. 14.4

На рис. 14.4 проиллюстрирован метод поиска точки пересечения.

Луч входит в бокс в точке "пола"  $P_0$ , проходит через *среднюю точку*  $P_1$  и покидает бокс в точке "потолка"  $P_2$ . Важно, чтобы три точки располагались на трех параллельных плоскостях, две из которых — плоскости, содержащие противоположные грани бокса, а третья — делит бокс на две равных половины. {Совершенно очевидно, что точки  $P_0$  и  $P_2$  не обязательно будут принадлежать граням бокса, в то время как средняя точка  $P_1$  обязательно будет находиться внутри бокса.

Всего может 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.

Тогда *осью главного направления* луча (ОГНЛ) называется ось, составляющая которой в векторе направления луча преобладает над другими. То есть:

$$ОГНЛ = \begin{cases} X, & \text{если } dx = \max(dx, dy, dz) \\ Y, & \text{если } dy = \max(dx, dy, dz) \\ Z, & \text{если } dz = \max(dx, dy, dz) \end{cases}$$

Методика выбора главного направления совпадает с использованной в целочисленном алгоритме Брезенхема [12] 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) выполняется непосредственно во время отслеживания луча.

Несмотря на то, что алгоритм трассировки луча очень прост и использует всего несколько целочисленных и логических операций, он требует, чтобы точки "*пола*" и "*потолка*" были вычислены для каждого пикселя изображения для того, чтобы инициализировать процесс трассировки.

Для сокращения вычислительной стоимости использован алгоритм интерполяции точки "*пола*" и "*потолка*" для всех пикселей 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 4 операции целочисленного сложения и 4 операции битового сдвига (для поддержания точности без использования плавающей точки).

В сравнении 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 воксели;
- 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 4 раза (в противоположность однородному представлению воксельного пространства, где увеличение разрешения в два раза приводит к восьмикратному росту объема информации).

• Более высокая скорость: типичные медицинские объекты, полученные методом компьютерной томографии (разрешение воксельного пространства —  $256^3$ , разрешение изображения —  $320^2$ ;) визуализируются за 0.3-0.9 секунд/кадр на ПК с процессором Pentium на тактовой частоте 200 МГц (разброс в длительности интервалов обусловлен различием в углах наблюдения моделей). Такая скорость визуализации достигнута благодаря следующим свойствам алгоритма:

- главным образом, вычисления проводятся с использованием целочисленных операций типа сдвига, сложения, логического "И" и т.д.;
- наиболее длительные операции были ускорены с использованием look-up таблиц или интерполяции конечными разностями;
- использование иерархических структур данных позволяет сократить количество операций при поиске точки пересечения луча и объекта;
- использована когерентность цветов соседних пикселей, что позволяет значительно сократить количество лучей, которые необходимо отследить для получения конечного изображения.

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.

## 15. ИСПОЛЬЗОВАНИЕ ЦВЕТА В КОМПЬЮТЕРНОЙ ГРАФИКЕ

### 15.1. Ахроматический и хроматический цвет

Так как свет является еще и волной, то, разумеется, он имеет длину волны. Длин волн бесконечное множество, но наш глаз в состоянии регистрировать только их небольшой диапазон, известный под названием видимой части спектра.

Цвет имеет психофизиологическую и психофизическую природу. Цвет предмета зависит не только от самого предмета, но также и от источника света, освещдающего предмет и от системы человеческого видения. Некоторые предметы отражают свет (стена), другие его пропускают (стекло). Если поверхность, кото-

рая отражает только синий цвет, освещается красным светом, она будет казаться черной. Если источник зеленого света рассматривается через стекло, пропускающее только красный свет, он тоже покажется черным.

Зрительная система человека воспринимает электромагнитную энергию с длинами волн от 400 до 700 нм как видимый свет.

Источник или объект являются ахроматическим, если наблюдаемый свет содержит все видимые длины волн в примерно равных количествах. Ахроматический источник кажется белым, а свет от него — белым, черным или серым. Ахроматический свет — это то, что мы видим на экране черно-белого телевизора. 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 верхнего края видимого спектра, то свет кажется красным, если у нижнего — то синим.

Но сама по себе эл/м 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 длий волн и поглощает все остальные.

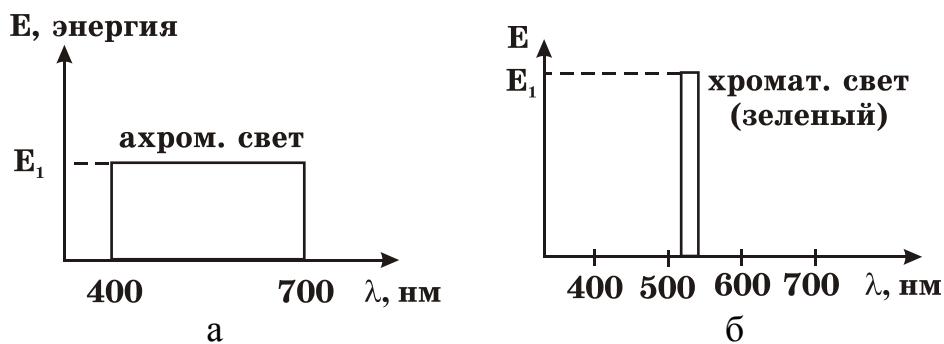


Рис. 15.1

Психофизиологическое представление света определяется:

- 1) цветовой тон
- 2) насыщенность
- 3) светлота

Цветовой тон позволяет различать цвета (к, з, с).

Насыщенность определяет степень ослабления (разбавления) данного цвета белым цветом и позволяет различать розовый цвет от красного, голубой от синего. У чистого цвета насыщенность = 100 % и уменьшается по мере добавления белого. Насыщенность ахроматического цвета = 0 %.

Светлота — это интенсивность, которая не зависит от цветового тона и насыщенности. Ноль — значит черный, более высокие значения характеризуют более яркие значения.

Психофизические определяющие цвета:

- 1) доминирующая длина волны
- 2) чистота
- 3) яркость.

Доминирующая длина волны определяет монохроматический цвет (рис. 6)  
 $\Rightarrow \lambda = 520 \text{ нм} \rightarrow \text{зеленый.}$

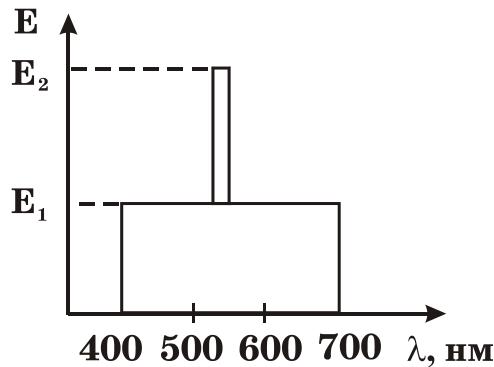


Рис. 15.2

Чистота характеризует насыщенность цвета и определяется отношением  $E_1$  и  $E_2$ .  $E_1$  — характеризует степень разбавления чистого цвета с  $\lambda = 520 \text{ нм}$  белым. Если  $E_1$  стремится к 0, то чистота — к 100 %, если  $E_1$  стремится к  $E_2$ , то свет — к белому и чистота — к 0.

Яркость пропорциональна энергии света и рассматривается как интенсивность на единицу площади. Для ахроматического света яркость есть интенсивность.

Художники используют другие характеристики цвета:

- 1) разбелы
- 2) оттенки
- 3) тона.

Разбелы получаются при добавлении в чистый цвет белого, оттенки — черного, тона — и черного, и белого.

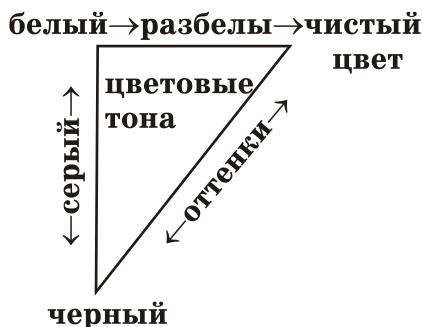


Рис. 15.3

Обычно встречаются не чистые монохроматические цвета, а их смеси. В основе 3-х компонентной теории света лежит предположение о том, что в сетчатке глаза есть 3 типа чувствительных к свету колбочек, которые воспринимают соответственно зеленый, красный и синий цвета. Относительная чувствительность глаза максимальна для зеленого цвета и минимальна для синего. Если на все 3 типа колбочек воздействует одинаковый уровень энергетической яркости (энергия в единицу  $t$ ), то свет кажется белым.

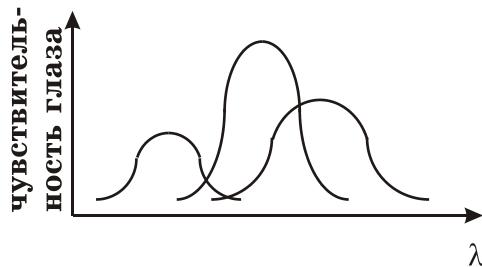


Рис. 15.4

## 15.2. Цветовые модели

RGB цвета используются в телевидении и выводе изображений на экран монитора. Эти три цвета дают возможность воспроизвести большинство цветов, которые вы можете видеть. Большинство, но не все. Цвета, производимые монитором, не являются абсолютно чистыми, поэтому и все производимые ими оттенки не могут быть воспроизведены с точностью.

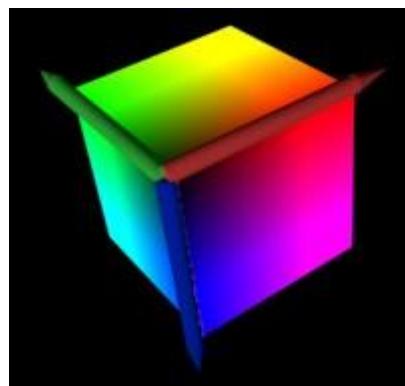


Рис. 15.5

Более того, яростный диапазон мониторов сильно ограничен. Человеческий глаз в состоянии различать гораздо больше градаций яркости. Максимальная яркость монитора едва ли соответствует и половине максимальной яркости, которую наш глаз способен различить. 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 красного, зеленого и синего изображений и их дальнейшим комбинированием.

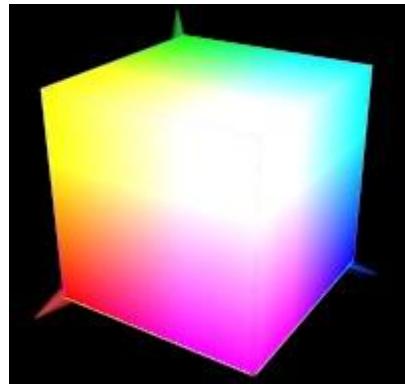


Рис. 15.6

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, Желтый - равное количество красного и зеленого и полное отсутствие синего. Все цветовые оттенки можно визуально представить в виде куба, где по осям координат будут отложены соответствующие величины трех исходных цветов. Это и есть трехцветная световая модель (RGB Model).

#### **Системы смешивания основных цветов**

1. Аддитивная — красный зеленый синий (RGB)
2. Субтрактивная — голубой (cyan, точнее сине-зеленый), пурпурный (magenta), желтый (yellow)

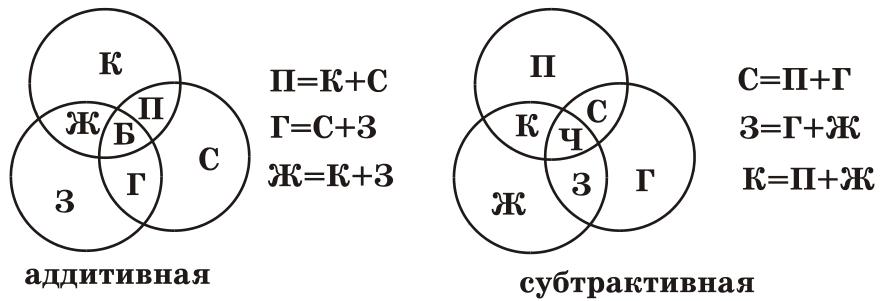


Рис. 15.7

Цвета одной системы являются дополнением к другой. Дополнительный цвет — это разность белого и данного цвета ( $\Gamma=B-K$ ,  $\Pi=B-Z$ ,  $\text{Ж}=B-C$ ).

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,

$R, G, B$  — 3 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 3-хмерны и условно их можно изобразить в виде куба;

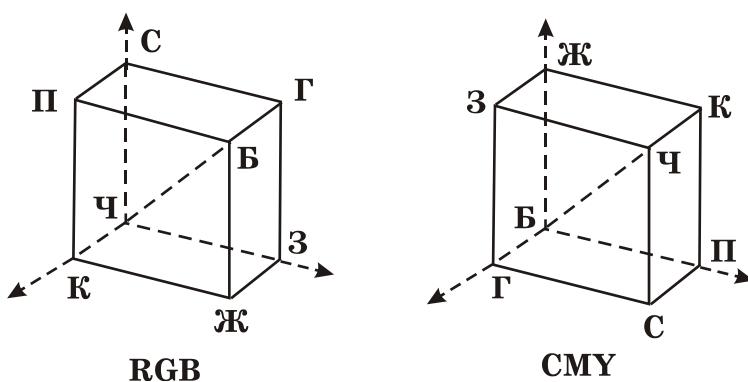


Рис. 15.8

Началом координат в цветном кубе RGB является черный цвет, а в CMY — белый. Ахроматические, т.е. серые цвета, в обеих моделях расположены по диагонали от Б до Ч.

Модели RGB и CMY аппаратно-ориентированы. Модель HVS ориентирована на пользователя. В основе лежат интуитивно принятые художниками понятия разбела, оттенка, тона.

### Цветовая модель HSV

Смит предложил построить модель субъективного восприятия в виде объемного тела HVS

(H — цветовой тон (Hue))

S — насыщенность (Saturation)

V — светлота (Value))

Если цветной куб RGB спроектировать на плоскость вдоль диагонали Б-Ч, получается шестиугольник с основными и дополнительными цветами в вершинах. Интенсивность возрастает от 0 в вершине до 1 на верхней грани. Насыщенность определяется расстоянием от оси, а тон — углом ( $0^\circ$  —  $360^\circ$ ), отсчитываемым от красного цвета. Насыщенность меняется от 0 на оси до 1 на границе шестиугольника.

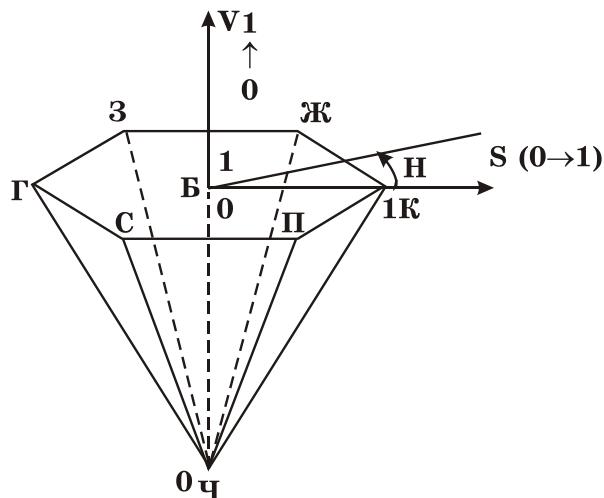


Рис. 15.9

Насыщенность зависит от цветового охвата (расстояние от оси до границы). При  $S=1$  цвета полностью насыщены. Ненулевая линейная комбинация трех основных цветов не может быть полностью насыщена. Если  $S=0$ , Н неопределен, т.е. лежит на центральной оси и является ахроматическим (серым)

- чистые цвета у художников:  $V=1, S=1$
- разбелы — цвета с увеличенным содержанием белого, т.е. с меньшим  $S$  (лежат на плоскости шестиугольника)
- оттенки — цвета с уменьшенным  $V$  (ребра от вершины)
- тон — цвета с уменьшенным  $S$  и с уменьшенным  $V$ .

### Модель HLS

В основе цветной модели HLS, применяемой фирмой Textronix, лежит цветная система Оствальда.

H — цветовой тон (Hue)

L — светлота (Lightness)

S — насыщенность (Saturation)

Модель п.с. двойной шестигранный конус. Цветной тон задается углом поворота вокруг вертикальной оси относительно красного цвета. Цвета следуют по периметру, как и в модели HVS. HLS — результат модификации HSV за счет вытягивания вверх белого цвета. Дополнение каждого цвета отстоит на  $180^\circ$  от этого

цветового тона. Насыщенность измеряется в радиальном направлении от 0 до 1. светлота измеряется вертикально по оси от 0 (Ч) до 1 (Б).

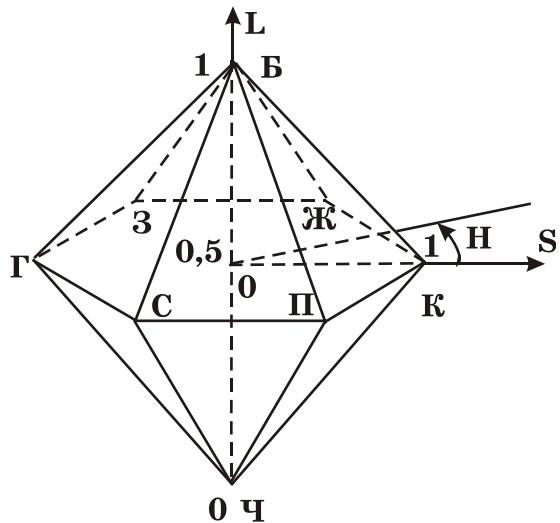


Рис. 15.10

Для ахроматических цветов  $S=0$ , а максимально насыщенные цветовые тона получаются при  $S=1$ ,  $L=0,5$ .

### Цилиндрическая цветовая модель

Используется цветовая система Манселла, основанная на наборе образцов света. Система Манселла — это стандарт восприятия. Цвет определяется:

- цветовым тоном
- насыщенностью
- светлотой

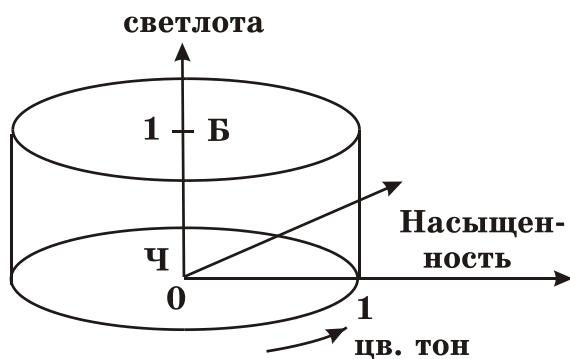


Рис. 15.11

На центральной оси — значение интенсивности меняется от черного к белому. Цветовой тон определяется углом. Главное преимущество — одинаковые приращения насыщенности, тона и интенсивности вызывают ощущения одинаковых изменений при восприятии.

## **15.3. Цветовая гармония**

Цветные дисплеи и устройства получения твердых копий позволяют создавать широкий диапазон цветов. Одни цветовые сочетания хорошо гармонируют друг с другом, другие — взаимно несовместимы. Как отбирать цвета, чтобы они гармонировали друг с другом?

— Выбор цветов обычно определяется путем проведения гладкой траектории в цветовом пространстве и/или путем ограничения диапазона используемых цветов в цветовой модели плоскостями (или шестиугольными конусами) постоянной насыщенности

- Использование цветов одного и того же цветового тона
- Использование двух дополнительных цветов и их смесей
- Использование цветов постоянной светлоты

При выборе цветов случайным образом, они будут выглядеть слишком яркими. Смит провел эксперимент, где сетка  $16 \times 16$  заполнялась цветами случайным образом и имела мало привлекательный вид.

— Если рисунок включает несколько цветов, то в качестве фона надо использовать дополнение к одному из них. Если цветов много, то фон лучше сделать серым.

— Если 2 примыкающих друг к другу цвета не гармонизируют, 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).

## **16. СЖАТИЕ ИЗОБРАЖЕНИЙ**

### **16.1. Основные сведения**

Стоит начать считывать цветные или полутоновые изображения сканером в  $\frac{1}{2}$  формата А4 и 100 Мб-ый диск будет заполнен меньше чем за 1 час (размер графического файла от 400 Кб до нескольких Мб). А сравнимый по качеству с телепередачей компьютерный фильм требует хранения данных объемом около 22 Мб/сек. Поэтому остро всталась проблема сжатия и восстановления информации. Но сжатие файла сильно зависит от его структуры.

Принципиально сжатие делают на архивацию и компрессию. Первое - без потери качества, второе - с потерями. Разница между этими способами в том, что второй не подразумевает полного восстановления исходного сохраненного изображения в полном качестве. Но каким бы не был алгоритм компрессии данных, для работы с ним файл нужно проанализировать и распаковать, т. е. вернуть данные в исходный незапакованный вид для их быстрой обработки (обычно это происходит прозрачно для пользователя).

Архивация, или сжатие графических данных, возможно 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 могут быть пиксели одного цвета, повторяющиеся текстовые данные, или некая избыточная информация, которая в рамках данного массива данных повторяется несколько раз. Например, растровый файл, состоящий из подложки строго одного цвета (например, серого), имеет в своей структуре очень много повторяющихся фрагментов.

Компрессия (конвертирование) данных - это способ сохранения данных таким образом, при использовании которого не гарантируется (хотя иногда возможно) полное восстановление исходных графических данных. При таком способе хранения данных обычно графическая информация немного "портится" по сравнению с оригинальной, но этимиискажениями можно управлять, и при их небольшом значении ими вполне можно пренебречь. Обычно файлы, сохраненные с использованием этого способа хранения, занимают значительно меньше дискового пространства, чем файлы, сохраненные с использованием простой архивации (сжатия). Суть методов сжатия с потерей качества - ликвидировать те места, которые человеческим глазом не воспринимаются или воспринимаются не очень хорошо, другими словами, практически не заметны. Чем выше степень компрессии, тем больше ущерб качеству. Оптимальное решение выбирается для конкретного случая с учетом применения.

Иногда не стоит прибегать к компрессии: проще уменьшить избыточный размер, цветность или разрешение. Результат тот же - уменьшение размера.

## 16.2. Алгоритмы сжатия файлов без потерь

Файлы состоят из символов и, возможно, «невидимых» кодов управления печатью. Каждый символ в кодах ASCII представляется 1 байтом.

### 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 они могут строиться на основе статистического анализа конкретного файла. Гарантируется возможность декодирования, хотя кодовые последовательности имеют разную длину. Сжатие до 50%.

### Алгоритм Лемпеля—Зива. LZW (Lemple-Zif-Welch)

Алгоритм сжатия данных, основанный на поиске и замене в исходном файле одинаковых последовательностей данных для их исключения и уменьшения размера "архива". В отличие от предыдущих рассмотренных методов сжатия, бо-

лее "интеллектуально" просматривает сжимаемое содержимое, а все ради большей степени сжатия данных.

Основан на сведении к минимальной избыточности. Вместо кодирования каждого символа кодируются часто встречающие 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 ≈ 50 %.

При применении этого алгоритма к загрузочным файлам (\*.exe, \*.com), результат — 10-20%, так как избыточность кода, создаваемого компиляторами меньше избыточности текста на естественном языке.

Файлы баз данных — тоже трудный случай, так как может содержать редко повторяющуюся информацию (имена, номера телефонов, адрес).

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 - монохромных, черно-белых или полноцветных. Наилучшие результаты получаются при компрессии изображений с большими областями одинакового цвета или изображений с повторяющимися одинаковыми структурами. Этот метод демонстрирует самые положительные результаты степени сжатия (среди других существующих методов сжатия графических данных) при полном отсутствии потерь или искажений в исходных файлах. Используется в файлах формата TIFF, PDF, GIF, PostScript (в инкапсулированных объектах) и других.

## **ZIP**

Метод сжатия, аналогичный использующемуся в популярном алгоритме архивации PKZip. В основу ZIP положен метод, аналогичный LZW. Как и LZW, не вносит искажений в исходный файл и лучше всего подходит для обработки графических данных с одинаковыми одноцветными или повторяющимися областями. Используется в файлах формата PDF, TIFF и некоторых других.

## **Алгоритм RLE (Run Length Encoding) «сжатие последовательности одинаковых символов»**

Изображение рассматривается как последовательность байтов. Однаковые байты кодируются парой, 1-ый байт (count) — счетчик одинаковых байтов, 2-ой — байтом из кодируемой последовательности. Для отличия счетчика 2 его старших бита устанавливается равным 1. Это позволяет кодировать последовательно-

сти длиной не более 63. Байты изображения со значениями больше 191 кодируются 2 байтами.

RLE дает хороший результат для графических адаптеров с 1 или 2 битом на  $\bar{p}$  (CGA), либо при раздельном хранении цветовых плоскостей (EGA, VGA — 16-ти цветовые режимы). Степень сжатия зависит от графического адаптера и самого изображения («гладкие» картинки сжимаются сильнее). Для CGA и EGA коэффициент сжатия больше 2. RLE работает плохо для режима 8 бит/  $\bar{p}$ .

Достоинства: простота, скорость декодирования.

Декодирование — для очередного байта В осуществляется проверка, установлены ли старшие биты не равными 0, если да, то байт является счетчиком, число повторений равно count (сбрасывается старшие биты). Следующий байт переписывается в видео память в Count экземплярах. Если старшие биты байта не установлены, то он переписывается в видеопамять без изменения.

### **CCITT Group 3, CCITT Group 4**

Эти два похожие метода сжатия графических данных работают с однобитными изображениями, сохраненными в цветовой модели Bitmap. 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 пикселей 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 - в ограничениях при компрессии графических данных (не все данные удается эффективно сжать таким образом). Этот метод - для файлов формата PDF, PostScript (в инкапсулированных объектах) и других.

### **Обрезание хвостов**

При форматировании жесткий диск разбивается на области (кластеры). Каждый кластер содержит определенное число секторов по 512 байт. В зависимости от НЖМД кластер содержит 7, 8 или 16 секторов (2, 4 или 8 Кб). DOS очень неэкономично управляет ресурсами. Она всегда выделяет для файла целое число кластеров. Если надо сохранить файл размером 1 байт, DOS выделяет 1 кластер размером 8 Кб, причем оставшееся место использовано не будет. По теории вероятности на каждый файл теряется в среднем  $\frac{1}{2}$  кластера. (Чтобы проверить потерянное место на диске, надо воспользоваться программой FileSize пакета Norton Utilities).

Программы сжатия данных помогают занять практически все свободное пространство диска, упаковывая и размещая больше данных на неиспользованных участках.

### **16.3. Сжатие цветных и полутоновых файлов. Сжатие с потерями.**

Они содержат большое количество информации ( $24 \text{ бит}/\bar{p}$  (цв) и  $8 \text{ бит}/\bar{p}$  (ч-б)) и могут весить до нескольких Мб памяти (25 Мб при сканировании цветного изображения (10 т/мм)). Характер информации постоянно меняется вдоль линии сканирования.

Алгоритмы основаны на особенностях цветовой чувствительности человеческого глаза. Глядя на картинку, человек выделяет крупные цветовые пятна, переходы между ними. Но человек может проигнорировать:

- мелкие детали,
- изменения оттенков,
- абсолютная яркость.

Например, абсолютная яркость: никакая точка телевизора не может быть чернее, чем серый цвет выключенного телевизора. Видимый нами 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 изменение цвета и интенсивности часто незначительны (небо). Сжатие по методу RLE позволяет уменьшить размеры файлов в 2-3 раза. Но это не решает полностью проблему, нужны более высокие степени сжатия.

#### **Сжатие изображения по стандарту JPEG**

(Joint Photografic Experts Group — объединенная группа экспертов по обработке фотографий). Обеспечивает уменьшение размера файла в 25-100 раз. Разработан Международной организацией по стандартизации (ISO). Такое сжатие достигается за счет сжатия с потерями. 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. Некоторую информацию можно исключить, и большинство людей этого не заметят. Кроме того, пользователь может контролировать уровень потерь, указывая степень сжатия. Что важнее — качество изображения или экономия памяти?

1 этап. Кодирование изображения по алгоритму JPEG начинается с преобразования 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-компоненты — это цветные 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 этап (необязательный). Прореживание. Отбрасываются  $U$  и  $V$  — компоненты строк или столбцов  $\bar{p}$  с определенными номерами. Например, при прореживании с коэффициентами 2:1:1 будет отбрасываться информация о цвете для каждой 2-ой строки и каждого 2-го столбца, в результате чего будет потеряно 75% данных цветности. Коэффициента 1:1:1 — прореживания нет. На  $Y$ -компоненту прореживание не отражается.

3 этап — «Дискретное косинусное преобразование» — удивительная математическая операция.

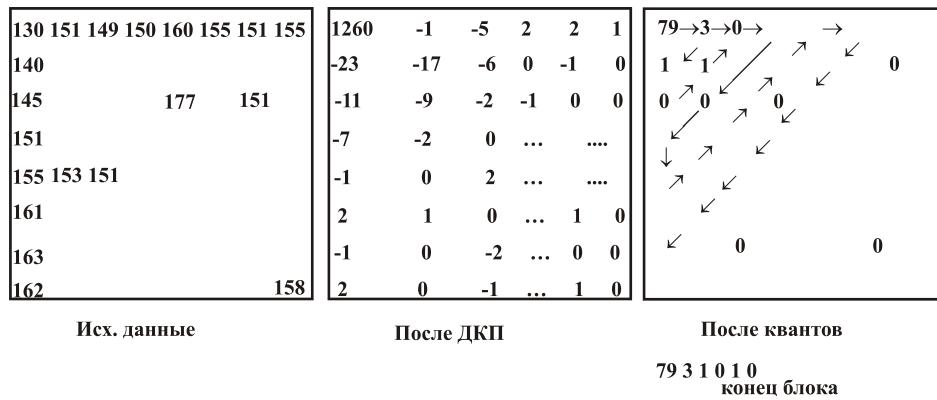


Рис. 16.1

ДКП выполняется отдельно для  $Y$ ,  $U$  и  $V$ -компонент. Изображение разбивается на блоки размером  $8 \times 8 \bar{p}$ . Такой участок с большой вероятностью содержит  $\bar{p}$  близкого цвета. При ДКП информация о  $64 \bar{p}$  преобразовывается в матрицу из 64 коэффициентов, которые характеризуют «энергию» исходных  $\bar{p}$ . 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», представляемой остальными коэффициентами, быстро убывает. Таким образом большая часть информации исходной матрицы  $8 \times 8 \bar{p}$  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 этап. Квантование. Применяется для сокращения разрядности коэффициентов. Здесь происходит значительная потеря информации. Отбрасываются малые изменения коэффициентов. Поэтому после восстановления изображения получаются уже другие значения  $\bar{p}$ . (Для  $U$  и  $V$  компонентов квантование более грубое).

5 этап. — 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  $\bar{p}$  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 объем выходного файла в 3-5 раз меньше исходного незапакованного. При качестве похуже - меньше исходника в десятки раз, но, как правило, при этом качество изображения уже не позволяет использовать его где-либо. 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 (для включенных объектов), собственно в JPEG и других.

6 этап. Сглаживание в процессе восстановления изображения. Из-за потери информации на границах между блоками ( $8 \times 8 \bar{p}$ ) могут возникать разрывы. Поэтому необходимо сглаживание.

### **Новый стандарт JPEG 2000**

В его разработке приняли участие Международная организация по стандартизации (International Organization for Standardization), Международный союз телекоммуникаций (International Telecommunications Union), компании Agfa, Canon, Fujifilm, Hewlett-Packard, Kodak, LuraTech, Motorola, Ricoh, Sony и другие. Позволяет сжимать изображения в 200 раз без заметной для невооруженного глаза потери качества. Основным отличием JPEG2000 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. Умеет также 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 JPEG2000 быстро найдет свое место в Сети, поскольку обеспечит возможность загружать картинку с разными значениями разрешения в зависимости от пропускной способности конкретного канала связи. Да и тот факт, что пользователи Интернета смогут получать изображения высокого качества, немаловажен. Еще одно значимое преимущество JPEG2000 - 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, выражющиеся в искажении цветового тона или появлении характерной кубической структуры в контрастных участках изображения (всплытие так называемых артефактов), возникают каждый раз при сохранении изображения и "накладываются" друг на друга при многократном сохранении файла в этом формате. Поэтому специалисты рекомендуют использовать форматы с частичной потерей качества только для хранения окончательных результатов работы, а не для промежуточных рабочих файлов.

Кроме того, вспомогательным средством уменьшения объема файлов в JPEG 2000 можно считать изменение цветовой модели графического файла, изменение разрешения растрового файла и ресемплирование (изменение глубины цвета пикселей).

Изменение цветовой модели графического файла можно объяснить на примере. Файлы в цветовом пространстве CMYK больше аналогичных в пространстве RGB на 33% (так как в CMYK имеется дополнительный четвертый черный канал). Если не планируешь печатать файлы или ты уверен в том, что сможешь корректно провести цветоделение (переход в субтрактивную модель CMYK) позже, можешь хранить рабочие файлы в RGB.

Изменение разрешения растрового файла используется для уменьшения избыточности файла. Файл с разрешением 600 точек на дюйм больше своего аналога разрешением в 300 точек в четыре раза (!), а качество печати при повышенном разрешении не всегда будет выше. Так что если разрешение избыточно, можешь его понизить, только надо помнить, что после такого назад дороги не будет (процесс является необратимым) и никакая последующая интерполяция не восстановит потерянные пиксели. Так что при задании необходимого разрешения необходимо быть внимательным. Следует учесть, что параметр разрешения контуров, применительно к векторной графике, не имеет отношения к объему выходного файла (это уже другое разрешение и другое понятие), но влияет на аккуратность "прорисовки" вектора при его растеризации в устройстве, где производится печать. Так что уменьшение этого параметра для векторной графики не уменьшит объем файлов, а только ухудшит качество печати.

Ресемплирование, или изменение глубины цвета растрового изображения, - изменение начальной глубины цвета файла. Некоторые оцифровывающие устройства выдают растровую информацию с глубиной цвета, превышающей достаточное для печати значение 8 бит на канал. Это иногда оправданно, так как большее значение бит на канал позволяет задавать большее число градаций цвета, что важно, например, при сильной, "кардиальной" цветокоррекции - сильном освещении или затенении отдельных участков. Однако в большинстве случаев для хранения растровых данных в различных цветовых моделях с лихвой хватит глубины цвета 8 бит на канал. Кроме того, один из стандартов сжатия для RGB-изображений подразумевает использование разного количества бит для разных цветовых составляющих (обычно наибольшее количество бит используется для зеленого канала). Также большинство фильтров Adobe Photoshop рассчитано на работу с изображениями с глубиной цвета в 8 бит (с изображением, использую-

щим нестандартную глубину цвета, становится практически невозможно работать, так как большинство фильтров рассчитаны на значение глубины цвета именно в 8 бит).

## **Фрактальное сжатие изображений**

Революция в обработке изображений реального мира произошла с выходом книги Мандельброта «Фрактальная геометрия природы». Фрактал — это структура, обладающая схожими формами разных размеров. Так, в раздавленной пачке чипсов можно найти вперемешку большие, маленькие и микроскопические кусочки. Эти структуры могут имитировать с помощью рекурсивных функций. Фракталы не зависят от разрешения устройства. Это масштабированные картинки, которые можно описать небольшим конечным набором инструкций, с помощью компьютерной программы.

### Сжатие изображения.

— Разделение изображения на неперекрывающиеся области (домены). Набор доменов должен перекрывать все изображение полностью.

— Задание набора ранговых областей изображения. Ранговые области могут перекрываться. Они не должны обязательно закрывать всю поверхность картинки.

— Фрактальное преобразование. Для каждого домена подбираем такую ранговую область, которая после аффинного преобразования наиболее точно аппроксимирует домен. Подобное преобразование не только сжимает и деформирует изображение ранговой области, но и изменяет яркость и контраст.

— Сжатие и сохранение параметров аффинного преобразования. Файл со сжатым изображением содержит две части: заголовок, содержащий информацию о расположении доменов и ранговых областей, и эффективно упакованную таблицу аффинных коэффициентов для каждого домена.

## **Восстановление изображения**

— Создание двух изображений одинакового размера, А и Б. Размер изображений может быть не равен размеру исходного изображения. Начальный рисунок областей А и Б не имеет значения. Это могут быть случайные данные, белое или черное.

— Преобразование данных из области А в область Б. Для этого сначала делится изображение Б 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.

— Преобразование данных из области Б в область А. Этот шаг идентичен предыдущему, только изображения Б и А поменялись местами, т.е. теперь

разделяется на блоки область А и отображаются на эти блоки ранговые области изображения Б.

— 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, разработанные компанией Iterated Systems, используют целочисленную арифметику и специальные методы уменьшения накапливающихся ошибок округления. Проведенные компанией исследования позволили также сократить время нахождения и обработки аффинных коэффициентов изображения. Процедуры сжатия и восстановления данных были оптимизированы по скорости работы.

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.

Фрактальное восстановление изображений можно быстро выполнить программным способом. Было показано, что персональный 33-МГц компьютер на основе микропроцессора 386 и с адаптером VGA может восстанавливать сжатые данные и демонстрировать цветной видеофильм со скоростью 20 кадров в секунду без специального аппаратного обеспечения.

### **Преимущества метода фрактального сжатия изображений**

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.

Преимущества фрактального метода распространяются и за пределы обычной технологии сжатия данных. Этот метод использует внутреннюю взаимосвязь элементов изображения. Изображение легко масштабируется, причем изменение размера репродукции получается за счет пропорционального изменения ранговых областей восстановленной картины по отношению к областям исходного изображения. Кроме того, фрактальное масштабирование может выявить невидимые на исходном изображении детали. Малый размер сжатого файла говорит о малом количестве информации в исходном изображении. С учетом этого факта и независимости изображения от разрешающей способности можно сделать вывод о фрактальности восстановленного изображения. Успешное воплощение идеи фрактального масштабирования подтверждает фрактальный характер геометрии природы.

## Аффинное преобразование

Аффинное преобразование — это основная операция фрактального метода сжатия изображений. В общем виде функция аффинного преобразования записывается следующим образом:

$$f(x, y) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}.$$

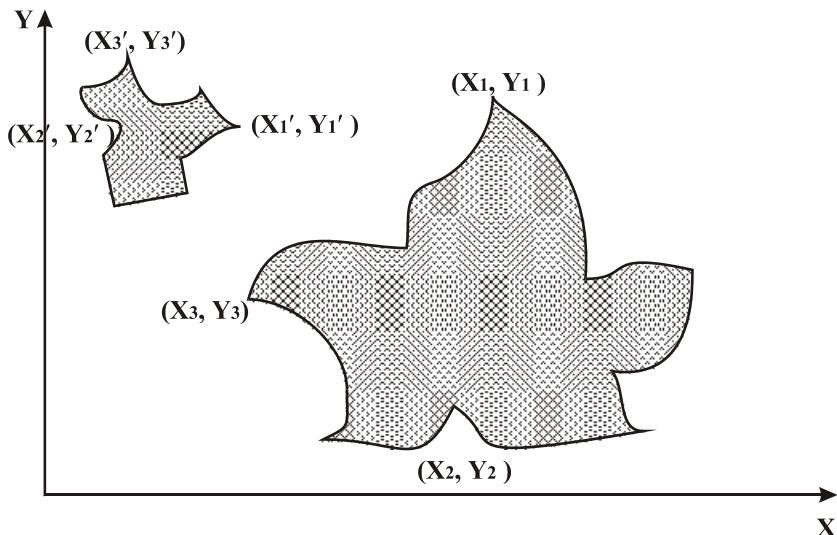


Рис. 16.2

Преобразовав в скалярную форму, получим,

$$\begin{aligned} f(x) &= ax + by + e \\ f(y) &= cx + dy + f, \end{aligned}$$

где  $a, b, c$  и  $d$  — аффинные коэффициенты вращения, деформации, расширения/сжатия,  $e$  и  $f$  — коэффициенты перемещения.

Путем простого изменения аффинных коэффициентов любой заданный набором координат  $x$  и  $y$  объект можно вращать, искажать, расширять/сжимать и (или) перемещать в двумерном пространстве.

Справедливо и обратное. С помощью аффинных коэффициентов можно восстанавливать объекты. Большой лист плюща, заданный координатами многих точек, в том числе  $x_1, y_1, x_2, y_2, x_3, y_3$ , преобразуется в малый лист с координатами  $x'_1, y'_1, x'_2, y'_2, x'_3, y'_3$ . Эти шесть координат связаны между собой аффинными соотношениями:

$$f(x1) = x1' = ax1 + by1 + e$$

$$f(x2) = x2' = ax2 + by2 + e$$

$$f(x3) = x3' = ax3 + by3 + e$$

$$f(y1) = y1' = cx1 + dy1 + f$$

$$f(y2) = y2' = cx2 + dy2 + f$$

$$f(y3) = y3' = cx3 + dy3 + f$$

Решая эти системы уравнений, можно найти аффинные коэффициенты  $a, b, c, d, e$  и  $f$ . Если применить полученные коэффициенты к любому описывающему лист набору координат, образуется другой лист.

## **Список литературы**

1. Наумович В.М. Искусственная сушка торфа. – М.: Недра, 1964. – 222с.
2. Наумович В.М. Сушка торфа и сушильные установки брикетных заводов. – М.: Недра, 1971. – 279с.
3. Булынко М.Г., Иванов В.Н., Сарматов М.И., Брикетирование торфа. – М.: Недра, 1968. – 312с.

