Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Программа развития национального исследовательского университета информационных и оптических технологий «ИТМО»

А.В. Ефремов

АЛГОРИТМЫ КОМПЬЮТЕРНОЙ ВИЗУАЛИЗАЦИИ И АНИМАЦИИ

Учебно-методическое пособие

Санкт-Петербург
2009

А. В. Ефремов
СПб: СПбГУ ИТМО, 2009, – 132 с.
Алгоритмы компьютерной визуализации и анимации

В учебно-методическом пособии рассмотрены вопросы построения компьютерного изображения по математическому описанию, методы и принципы работы основных алгоритмов, а также способы реализации приведенных алгоритмов при помощи системы визуализации DirectX.

Пособие предназначено для магистров, обучающихся по программе магистерской подготовки «Компьютерная видеоинформатика», а также студентов других оптико-информационных специальностей.

Одобрено на заседании Совета факультета Фотоники и оптоинформатики Санкт-Петербургского государственного университета информационных технологий, механики и оптики «___» _____ 2009 г. (протокол № ____).

Оглавление

1. Методы визуализации

1.1. Методы и этапы визуализации

Все основные задачи компьютерной графики так или иначе связаны с обработкой, преобразованием и построением изображений. Задачей компьютерной визуализации, в частности, является построение изображения по некоторому описанию.

Описание для подобной задачи может быть составлено с помощью некоторых примитивов или последовательности действий. В общем случае можно считать, что описание представляет собой структурированную группу данных на некотором языке, выраженное в текстовой форме.

Таким образом, задача визуализации может быть сведена к преобразованию описания на некотором специфичном языке в изображение.

Прежде чем приступить к дальнейшему рассмотрению способов реализации подобного преобразования, необходимо определить формат представления изображений.

Существует две основных группы представления изображений — **векторные** и **растровые**. Векторные изображения задаются в виде совокупности базовых примитивов, таких как точки, линии, многоугольники, сплайны и т.п.

Векторные изображения обладают рядом важных достоинств:

- Характерно малый объем данных для описания, не зависящий от линейного размера представляемого изображения
- Возможность осуществления преобразований (таких как мастштабирование, растяжение, перенос, поворот и т.п.) без малейшей потери качества.

Возможность масштабирования делает векторные изображения очень удобными, например. при полиграфических работах (на устройствах разного разного разрешения и точности). Существует ряд устройств, как например, принтеры, поддерживающее формат PostScript, работающие непосредственно с векторными форматами представления данных.

Однако у векторного формата есть и ряд существенных недостатков:

- Невозможность представить в векторном виде некоторые типы изображений (например, фотографии).
- Необратимость процесса растеризации (то есть преобразование из векторного формата в растровый является сравнительно легкой процедурой, когда как из растрового в векторный — очень неточный и трудоемкий процесс, называемый трассированием).

В противовес векторным изображениям рассматриваются изображения растровые, то есть состоящие из «растров». Растр представляет собой одну точку изображения, характеризуемую некоторым цветом. Растровое изображение представляет собой двумерную матрицу из растров.

Растровая матрица имеет две главные характеристики — ширину (width) и высоту (height). Таким образом, растровая матрица состоит из width*height растров. Эти параметры также называются «размером» или «разрещением» изображения.

Для именования растра также часто используется термин «пиксель» (pixel) — производное от PICture Element, т.е. элемент изображения. Именно этот термин будет использоваться далее для обозначения базовой единицы изображения.

Природа растровых изображений берет начало в технических особенностях средств визуализации. Так, например, в основе визуализации при помощи электронно-лучевых трубок (Cathode-Ray Tubes, CRT) лежит принцип попадания луча на матрицу из люминофоров. Жидкокристаллические мониторы (Liquid Crystal Displays, LCD) используют принцип возбуждения жидких кристаллов, также организованных в прямоугольную матрицу. Подобный список может быть продолжен до бесконечности.

Таким образом, любое изображение при визуализации на большинстве современных устройств имеет некоторый шаг дискретизации, обусловленный техническими особенностями устройства. И представление изображений в виде двумерной матрицы точек-пикселей является прямым следствием из этих особенностей.

К достоинствам растровых изображений можно отнести следующие особенности:

- возможность цветовой трансформации изображений (т.е. некоторое «поточечное» преобразование пикселей);
- возможность представления сколь угодно сложной информации в виде конечной последовательности данных предсказуемого размера.

Однако растровые изображения обладают и рядом недостатков. К таковым можно причислить:

- невозможность точного масштабирования и некоторых других пространственных преобразований без потерь данных;
- большой объем хранимых данных даже для простых изображений (частично нивелируется «сжатыми» форматами).

Первый недостаток является одним из самых весомых при рассмотрении используемого представления для изображения. Растровые изображения, используемые, например, для типографской печати, должны иметь очень высокое разрешение. При стандартном разрешении 300 DPI (Dots Per Inch, точек на дюйм) изображение формата A5 должно иметь размеры 2480×1750 пикселей, что является довольно крупным размером для растрового изображения даже на сегодняшний день.

Различия, возникающие при масштабировании изображения, представленного в векторном и растровом формате, представлены на рисунке 1.

Рисунок 1.1. Различия векторного и растрового изображений

Как видно из рисунка 1, невозможность масштабирования является самым серьезным недостатком растровой графики. Также «разрушающими» для растровой графики являются все преобразования, которые не могут быть описаны простым переводом «из пикселя в пиксель». Такими преобразованиями являются, например, поворот на отличный от прямого угол, растяжение, сжатие и т.п.

Настоящий курс лекций посвящен визуализации в первую очередь на экране компьютера. Тем самым основная задача визуализации сводится к построению именно растрового изображения по его описанию.

Способы построения и применения векторных изображений являются отдельной темой, выходящей за рамки настоящего курса.

1.2. Цветовые пространства

Как было показано выше, растровое изображение представляет собой двумерную матрицу, состоящую из отдельных точек — пикселей. Основной характеристикой каждого пикселя (помимо координаты его расположения в матрице) является его цвет. Очень важно понимать, каким образом цвет представляется на экране компьютера, как он задается и хранится в компьютерном представлении.

В настоящей главе мы рассмотрим основные цветовые модели, применяемые для описания цвета и способы их задания и представления.

С точки зрения физики цвет является свойством световой волны (при рассмотрении света как электромагнитной волны с точки зрения волновой теории). В общем случае световая волна, как любая другая, описывается амплитудой, длиной волны λ, фазой и поляризацией. Последние две характеристики мы рассматривать не будем как не имеющие отношения к вопросу.

Цвет световой волны определяется ее длиной λ. Как известно, видимый диапазон света расположен между длинами волн 400 и 700 нм соответственно.

Распределение длин волн изображено на рисунке 1.2.

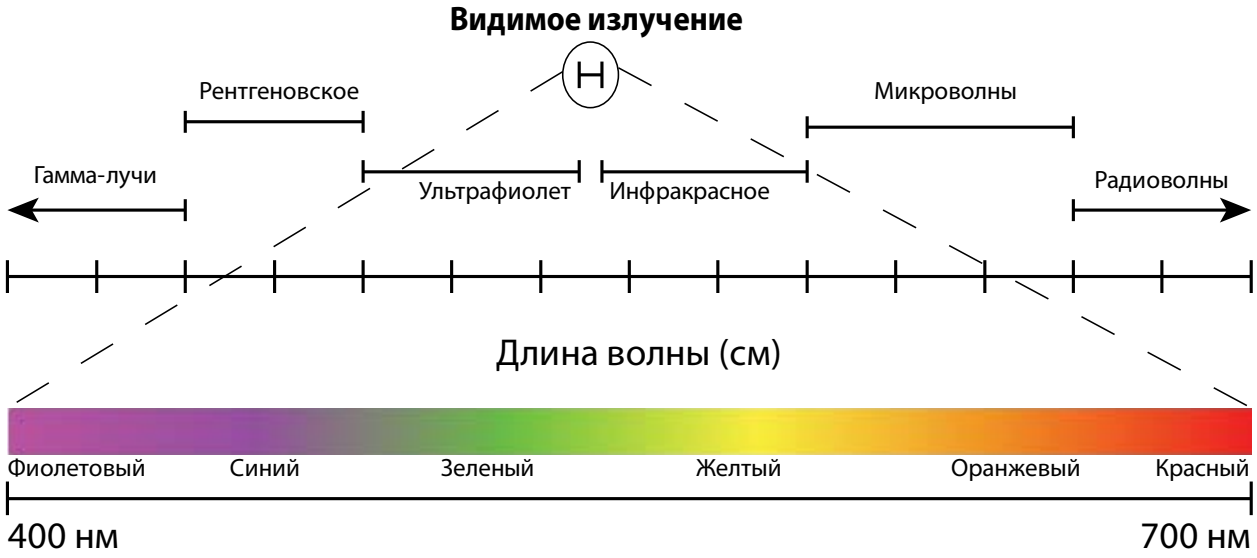


Рисунок 1.2. Диапазон видимых световых волн.

В реальной жизни видимый свет редко бывает представлен только одной длиной волны — такое случается только в луче лазера. Луч света в реальной жизни обычно состоит из множества волн различной интенсивности и длины. Окраска такого пучка означает, что интенсивность волн определенных длин преобладает над интенсивностью волн с остальными длинами.

1.2.1. Восприятие цвета человеком

Вообще говоря, понятие цвета связано с восприятием света человеческим глазом. Свет, попадающий на сетчатку, возбуждает фоторецепторы, которые в свою очередь, передают мозгу сигнал об интенсивности воспринятого пучка световых волн.

Человеческий глаз содержит четыре вида рецепторов — один с широкой спектральной чувствительностью («палочки») и три с узкой («колбочки»). Широкая спектральная чувствительность «палочек» обуславливает невосприимчивость их к длине волны. Иначе говоря, «палочки» не регистрируют цвет. Однако они обладают высокой чувствительностью к амплитуде волн и позволяют глазу различать предметы при очень слабом освещении. «Палочки» рассредоточены по периферии сетчатки, что делает боковое зрение более светочувствительным, но менее восприимчивым к цвету видимых объектов.

В центральной части сетчатки, в области так называемого «острого» зрения, расположены фоторецепторы другого рода, называемые «колбочками». Колбочки имеют более высокий порог чувствительности к амплитуде волны (то есть менее чувствительны к свету), однако имеют узкий спектр регистрации длины световых волн, что позволяет мозгу использовать их сигнал для определения цвета светового пучка.

Человеческий глаз имеет три вида «колбочек» — красные, желто-зеленые и синие. Глаза собак, например, имеют только два типа цветовых

фоторецепторов. Существует также уникальный вид морских моллюсков, глаза которых обладают

График спектральной чувствительности «колбочек» изображен на рисунке 1.3.

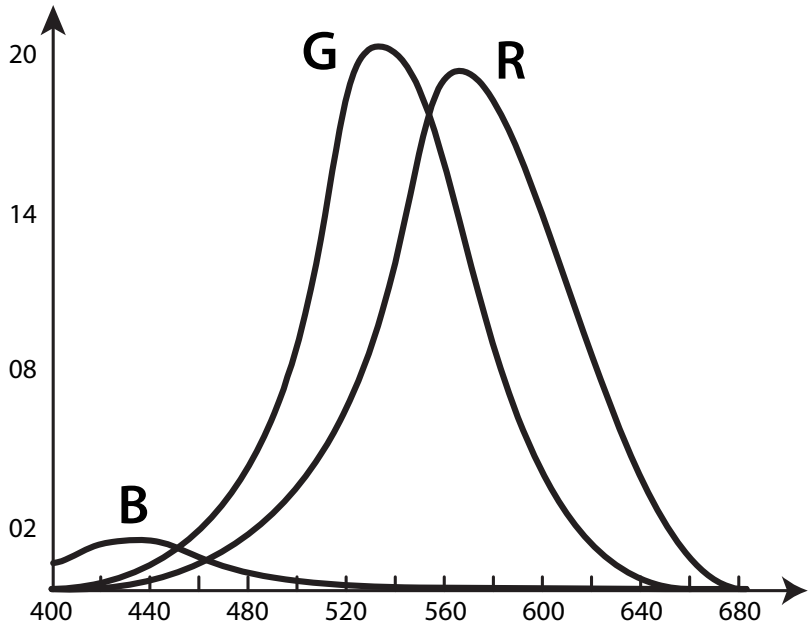


Рисунок 1.3. Спектральная чувствительность сетчатки

Из рисунка видно, что глаз обладает наибольшей чувствительностью в желто-зеленом диапазоне, а наименьшей — в синем. Также можно отметить, что близость пиков кривых красного и зеленого цвета обуславливает частоту именно такого отклонения зрения, как красно-зеленая слепота (дальтонизм). Достаточно небольшого сдвига в кривых, и разница сигналов между красными и желто-зелеными рецепторами практически нивелируется.

Если задать функциями P_R, P_G и P_Bсоответствующие кривые чувствительности колбочек, а за $I(\lambda)$ обозначить интенсивность световых волн с длиной λ, получаемую глазом цветовую информацию можно описать в следующем виде:

$$R = \int I(\lambda)P_R(\lambda)d\lambda$$
$$G = \int I(\lambda)P_G(\lambda)d\lambda \quad (1)$$
$$B = \int I(\lambda)P_B(\lambda)d\lambda$$

Отметим, что одни и те же коэффициенты R, G и B могут быть получены при помощи разных распределений $I(\lambda)$. Такие распределения, называются метамерами.

На рисунке 4 представлены значения коэффициентов для волн различной длины из видимой части спектра.

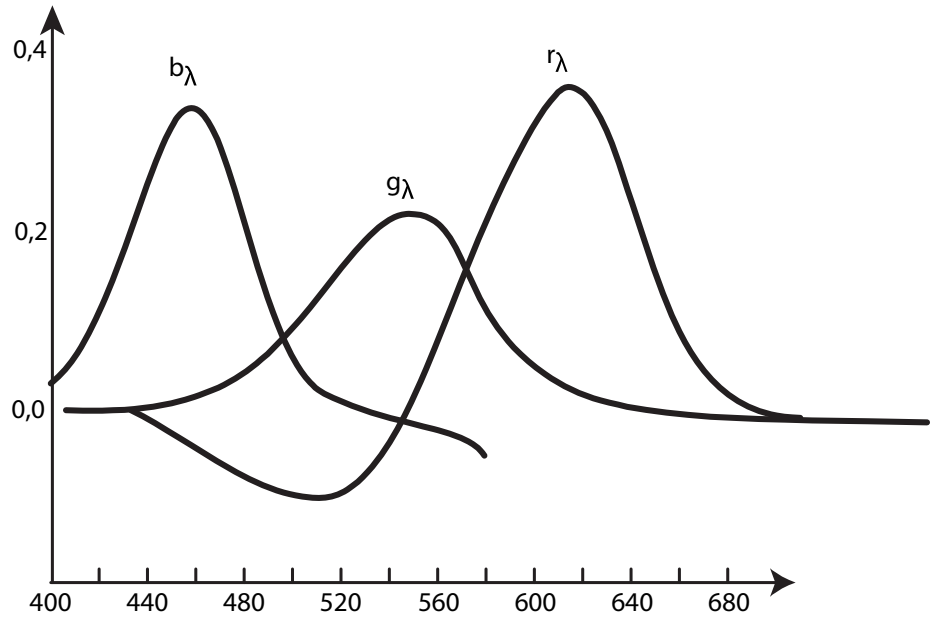


Рисунок 1.4. Коэффициенты для видимой части спектра

Характерная особенность графиков — некоторые коэффициенты (в частности, синий цвет) могут быть отрицательными. В частности, это означает, что при помощи RGB-модели не все цвета могут быть адекватно описаны. Так, например, с помощью этой модели невозможно предать цвет спелого апельсина.

Отсюда можно заключить, что цветные мониторы, в большинстве своем использующие RGB-модель для представления цвета (то есть использующие люминофоры трех различных цветов), не могут передать всех воспринимаемых глазом оттенков.

1.2.2. Модель XYZ

Приведенный выше факт обуславливает необходимость использования другой цветовой модели, которая бы позволила описать видимые цвета при помощи неотрицательных коэффициентов.

Такая модель была разработана комиссией CIE (Commission Internationale de L'Eclairage) в 1931 году. Комиссией были приняты так называемые «стандартные кривые для гипотетического наблюдателя», то есть как если был на наш мир смотрел сферический наблюдатель в вакууме. Цветовые кривые, предложенные комиссией CIE, изображены на рисунке 5.

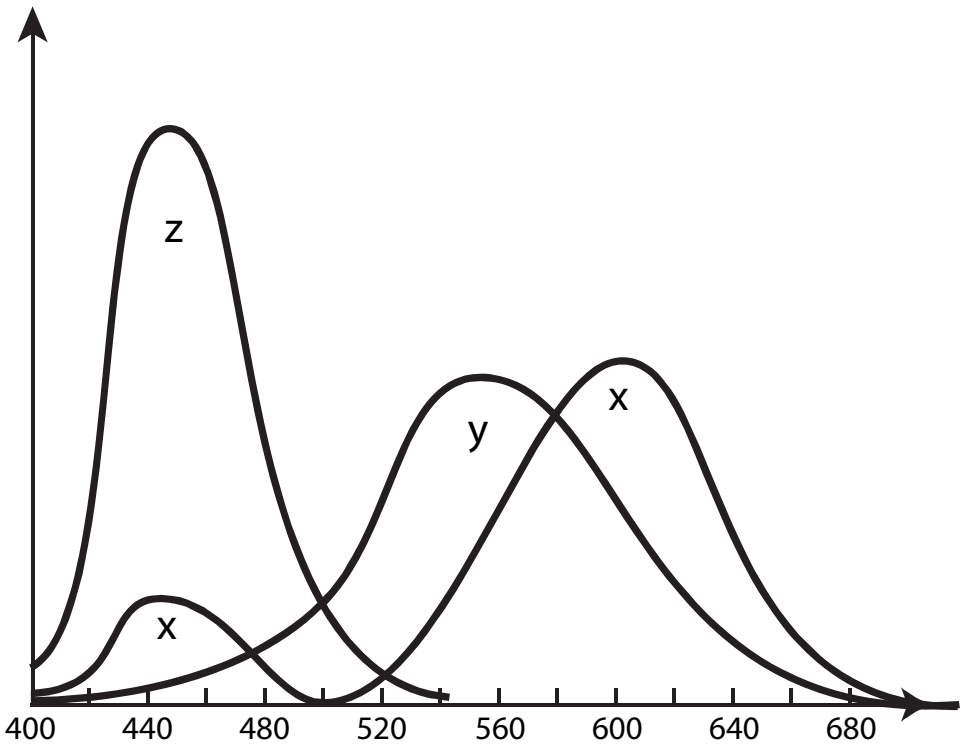


Рисунок 1.5. Спектральная чувствительность CIE XYZ

Как видно из рисунка, комиссией были введены три кривых $\bar{x}$, $\bar{y}$ и $\bar{z}$, и им соответствуют три цветовых координаты, определяемых по формулам:

$$X = \int I(\lambda)\bar{x}(\lambda)d\lambda$$
$$Y = \int I(\lambda)\bar{y}(\lambda)d\lambda \quad (2)$$
$$Z = \int I(\lambda)\bar{z}(\lambda)d\lambda$$

Кривые выбраны так, что задаваемый коэффициентами цвет (X, Y, Z) определяется однозначно. Также из рисунка 5 можно отметить, что кривая $\bar{y}$ совпадает с кривой чувствительности глаза к свету. Соответственно, величина Y в координатах (X, Y, Z) отвечает за воспринимаемую яркость и называется люминантностью (CIE luminance).

Чтобы отделить информацию о яркости от информации о цветовой составляющей, зачастую вводятся так называемые «хроматические» координаты x и y:

$$x = \frac{X}{X + Y + Z}$$
$$y = \frac{Y}{X + Y + Z} \quad (3)$$

Задаваемый таким образом цвет (x, y, Y) позволяет однозначно восстановить тройку (X, Y, Z), и при этом координаты (x, y) будут описывать на плоскости картину, изображенную на рисунке 1.6:

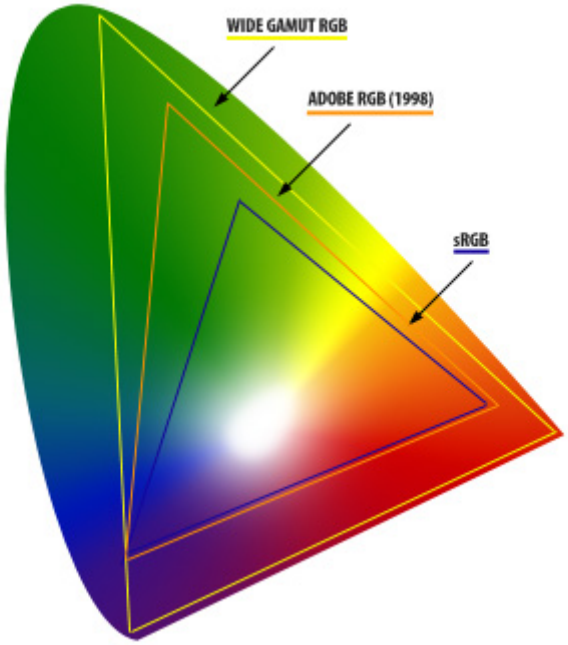


Рисунок 1.6. Хроматическая диаграмма

Окаймляющая часть кривой построена точками (x, y) при движении длины волны λ от 400 до 700нм. Соединяющий их отрезок, проходящий по сиреневым цветам, не является спектральным и содержит в себе смешение волн красного и синего диапазонов.

Также неспектральными цветами являются черный, получаемый отсутствием света вообще, и белый, получаемый смесью всех цветов. CIE задает белый цвет при помощи так называемой кривой D_{65}, определяющей белый цвет как приближение дневного света. В системе XYZ координат былого цвета принято обозначать как (X_n, Y_n, Z_n). На рисунке 6 белому цвету соответствует точки, расположенная примерно в центре области.

Если на хроматической диаграмме отметить две точки, соответствующие двум разным цветам, то их смешению будут соответствовать все точки, расположенные на отрезке прямой между этими цветами на диаграмме. Соответственно, для трех точек цветам, получаемым при их смешении, будут соответствовать точки из образуемого ими треугольника.

Из формы цветовой области на диаграмме видно, что покрыть полностью весь цветовой диапазон некоторым треугольником не удастся — всегда буду оставаться некоторые неиспользуемые коэффициенты, или же непокрытые области.

На рисунке 6 отображены несколько стандартных моделей RGB. Как видно, ни одна из них полностью не покрывает весь видимый спектр.

Восприятие светового пучка глазом носит нелинейный характер — он ближе к экспоненциальному. Так, например, если взять два источника света, один из которых имеет интенсивность 18% от интенсивности второго, то первый источник будет восприниматься всего вдвое менее ярким. Именно

этим обусловлено приведение усредненной яркости картинки к 18%-му серому в фотоаппаратах при автоматическом замере экспозиции.

Модель XYZ, как следствие, также является нелинейной. Для получения линейного освещения были введены модель CIE L*u*v*

$$L^* = \begin{cases} 116\left(\frac{Y}{Y_n}\right)^{1/3} - 16, \frac{Y}{Y_n} > 0.008856 \\ 903.3\frac{Y}{Y_n}, \frac{Y}{Y_n} < 0.008856 \end{cases} \quad (4)$$

$$u'_n = \frac{4X_n}{X_n + 15Y_n + 3Z_n} \qquad v'_n = \frac{4Y_n}{X_n + 15Y_n + 3Z_n}$$
$$u' = \frac{4X}{X + 15Y + 3Z} \qquad v' = \frac{4Y}{X + 15Y + 3Z}$$
$$u^* = 13L^*(u' - u'_n) \qquad v^* = 13L^*(v' - v'_n)$$

И модель CIE L*a*b*

$$a^* = 500\left[\left(\frac{X}{X_n}\right)^{1/3} - \left(\frac{Y}{Y_n}\right)^{1/3}\right]$$
$$b^* = 500\left[\left(\frac{Y}{Y_n}\right)^{1/3} - \left(\frac{Z}{Z_n}\right)^{1/3}\right] \quad (5)$$

В обеих моделях величина L* считается изменяющейся от 0 до 100, причем изменение на 1 считается порогом чувствительности глаза.

Также CIE определяет следующие понятия:

Тон (hue) — атрибут визуального восприятия, согласно которому цвет кажется одним из воспринимаемых (красным, желтым, зеленым, синим) или комбинацией любых двух из них.

Насыщенность (saturation) — пропорция «чистого» и белого цветов, необходимая для определения цвета. Насыщенность показывает, насколько цвет отличается от белого. Так, например, красный или синий цвета имеют насыщенность 100%, а оттенки серого (в том числе белый и черный) — 0%.

1.2.3. Цветовоспроизводящая аппаратура

Интенсивность света, генерируемого некоторым устройством (гапример, трубкой лучевого монитора) обычно зависит от силы сигнала. Для ЭЛТ, например, зависимость интенсивности от сигнала нелинейна и может быть выражена формулой

$$Intensity \sim Voltage^{\gamma} \quad (6)$$

Коэффициент γ обычно лежит в пределах 1.8-2.5. Для большинства мониторов $\gamma = 2.2$.

В связи с подобным преобразованием сигнала получаемое изображение необходимо подвергать так называемой «γ-коррекции». В соответсвие с общей рекомендацией, видеокамера при записи сигнала производит преобразование линейного RGB-сигнала, полученного с сенсора, по формуле:

$$R' = \begin{cases} 4.5R, & R \leq 0.018 \\ -0.099 + 1.099R^{0.45}, & R > 0.018 \end{cases} \quad (7)$$

И аналогично для компонент G и B. Идеальный монитор инвертирует изображение по формуле

$$R = \begin{cases} R/4.5\,, & R \leq 0.08 \\ \left(\dfrac{R' + 0.099}{1.099}\right)^{\frac{1}{0.45}}, & R > 0.08 \end{cases} \quad (8)$$

На основе этих цветов вводится величина линейной яркости, называемая «люма» (luma), которая определяется по формуле

$$Y' = 0.299R' + 0.587G' + 0.114B' \quad (9)$$

1.2.4. Цветовая модель RGB

Цветовая схема RGB, применяемая в обычных мониторах и ряде других устройств ввода и вывода изображений, является наиболее простой и общеупотребимой. Такая модель является «аддитивной», то есть для получения финального цвета его компоненты складываются.

Цветовым пространством в такой модели является единичный куб, изображенный на рисунке 7.

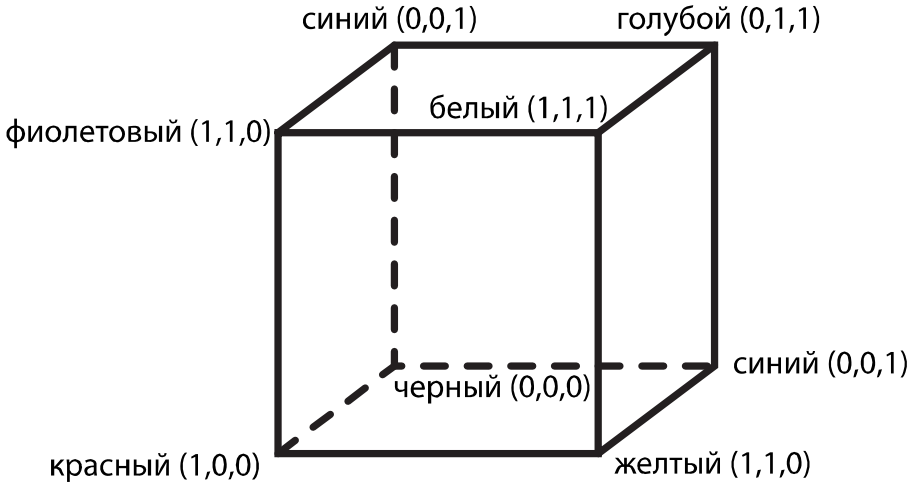


Рисунок 1.7. Цветовой куб.

На диагонали куба располагаются оттенки серого — от черного (0,0,0) до белого (1,1,1). Упомянутая выше рекомендация 709 определяет хроматические координаты люминофора монитора и белого цвета следующим образом:

	Красный	Зеленый	Синий	Белый
x	0.640	0.300	0.150	0.3127
y	0.330	0.600	0.060	0.3290

(10)

Отсюда можно получить формулы для перевода цвета из системы CIE XYZ в RGB монитора:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 3.24079 & -1.537156 & -0.498535 \\ -0.969256 & 1.875992 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (11)$$

Если какой-то цвет не может быть представлен в модели RGB, одна из координат будет отрицательной. Преобразование обратимо:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad (12)$$

1.2.5. Цветовая модель CMYK

В цветной печати, наоборот, используется субтрактивные модели CMY (Cyan, Magenta, Yellow) и CMYK (Cyan, Magenta, Yellow, blacK). Субтрактивность модели означает, что для получения результирующего цвета базовые цвета необходимо вычитать из белого.

Свет от люминофоров монитора является прямым, то есть в некотором смысле люминофоры сами являются источниками света. И их свет при попадании на сетчатку глаза складывается.

При печати же на сетчатку попадает свет, отраженный от поверхности бумаги. Нанесенная на бумагу желтая краска поглощает синий цвет и отражает только красный и зеленый. Таким образом синий цвет как бы вычитается из падающего белого. Именно по этой причине подобная модель названа субтрактивной.

Поверхность, покрытая одновременно голубым и желтым красителем, будет поглощать красный и синий цвета соответственно, отражая только зеленый свет. Подобное преобразование можно описать в следующем виде:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} C \\ M \\ Y \end{pmatrix} \tag{13}$$

Для экономии красителей и для устранения цветовых отклонений при выводе оттенков серого в трехкомпонентную систему добавляют еще и черную краску (blacK). В модели CMYK компонента черного цвета определяется как минимум из трех основных компонент, а затем вычитается из остальных.

$$\begin{gathered} K = \min(C, M, Y) \\ C' = C - K \\ M' = M - L \\ Y' = Y - K \end{gathered} \tag{14}$$

1.2.6. Другие цветовые модели

В телевидении используются цветовые схемы, основанные на яркости и хроматических компонентах. Обусловлено это тем, что изначально телевещание было черно-белым, и единственная яркостная составляющая передавалась при помощи модуляции на некоторой несущей частоте. Для сохранения совместимости при появлении цветного вещания была оставленная яркостная составляющая, а две других компоненты цвета транслировались на соседних частотах, но модулированные с меньшей амплитудой.

Существует три телевизионные цветовые модели — американская NTSC и европейские PAL и SECAM. Последняя система была разработана совместно с французскими учеными и применяется в телевещании России и стран бывшего СССР. Приведем формулы расчета цветов для вещательной системы SECAM:

$$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0.30 & 0.59 & 0.11 \\ 0.60 & -0.28 & -0.32 \\ 0.21 & -0.52 & 0.31 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{15}$$

Отметим, что в этой модели при R=G=B цветовые составляющие I и Q примут нулевые значения. Подобные YIQ и YUV системы часто применяются в цветовом кодировании для сжатия изображений и видео, но иногда с другими наборами коэффициентов.

Все описанные модели удобны для работы с цветопередающей или цветовоспроизводящей аппаратурой, однако для человека думать в подобных системах непривычно и неудобно. Для задания цветов человеком были разработаны системы типа HSV (Hue, Saturation, Value) и HSL (Hue, Saturation, Lightness), которые строятся на более понятных человеку понятиях тона, яркости и насыщенности.

Модель HSV может быть представлена конусом с шестигранным основанием, где величина V(Value) задает проекцию на ось конуса, S(Saturation) — удаленность от оси, а Hue(тон) — угол вокруг оси.

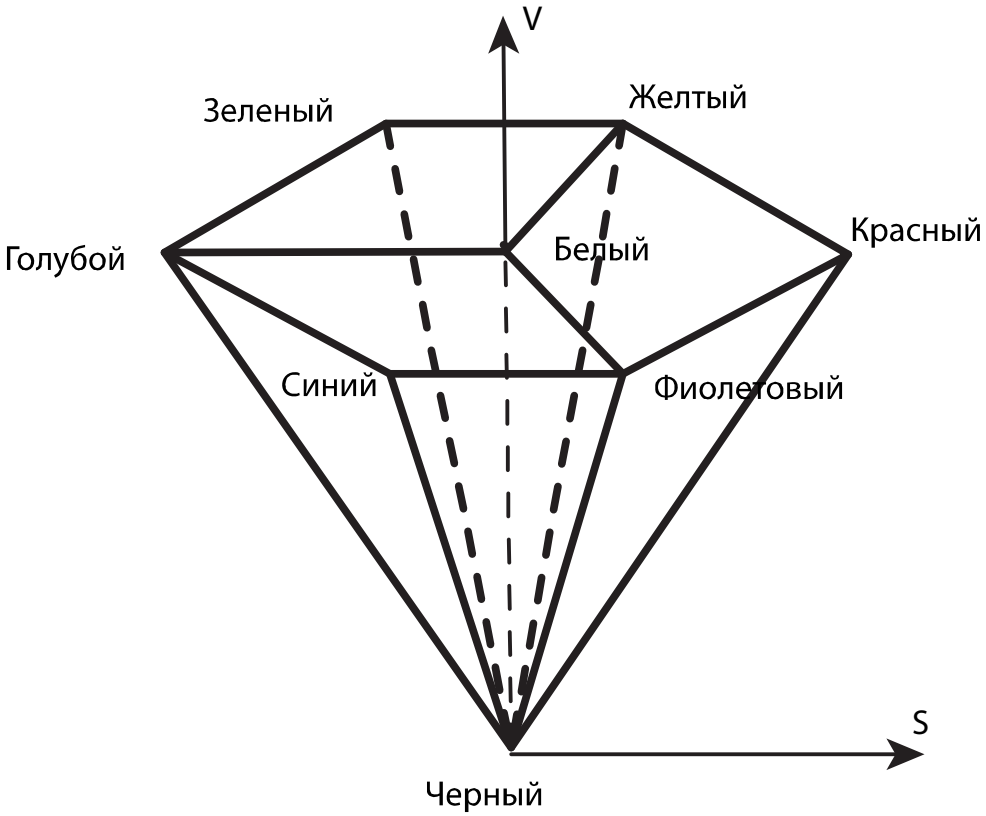


Рисунок 1.8. Цветовая модель HSV

Конус имеет единичную высоту (V=1), где нулевой яркости соответствует черный цвет, а единице — максимально яркий (белый или, например, чисто синий). Насыщенность также изменяется от 0 к 1, причем нулю соответствуют точки на вертикальной оси, а единице – на сторонах конуса.

Тон (Hue) определяется как угол отклонения от красного цвета, то есть 0^0 соответствует красному, 120^0 — зеленому, 240^0 — синему и т.д. Заметим, что при $S = 0$ параметр H не имеет смысла, а при $V = 0$ не имеют смысла оба остальных параметра.

Формулы для перевода из RGB в HSV могут быть представлены следующим образом. Пусть $Max = \max(R, G, B)$ и $Min = \min(R, G, B)$. Тогда

$$\Delta = Max - Min$$

$$V = Max$$

$$S = \begin{cases} \frac{\Delta}{V}, & V > 0 \\ 0, & V = 0 \end{cases}$$

$$H = 60^0 * \begin{cases} \frac{G - B}{\Delta}, & Max = R \\ 2 + \frac{B - R}{\Delta}, & Max = G \\ 4 + \frac{R - G}{\Delta}, & Max = B \end{cases} \quad (16)$$

Легко может быть построена обратная формула.

Другой также часто употребляемой системой является HLS. В этой системе цветовое пространство описывается уже двум шестигранными конусами, как приведено на рисунке 1.9.

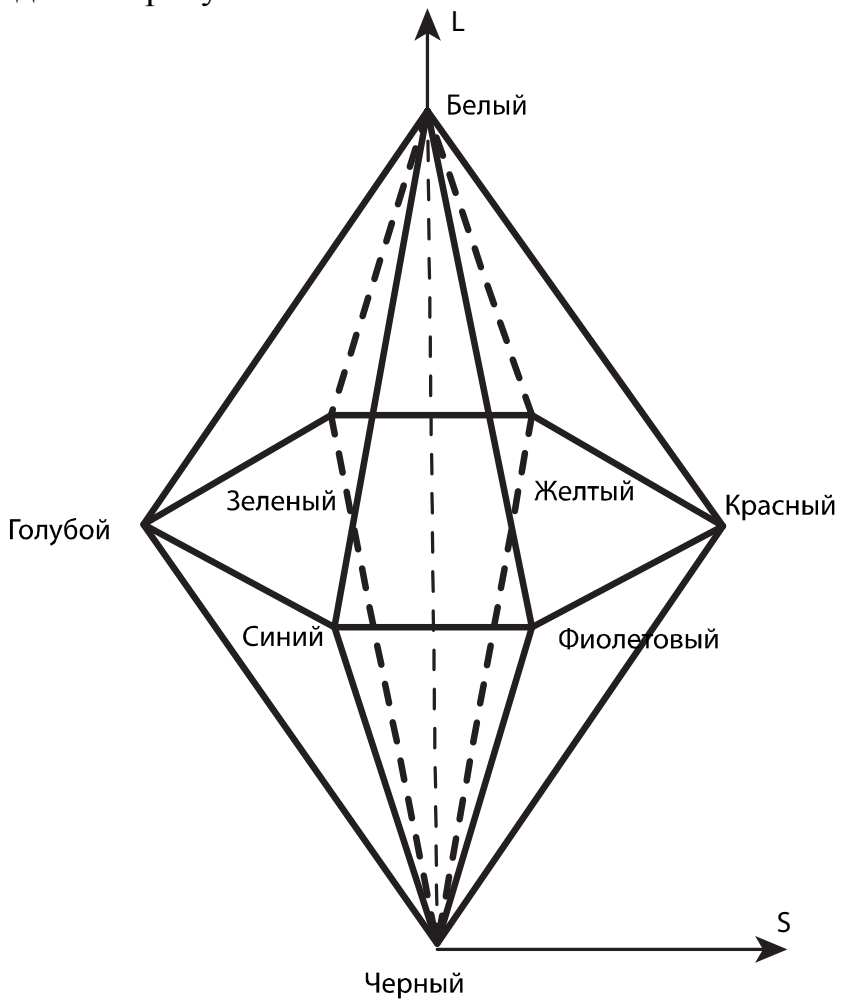


Рисунок 1.9 . Цветовая модель HLS

Такая модель во многом идентична модели HSV, но позволяет более точно описывать цвета в области средней и высокой яркости за счет введения понятии «светлости». Так, яркость синего цвета равна 1, когда как его «светлость» — всего 0.5.

В разделе были описаны самые основные цветовые модели, используемые повсеместно. Существуют и другие модели, как узкоспециализированные, предназначенные для конкретных задач, так и сходные с приведенными, и отличающиеся лишь значением коэффициентов приведения.

Настоящий курс лекции будет опираться, в основном, на аддитивную модель RGB, как основную, используемую для описания цветов на экране монитора.

1.2.7. Форматы изображений

Описанный при помощи той или иной цветовой модели цвет необходимо некоторым образом представлять и хранить в памяти компьютера.

Цвет, представленный тройками RGB, очевидно, может быть записан тремя парами чисел с плавающей точкой для каждого пикселя изображения. Минимальный размер числа с плавающей точкой по стандарту IEEE требует 4 байта для записи, то есть 12 байт на пиксель. Такое количество данных довольно велико, однако порой используется для записи данных, которые после будут подвергнуты обработке. Например, такая точность записи может быть использована при подготовке графики для последующего монтажа.

Однако в большинстве случаев такая точность не нужна — человеческий глаз с трудом воспринимает более 100 различный яркостей одного канала (базового цвета). Поэтому самым популярным способом представления цвета в компьютере является представление, отводящие один байт (8 бит) на канал, то есть 24 бита для записи всех трех каналов цвета.

Также часто используются модели с 16 битами (2 байтами) на канал — например, в компьютерных форматах PNG, TGA, TIFF. Более 8 бит на канал также применяются в так называемых «сырых» (RAW) форматах данных, применяемых для кодирования информации с матрицы фото или видеокамеры. Поскольку эти данные также подлежат последующей обработке, для их хранения используются представления с 10, 12 или 16 битами на канал.

При кодировании с 8 битами на канал (т.н. True Color) представление цвета канала числом от 0 до 1 преобразовывается в целое число о 0 до 255, где 255 — максимальное значение канала. Для удобства цвета принято записывать в шестнадцатеричном формате. Ниже приведена таблица основных цветов:

Цвет	Шестнадцатеричное представление
Белый	`0xFFFFFF`
Красный	`0xFF0000`
Синий	`0x0000FF`
Зеленый	`0x00FF00`
Черный	`0x000000`
Серый	`0x808080`

Несколько лет назад, когда устройства для хранения данных обладали не очень большой емкостью и объемы данных были важны, использовались также форматы хранения данных с меньшим количеством бит на канал. Чаще всего использовалась схема с 16 битами (2 байтами) для записи цвета. Цвета в таком формате хранились по схеме 5-5-5 (5 бит на цвет, один лишний) или 5-6-5 (пять бит на красный и синий канал, 6 на зеленый). Приоритет зеленого канала объясняется из рисунка 2, как самый «воспринимаемый» цвет. Иными словами, глаз наиболее чувствителен именно к градациям зеленого цвета.

Однако 5 бит (32 градаций) на канал все же недостаточно — плавные (фотографические) изображения имели характерные перепады цветов, и глаз легко улавливал места перехода. Для устранения подобного рода сложностей

использовался так называемый «дизеринг», то есть представление промежуточного цвета при помощи паттернов из соседних цветов. Глаз смешивал расположенные рядом цвета, получая промежуточные градации.

Также для наиболее оптимального кодирования цвета использовались «палитрованные» способы представления данных. Для каждого изображения создавалась «палитра» — набор соответствий 2-битного цвета и некоторого 8-битного (или даже менее) кода. Затем все изображение кодировалось при помощи цветов из палитры посредством указания кода цвета в этой палитре.

Для оптимизации занимаемого изображением места также использовались различные методы сжатия, однако их рассмотрение выходит за рамки настоящего курса.

1.3. Геометрические данные и преобразования

После того, как все вопросы с отображением изображений на экране компьютера и представлением этих изображений в компьютере решены, пришел черед обсудить то, как эти изображения получать.

Поскольку наш мир обладает тремя измерениями, то и основной задачей трехмерной визуализации является отображение трехмерных объектов на экране компьютера.

Не следует забывать о природе средств отображения — изображение на экране компьютера всегда двумерно. Существует ряд способов передачи трехмерных изображений — голография, сложные системы проективного отображения и т.п., но обсуждение подобных систем выходит за рамки настоящего курса.

Таким образом, основной задачей трехмерной визуализации является отображение трехмерных объектов на двумерном растровом экране компьютера. Такое отображение объектов может быть разбито на два этапа:

- Моделирование
- Визуализация (рендеринг)

Моделирование заключается в описании объектов и окружения, в которые они помещены. На этапе моделирования создается так называемая сцена — набор отображаемых объектов. Для этих объектов указывается их положение и ориентация в пространстве, размеры, анимация и т.п.

Визуализация — процесс перевода математически описанной модели, созданной на этапе моделирования, в растровое изображение. На этом этапе рассчитывается видимость объектов из определенной точки, освещенность, отбрасываемые тени и прочие визуальные атрибуты.

Для того чтобы сцена могла быть визуализирована, ее необходимо описать математически, чему и будет посвящено содержание следующего раздела.

1.3.1. Представление объектов

В общем случае объекты располагаются в трехмерном пространстве. Пространство описывается при помощи декартовых координат, то есть положение (но не ориентация!) объекта в пространстве однозначно задается тройкой координат (x,y,z), соответствующей смещению по трем осям координат декартовой системы.

Важнейшей частью моделирования является возможность геометрической трансформации объектов. К такого рода трансформациям относятся параллельный перенос, вращение вокруг оси, масштабирование, скос и т.п. Трансформации позволяют менять форму объектов, их размеры и размещать их друг относительно друга.

Рассмотрим основные виды трансформации на примере двумерных координат, а затем спроецируем полученные нами формулы на трехмерный случай.

1.3.2. Левая и правая координатные системы

Прежде чем говорить о преобразованиях в пространстве, необходимо оговорить систему координат, используемую при описании объектов.

Для задания положения объектов в пространстве используется декартова система координат. Базовая ориентация подобной системы строится из следующего положения: ось абсцисс X направлена вправо от наблюдателя, ось ординат Y вверх, а ось аппликат Z – от наблюдателя.

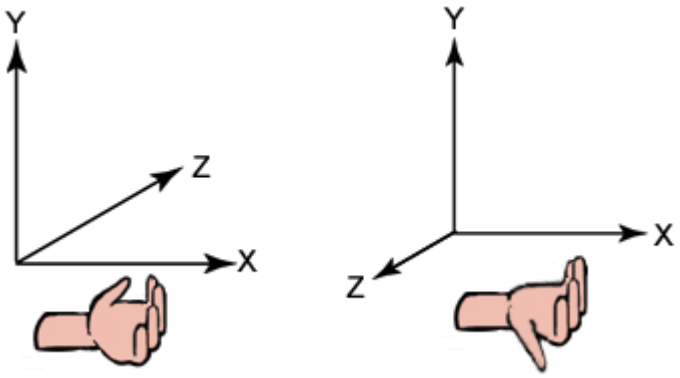


Рисунок 1.10. Левосторонние и правосторонние координаты

Подобная тройка осей образует так называемую левую (или левостороннюю) систему координат. Ориентация системы координат определяется по правилу рук — в левосторонней системе оси ориентированы по направлению пальцев левой руки (X вдоль указательного, Z вдоль большого, ось Y протыкает ладонь). Для правосторонней же системы используется правило правой руки.

В общепринятой математике для описания системы координат принято пользоваться правосторонней системой, однако зачастую в компьютерной графике используется левосторонняя. Так, среди самых общеупотребимых (и аппаратно поддерживаемых) систем визуализации трехмерной графики DirectX использует левостороннюю систему координат, а OpenGL — правостороннюю.

Сложно утверждать, с чем связано использование той или иной системы координат. Причины здесь могут быть сокрыты за пределами математики или вычислительной оптимизации и касаться, скорее, особенностей коммерческого продвижения систем и вопросами конкуренции продуктов на рынке.

Настоящий цикл лекций использует в качестве системы визуализации библиотеку DirectX, поэтому основной в качестве координатной системы будет использоваться левовосторонняя. Однако весь изложенный ниже материал может быть также применен и к правосторонним системам.

Координаты и формулы преобразования могут легко преобразовываться из одной системы в другую путем замены знака у в тех частях формулы, которые используют значение по оси Z.

1.3.3. Координатная система экрана

В качестве основной системы вывода мы рассматриваем экран компьютера. Экран представляет собой прямоугольную матрицу растров (пикселей), каждый из которых обладает собственным цветом.

Основными характеристиками экрана как растровой матрицы являются, как уже было упомянуто, размеры (разрешение), а также так называемая глубина цвета — количество бит, отводимое на запись цвета одного пикселя.

Пиксели экрана, как и вообще все растровых изображений в компьютерной графике, принято нумеровать, начиная с левого верхнего угла. Так, для экрана с разрешением $w \times h$ левый верхний пиксель будет иметь координаты $(0,0)$, а правый нижний — $(w-1, h-1)$.

Следовательно, можно говорить о том, что ось абсцисс для экранных координат направлена по горизонтали вправо, а ось ординат, соответственно, по вертикали вниз.

Рисунок 1.11. Экранные координаты

Если систему координат, в которой описываются изображаемые модели, можно считать непрерывной (то есть $\mathbb{R}^3$), то экран является пространством дискретным ($\mathbb{Z}^2$). Дискретность здесь обусловлена растровым представлением изображения на экране.

Таким образом, общую схему преобразования описания трехмерного изображения (сцены) можно представить как осуществление некоторого преобразования $T: \mathbb{R}^3 \to \mathbb{Z}^2$ и последующего определения цветовой составляющей результирующего изображения.

Далее будут описаны простые преобразования над описаниями объектов в трехмерном мире, а затем будет показано построенное преобразования, позволяющее перевести объект из трехмерных координат в экранные.

1.3.4. Параллельный перенос

Параллельный перенос или сдвиг — преобразование, перемещающее все точки объекта вдоль некоторого фиксированного направления на заданное расстояние.

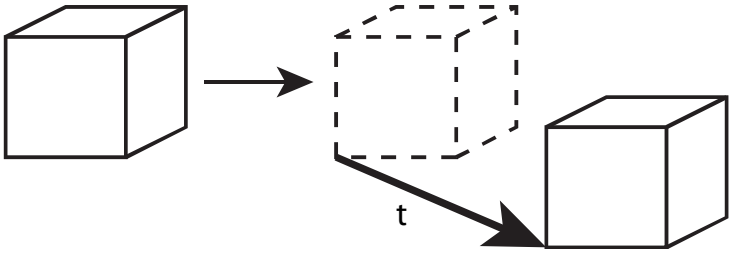


Рисунок 1.12. Параллельный перенос

Сдвиг задается только вектором t, в направлении которого и осуществляется перенос. В общем случае перенос задается следующей формулой.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} t_x \\ t_y \end{pmatrix} + \begin{pmatrix} x \\ y \end{pmatrix} \tag{17}$$

1.3.5. Поворот

Поворот — преобразование, при котором все точки объекта поворачиваются вокруг нуля координат. Поворот зависит только от угла θ, на который осуществляется вращение.

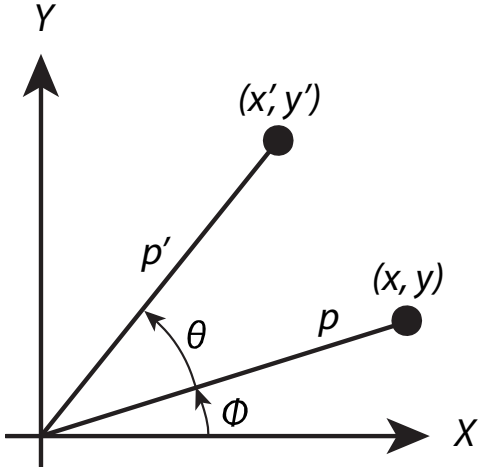


Рисунок 1.13. Расчет формулы поворота

Формулы для описания преобразования вращения могут быть получены следующим образом. Из рисунка следует, что:

$$\begin{aligned} x &= \rho \cos(\phi) \quad & x' &= \rho \cos(\phi + \theta) \\ y &= \rho \sin(\phi) \quad & y' &= \rho \sin(\phi + \theta) \end{aligned} \tag{18}$$

Откуда, воспользовавшись правилами приведения синусов и косинусов, можно получить следующее выражение

$$\begin{aligned} x' &= \rho\cos\phi\cos\theta - \rho\sin\phi\sin\theta = x\cos\theta - y\sin\theta \\ x' &= \rho\cos\phi\sin\theta - \rho\sin\phi\cos\theta = x\sin\theta + y\cos\theta \end{aligned} \tag{19}$$

То же самое можно записать и в матричной форме:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{20}$$

Следует отметить, что направление положительного вращения — против часовой стрелки. При рассмотрении того же вращения в трехмерном случае можно рассматривать двумерный случай как плоскость XY, и тогда абсолютно те же формулы будут верны для вращения в трехмерном пространстве в плоскости XY относительно оси Z.

Параллельный перенос и вращение относятся к так называемым изометрическим, или твердотельным (rigid-body) преобразованиям. Такие преобразования вместе и по отдельности не могут изменить форму объекта, а лишь меняют его позицию и ориентацию.

1.3.6. Масштабирование

Масштабирование относится к анизометрическим преобразованиям, так как изменяют форму тела.

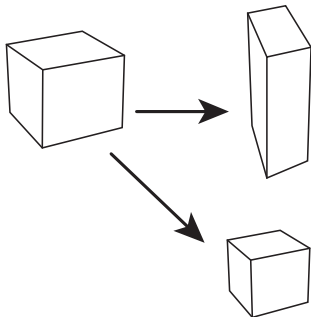


Рисунок 1.14. Преобразование масштаба

Масштабирование характеризуется точкой приложения (в простейшем случае это центр координат) и масштабом изменений по осям. Масштабирование бывает пропорциональным и непропорциональным, в зависимости от того, совпадают ли масштабы по осям. Если параметр масштабирования по оси менее 1, объект сжимается по этой оси, если больше — растягивается. Если же масштаб меньше нуля, происходит отражение (зеркалирование) объекта относительно соответствующей оси.

Масштабирование может быть задано следующими формулами:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{21}$$

1.3.7. Однородные координаты

Все описанные выше преобразования могут быть описаны при помощи следующей системы:

$$\begin{aligned} x' &= Ax + By + C \\ y' &= Dx + Ey + F \end{aligned} \tag{22}$$

Такое преобразование было бы удобно записать в матричном виде, однако наличии свободных коэффициентов C и F не позволяют использовать матричный способ записи полностью.

На помощь приходит введение так называемых однородных координат. Однородные координаты на плоскости задаются при помощи тройки чисел $(x, y, 1)$, Однородные координаты обладают тем свойством, что координата точки в пространстве, задаваемая однородной координатой, не меняется при умножении всех компонент этой тройки на константу.

Иными словами, однородные координаты задают класс эквивалентности со свойством $(x, y, 1) \approx (xw, yw, w)$. Однородные координаты, последняя компонента которых равна единице, называются нормализованными.

Преобразование, описанное в (22), называется аффинным. Однородные координаты позволяют описывать все аффинные преобразования при помощи матричного способа.

Самое простое, тождественное преобразование в терминах однородных координат будет задаваться матрицей

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{23}$$

Преобразование переноса на вектор t теперь будет задаваться матрицей

$$T = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \tag{24}$$

В самом деле, если домножить матрицу переноса T на заданный в однородных координатах вектор X, то получится

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix} \tag{25}$$

Аналогично описываются преобразования поворота на угол θ

$$R = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (26)$$

И масштабирования

$$S = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (27)$$

1.3.8. Суперпозиция преобразований

Поскольку произвольное аффинное преобразование может быть записано посредством матричной алгебры, можно говорить о суперпозиции преобразований. В самом деле, рассмотрим суперпозицию некоторых преобразований A, B и C над некоторым вектором p. Преобразование A над этим вектором даст нам некоторый вектор p_a, который впоследствии преобразуется при помощи B в вектор p_b, а затем при помощи C — в p_c. В функциональной форме подобное преобразование может быть записано в виде

$$p_c = C(B(A(p))) \quad (28)$$

Поскольку A, B и C могут быть описаны матрицами, скобки в подобном описании могут быть сняты.

Произведение матриц ассоциативно, а значит, результат не будет зависит от того, вычислим ли мы сначала действие матрицы A, а потом всех остальных, или же сначала перемножим все матрицы, а потом воздействуем полученной матрицей на вектор p.

Стоит отметить, что умножение матрицы на столбец — гораздо более быстрая операция, чем перемножение матриц, и для единственного вектора перемножение всех матриц было бы неэффективно. Однако если одна и та же группа матриц должна быть использована при преобразовании нескольких десятков, сотен или тысяч векторов, оптимизация от домножения на одну матрицы по сравнению с тремя будет уже существенной.

Таким образом, описание аффинных преобразований в матричной форме позволяет осуществлять конвейерную обработку множества вершин посредством суперпозиции всех преобразований в одно.

Использованию такой стратегии формирования матрицы сложного преобразования следует отдать предпочтение перед непосредственным вычислением элементов матрицы. Этот подход хорошо "вписывается" в конвейерную архитектуру графической системы.

1.3.9. Переход к трехмерным координатам

Случай двумерных координат легко обобщается на трехмерный. Вектор в однородных координатах в трехмерном пространстве задается четверкой $(x, y, z, 1)$. Таким образом, аффинное преобразование будет выглядеть следующим образом:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1' \end{pmatrix} = \begin{pmatrix} A & B & C & D \\ E & F & G & H \\ K & L & M & N \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{29}$$

Матрицы параллельного переноса приобретет вид:

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{30}$$

Матрица масштабирования, соответственно, будет выглядеть как:

$$S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{31}$$

Если в двумерном случае вращение могло быть задано только вокруг одной оси, то в трехмерном случае в качестве оси вращения может выступать произвольно ориентированный в пространстве вектор. Принято выделять три основных матрицы вращения, по количеству базовых осей:

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y = \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta\, 0 & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{32}$$

$$R_z = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

1.3.10. Скос

Хотя любое аффинное преобразование можно свести к последовательности поворотов, сдвигов и изменений масштаба, существует один вид преобразования, который настолько важен в компьютерной графике, что мы будем рассматривать его также как базовый тип. Это преобразование скоса (shear). Рассмотрим куб, центр которого находится в начале системы координат, а ребра параллельны осям координат.

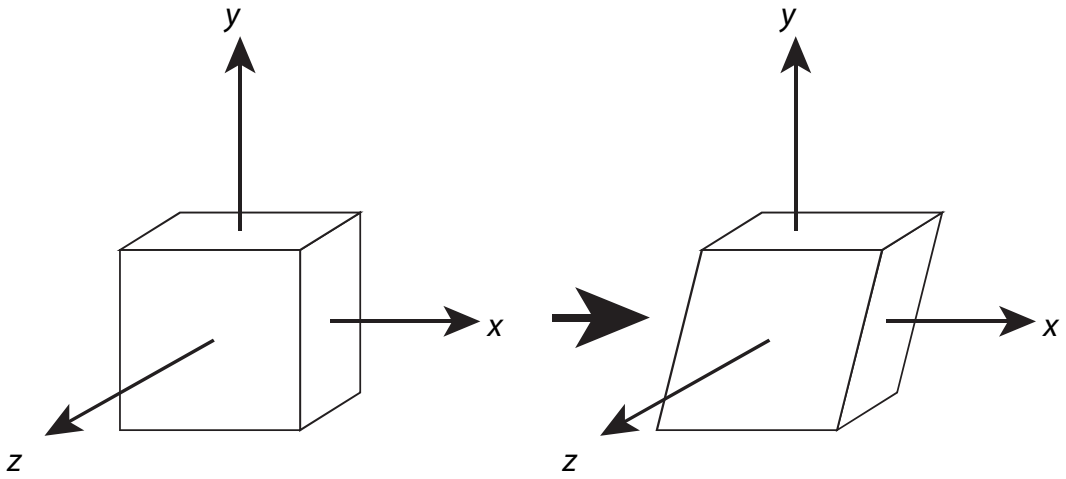


Рисунок 1.15. Преобразование скоса

Если сместить верхнюю грань куба вправо, а нижнюю — влево, объект будет скошен в направлении оси x. Следует учитывать, что при этом компоненты y и z всех точек на объекте остаются неизменными. Мы будем называть показанное на рисунке преобразование x-скосом, чтобы отличать его от скоса в другом направлении.

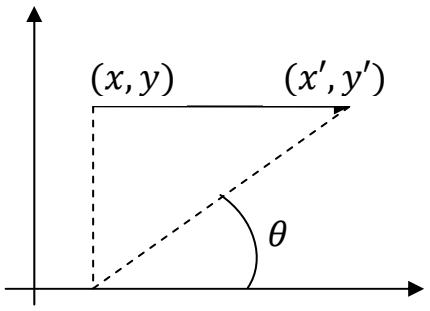


Рисунок 1.16 Расчет преобразования скоса

Воспользовавшись простыми тригонометрическими соотношениями, которые поясняются на рисунке, можно охарактеризовать преобразование такого типа единственным параметром — углом скоса θ. Уравнения преобразования имеют вид:

$$\begin{aligned} x' &= x + y\cot\theta \\ y' &= y \\ z' &= z \end{aligned} \tag{33}$$

Откуда следует общий вид матрицы скоса:

$$H_x(\theta) = \begin{pmatrix} 1 & \cot\theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{34}$$

Матрицу обратного преобразования можно получить, заметив что обратный скос — это скос в обратном направлении, то есть

$$H_x^{-1}(\theta) = H_x^T(-\theta) \tag{35}$$

1.3.11. Поворот вокруг произвольной точки

В первом примере будет показано, как формировать матрицу преобразований поворота вокруг произвольной фиксированной точки, используя канонические матрицы поворота вокруг начала координат. Направление оси поворота в примере совпадает с направлением координатной оси z, но этот же метод можно использовать и при другом направлении оси.

Рассмотрим куб, центр которого находится в точке p_f, а ребра параллельны осям координат. Требуется повернуть куб вокруг оси z, но сохранить неизменным положение центра.

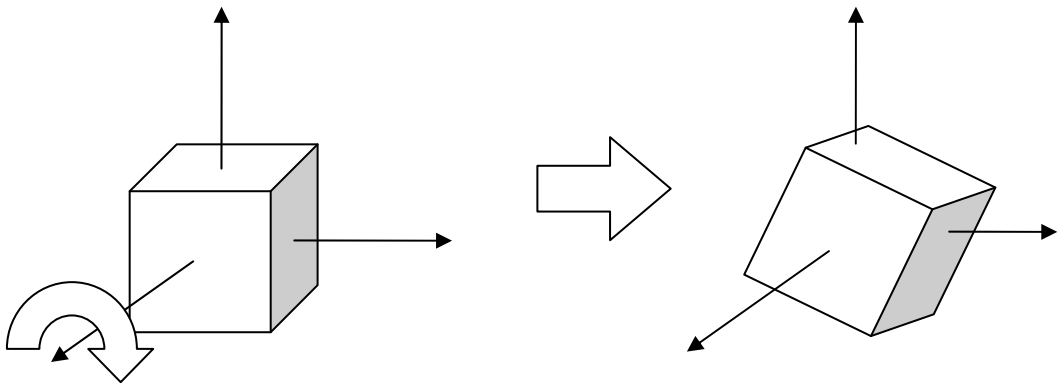


Рисунок 1.17. Вращение вокруг оси

Если бы точка p_f совпадала с началом координат, мы могли бы воспользоваться уже сформированной матрицей R_z. Отсюда следует, что сначала нужно сдвинуть куб таким образом, чтобы его центр оказался в начале координат, а затем воспользоваться матрицей R_z.

Последняя операция — сдвинуть куб так, чтобы его центр занял прежнее положение

Эта последовательность преобразований представлена на рисунке.

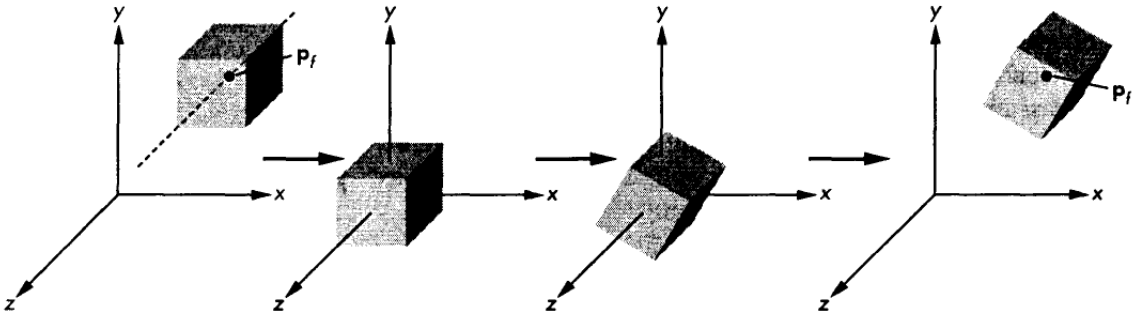


Рисунок 1.18. Вращение вокруг произвольной точки

Используя введенные раньше обозначения, нужно сначала выполнить преобразование $T(-p_f)$, далее — $R_z(\theta)$, а последним выполнить$T(p_f)$. В результате суперпозиции этих преобразований получим матрицу

$$M = T\left(-p_f\right)R_z(\theta)T(p_f) \tag{36}$$

После реального перемножения матриц получим следующий результат:

$$M = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & x_f - x_f\cos\theta + y_f\sin\theta \\ \sin\theta & \cos\theta & 0 & y_f - x_f\sin\theta - y_f\cos\theta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{37}$$

1.3.12. Представление объектов

Способ преобразования для объектов в трехмерном пространстве определен, и теперь необходимо определить способ задания самих объектов.

В общем случае объект может быть задан любым описание. Так, например, идеальная сфера задается как множество точек, равноудаленных от центра. Однако способ визуализации для произвольно описанных объектов с трудом обобщается, и сложно говорить о едином методе визуализации для такого рода объектов.

По этой причине необходимо использовать некоторый обобщенный способ задания фигур в пространстве, позволяющий трактовать и визуализировать их однотипным методом. В качестве такого способа было избрано приближение объектов и поверхностей при помощи многогранников с плоскими гранями.

В общем случае такой многогранник задается набором точек в пространстве, называемых его вершинами, а также набором граней на этих вер-

шинах. Далее, любая плоская грань может быть триангулирована, то есть быть представлена в виде набора треугольников. Таким образом, любой объект в пространстве задается в виде набора вершин и треугольных граней на этих вершинах.

Вершины таких фигур принято организовывать в списки. Такие списки называются вершинными. Каждая вершина имеет в таком списке индекс (положение по порядку), и может быть однозначно этим индексом определена.

Треугольные грани для такого списка вершин могут быть заданы тройками чисел (i, j, k), соответствующих индексам вершин в вершинном списке, на которые опирается грань.

Рассмотрим, например, куб в трехмерных координатах:

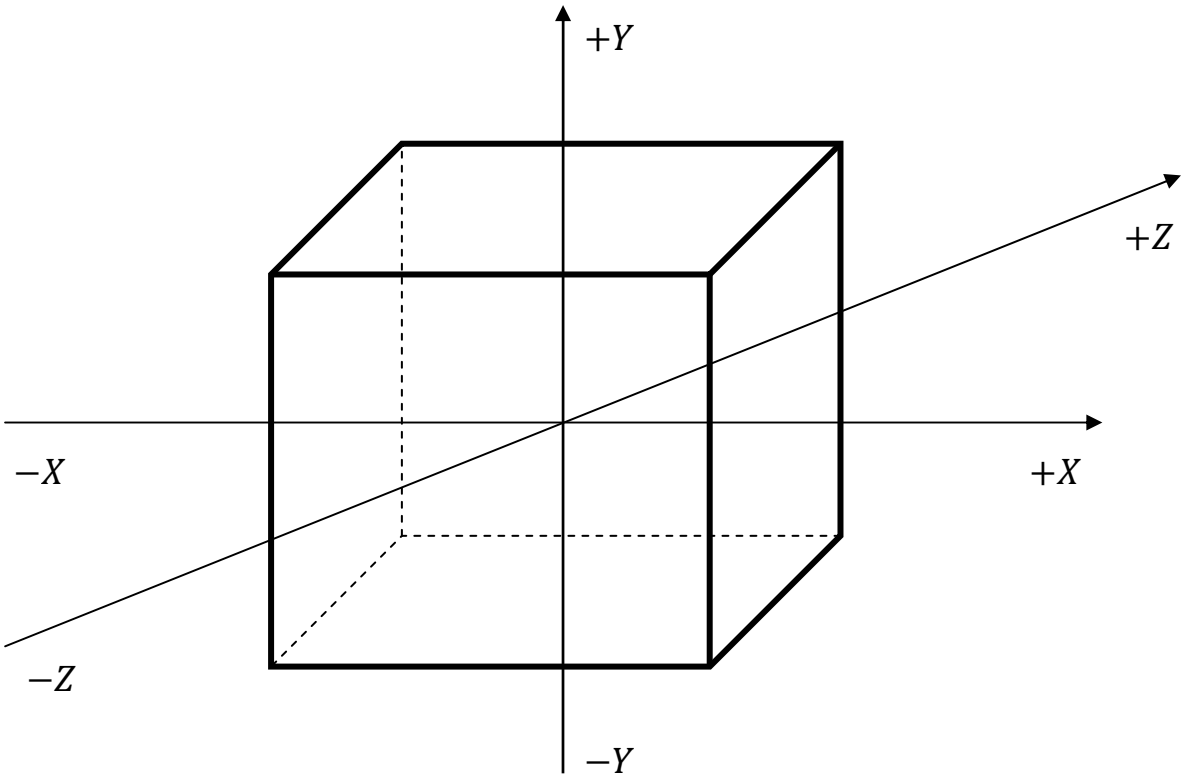


Рисунок 1.19 . Задание куба

Изображенный на рисунке 1.19 куб будет задаваться списком из 8 вершин

$$v0 = (+10, +10, +10)$$
$$v1 = (-10, +10, +10)$$
$$v2 = (-10, +10, -10)$$
$$v3 = (+10, +10, -10)$$
$$v4 = (+10, -10, +10)$$
$$v5 = (-10, -10, +10)$$
$$v6 = (-10, -10, -10)$$
$$v7 = (+10, -10, -10)$$

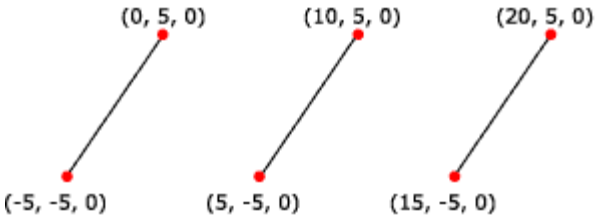
У куба 6 квадратных граней, то есть для его описания потребуется 12 троек чисел. Задающих треугольные грани: (0,1,2), (0,2,3), (3,2,6), (3,6,7), (0,3,7), (0,7,4), (1,2,6), (1,6,5), (0,1,5), (0,5,4), (4,5,6), (4,6,7).

Таким способом описываются практически все объекты для отрисовке в современных графических системах.

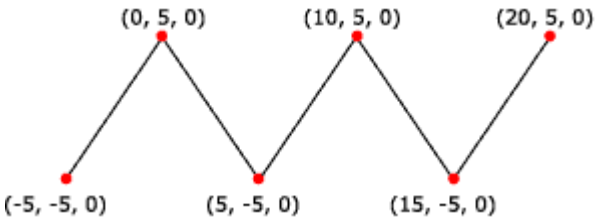
Кроме треугольных граней на списках вершин можно задавать еще отрезки и точки. Для задания последовательности точек индексный буфер не нужен.

Для задания отрезков индексный буфер может трактоваться двояко:

- Пары отрезков (line lists) — индексы в индексном списке группируются парами. Каждая пара означает начало и конец соответствующего отрезка



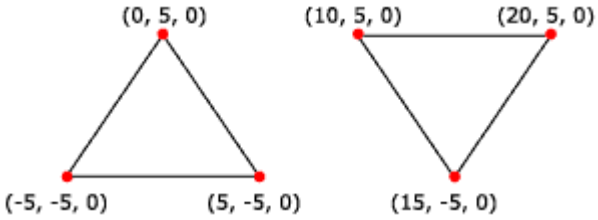
- Ломаная (line strip) — индексы в индексном списке означают последовательные соединяемые вершины. То есть для задания ломаной из n отрезков потребуется всего $n + 1$ индексов



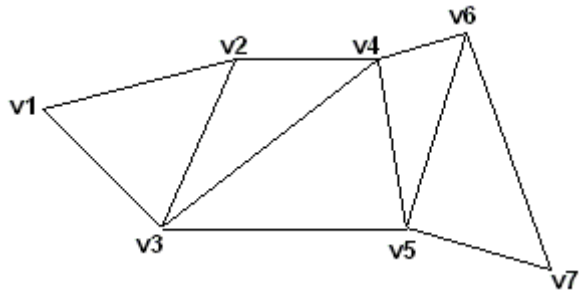
Такой способ представления данных очень удобен и экономичен при задании длинных непрерывных линий.

Задание треугольных граней также не ограничивается тройками чисел. Существуют три способа описания граней:

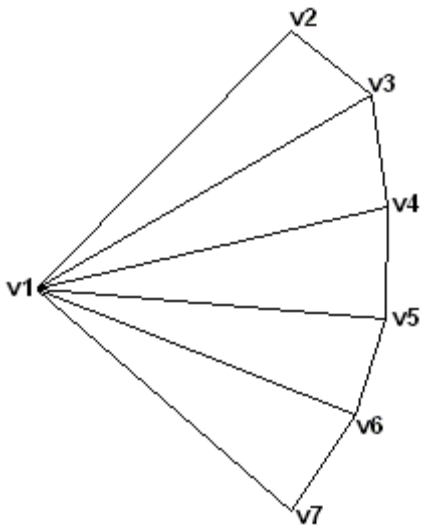
Список треугольников (triangle lists) — индексы вершин группируются по три, каждая тройка означает свой треугольник. Эта схема наиболее общая, но не самая экономичная



- Лента треугольников (triangle strip) — первые три индекса при таком задании формируют первый треугольник, четвертый индекс вместе с двумя предыдущим — следующий, и так далее. Это один из самых экономичных способов задания треугольников. Часто для создания разрывов в одной ленте создают вырожденные треугольники, и даже при таком допущении этот способ отрисовки остается эффективным.



- Веер треугольников (triangle fan) — первый индекс указывает на основание веера, второй и третий образуют с ним первый треугольник, третий и четвертый индексы — следующий, и так далее. Каждый новый треугольник образуется с помощью первого, последнего и предпоследнего индексов.



Такая схема не менее экономична, чем ленты треугольников, однако с помощью нее сложнее формировать сложные объекты — требуется больше разрывов и больше вырожденных треугольников.

Все описанные схемы задания трехмерных объектов применяются с современных видеоадаптерах и имеют специальное аппаратное ускорение.

1.3.13. Локальные и глобальные (мировые) координаты

Расположение вершин для объекта задается в некоторых абстрактных координатах, называемых для объекта собственными или локальными. Вообще говоря, при формировании сцены координаты всех вершин всех объектов могут быть заданы в одной общей системе координат (называемой глобальной или мировой). Но при таком задании координат любое перемещение какого-то объекта, создание копий одного или того же объекта было бы затруднительным.

Более того, представим некоторую иерархическую систему — например, руку робота, или подъемный кран. Такой манипулятор состоит из нескольких звеньев, каждое из которых прикреплено к своему «родителю», и при повороте «родителя» перемещаются все прикрепленные «дочерние» объекты. Понятно, что решать подобную задачу в глобальных координатах было бы крайне сложно.

Для этого вводится понятие локальных координат объекта. Каждый объект задается в своих собственных координатах. Точка отсчета для этих координат выбирается исходя из общих соображений. Так, например, центром предплечья робота можно считать плечо, а центром стула — точку у его основания.

Теперь, для каждого объекта нужно задать некоторую матрицу, которая отвечает за его расположение в мировых координатах. Такая матрица называется мировой матрицей объекта.

Мировая (World) матрица объекта обуславливает его расположение в пространстве. Можно показать, что любая аффинная матрица преобразования может быть представлена в идее комбинации матриц перемещения, масштаба и поворота. Иными словами мировая матрица отвечает за расположение, ориентацию и размер объекта в мировых координатах.

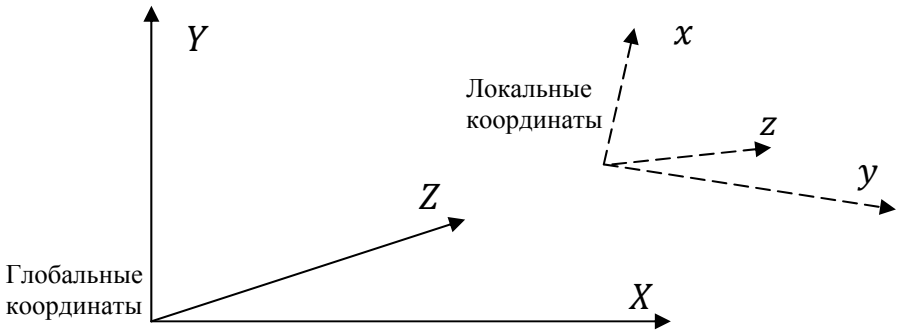


Рисунок. 1.20. Мировые координаты

Также мировую матрицу можно рассматривать как матрицу перевода локальных координат вершин объекта (x, y, z) в глобальные (X, Y, Z). То есть обозначив мировую матрицу как $World$, можно записать

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = World \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad (38)$$

Такая матрица хранится для каждого объекта. Соответственно, для получения координат каждого объекта в мировой системе координат, его координаты надо домножить на эту матрицу.

1.3.14. Матрица наблюдателя

Матрицы мирового преобразования позволяют перевести все локальные координаты объектов в глобальные. Таким способом формируется сцена в трехмерном пространстве. Однако для отображения сцены на экране необходимо учитывать, что сцена наблюдается из какой-то точки и в каком-то направлении.

Иными словами, в сцене всегда есть наблюдатель, глазами которого мы видим объекты. Можно сказать, что в определенной точке сцены размещена некоторым образом ориентированная мнимая камера. Изображение, которое мы в итоге хотим визуализировать, получается путем проецирования объектов на матрицу этой мнимой камеры.

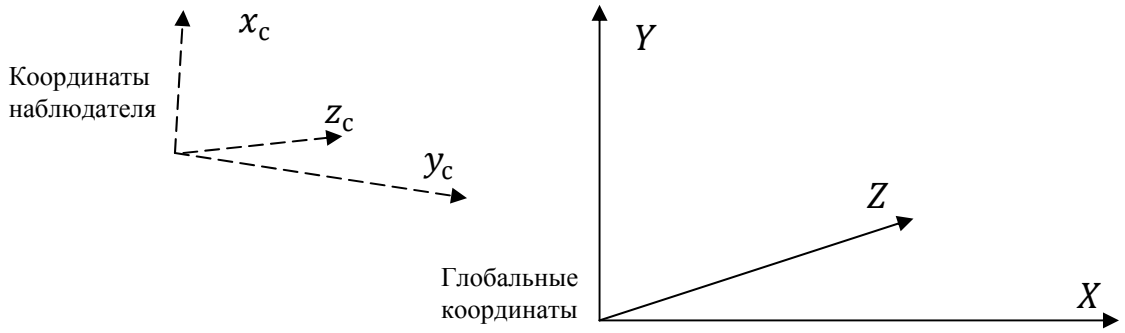


Рисунок 1.21. Матрица наблюдателя

Таким образом, объекты из мировой системы координат надо преобразовать в систему координат этой камеры. Для этого строится матрица преобразования обзора (View), позволяющая получать координаты объекта в системе координат наблюдателя.

$$\begin{pmatrix} X_{view} \\ Y_{view} \\ Z_{view} \\ 1 \end{pmatrix} = View \cdot World \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad (39)$$

Опять же, можно было бы отказаться от этой матрицы преобразования. Однако подход с камерой позволяет достаточно просто решать задачу отображения при перемещении камеры в статической сцене.

1.3.15. Проецирование

Окончательным результатом визуализации сцены является двумерное растровое изображение. Это изображение получается в результате проецирования трехмерного изображения на двумерную плоскость.

Задача отображений трехмерного мира на двумерной плоскости появилась задолго до первых компьютеров. Ведь, по сути, картина, которую рисует художник, также представляет собой двумерную проекцию трехмерного мира. По этой аналогии в компьютерной графике плоскость, на которою осуществляется проекция, называют картинной.

В изобразительном искусстве известно несколько видов различных проекций. По большому счету все такие проекции могут быть разделены на две группы — параллельные и точечные (перспективные).

Рисунок 1.22. Методы проектирования

Для получения проекции объекта на картинную плоскость необходимо провести через каждую его точку прямую из заданного проектирующего пучка (собственного или несобственного) и затем найти координаты точки пересечения этой прямой с плоскостью изображения. В случае центрального проектирования все прямые исходят из одной точки — центра собственного пучка. При параллельном проектировании центр (несобственного) пучка считается лежащим в бесконечности.

Основной особенность параллельной проекции является сохранение отношения линейных размеров. Объекты, имеющие равные размеры в пространстве, при параллельном проектировании будут иметь равные по размеру образы. При перспективном проектировании более удаленный объект будет иметь меньшие размеры, чем более близкий.

Параллельное проектирование может быть разделено на несколько категорий, изображенных на рисунке 1.23.

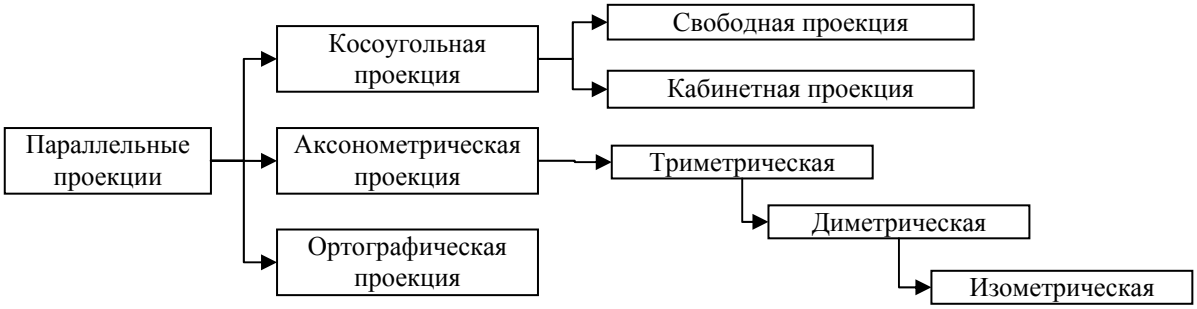


Рисунок 1.23. Классификация проекций

1.3.16. Ортографическая проекция

При ортографической (или ортогональной) проекции — проецирование осуществляется вдоль одной из осей (по ортам). При построении всех ортогональных видов проецирующие лучи перпендикулярны картинной плоскости. В техническом и архитектурном черчении часто на одном чертеже совмещают три ортогональные проекции — фронтальную, в плане и боковую. Для каждой из проекций картинная плоскость расположена параллельно соответствующей главной грани объекта.

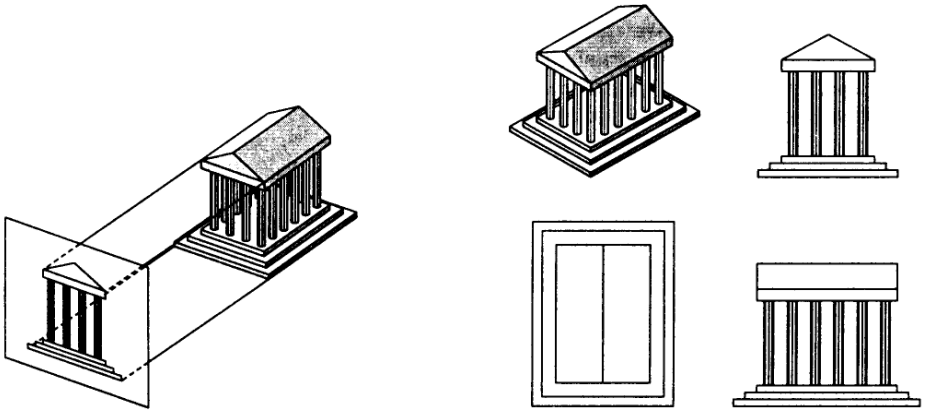


Рисунок 1.24. Ортогональные проекции

Использование только ортогональных проекций требует от наблюдателя определенного искусства, умения восстановить пространственную форму объекта по этим проекциям. В технике даже имеется для этого специальный термин — «умение читать чертежи». Такие проекции обладают весьма важным свойством — они сохраняют как пропорции размеров элементов, так и углы между ними.

1.3.17. Аксонометрические проекции

Аксонометрические проекции позволяют иметь на чертеже изображение не одной главной грани, а двух-трех, нужно устранить одно из ограничений, характеризующих ортографические проекции. В *аксонометрических (axonometric)* проекциях проецирующие лучи по-прежнему ортогональны картинной плоскости, как показано на рисунке, но сама картинная плоскость может иметь любую ориентацию относительно объекта.

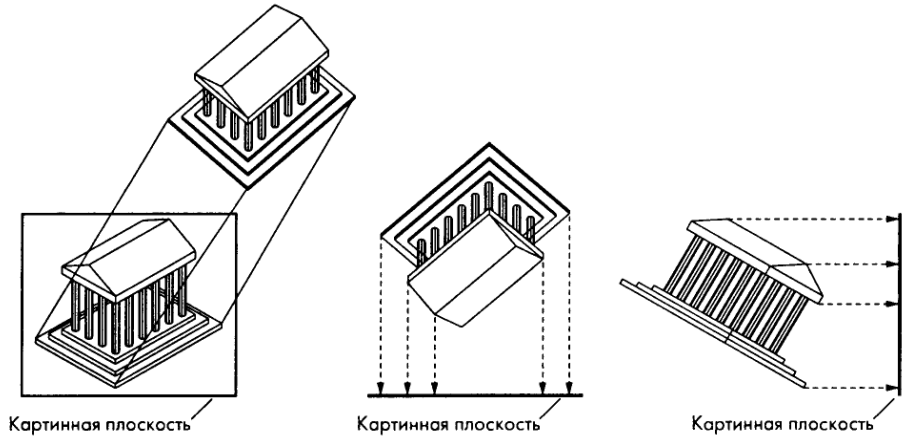


Рисунок 1.26. Аксонометрические проекции

Если картинная плоскость ориентирована симметрично по отношению к трем главным граням, пересекающимся в одном углу прямоугольного объекта, то образуется *изометрическая (isometric)* проекция. Если картинная плоскость ориентирована симметрично по отношению к двум главным граням, то образуется *диметрическая (dimetric)* проекция. Общий случай— *триметрическая (trimettic)* проекция. Все три вида аксонометрических проекций показаны на рисунке.

Рисунок 1.27. Виды аксонометрических проекций

Обратите внимание на то, что на изометрическом виде длины отрезков меньше, чем на исходном объекте. В изометрической проекции *коэффициент искажения (foreshortening)* длин одинаков по всем трем главным осям, а потому такой вид можно использовать для измерения длин. В диметрической проекции коэффициент искажения длин по двум осям одинаков, а по третьей отличается, т.е. мы имеем дело с парой коэффициентов. В триметрической проекции коэффициенты искажения по всем трем осям различны. В аксонометрических проекциях сохраняется параллельность прямых, но углы между ними искажаются. Окружность в аксонометрической проекции преобразуется в эллипс. Эти искажения — плата за возможность видеть на одной плоской проекции больше, чем одну главную грань объекта, при том что методы построения аксонометрических проекций достаточно просты и давно освоены в техническом черчении. Аксонометрические проекции широко применяются в архитектурном планировании и техническом конструировании.

1.3.18. Косоугольные проекции

Косоугольная (oblique) проекция является параллельной проекцией общего вида. При построении косоугольной проекции не накладываются никакие ограничения на угол между проецирующими лучами и картинной плоскостью, как изображено на рисунке.

В косоугольной проекции сохраняются углы между прямыми на объекте, расположенными в плоскости, параллельной картинной. Окружность в такой плоскости также проецируется в окружность и при этом на изображении присутствует более чем одна главная грань объекта. Строить косоугольные проекции вручную — задача довольно сложная. Кроме того, они выглядят все-таки как-то ненатурально. В большинстве физических систем форми-

рования изображения — в фотоаппаратах или в глазах человека — плоскость объектива параллельна картинной плоскости. Хотя в таких приборах фактически формируется перспективная проекция, но при разглядывании достаточно удаленных объектов получается почти параллельная проекция, но именно ортогонально параллельная, поскольку картинная плоскость параллельна плоскости объектива. Некоторые фотокамеры с растягивающимся мехом (их теперь можно увидеть только в музее или лавке антиквара) оснащены механизмом, который позволяет поворачивать объектив, сохраняя неизменной ориентацию фотопластинки. Это один из немногих физических приборов, в котором можно получить проекцию, близкую к параллельной косоугольной.

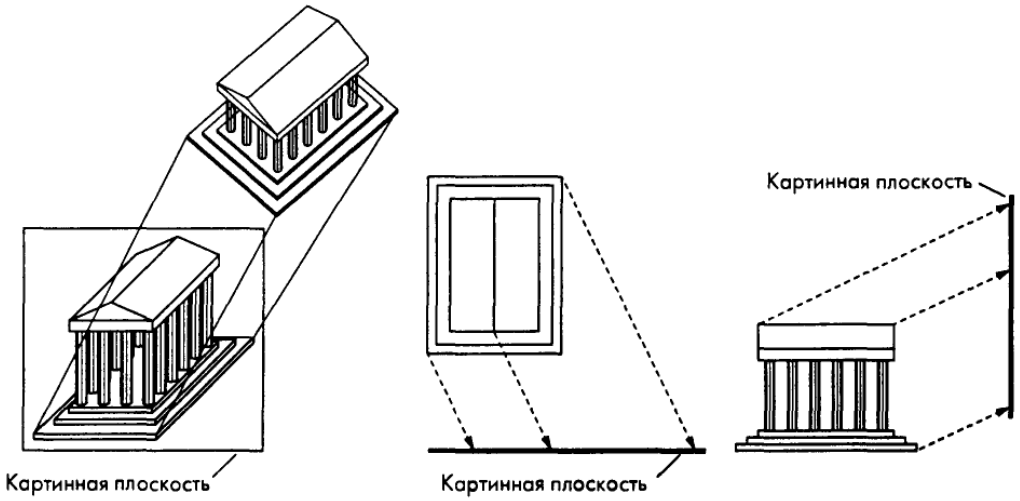


Рисунок 1.27. Косоугольная проекция

С точки зрения прикладного программиста особой разницы между разными видами параллельных проекций нет. От него требуется только указать тип проекции — параллельная или перспективная — и набор параметров, специфицирующих положение и характеристики камеры. Единственная проблема — как задать эти параметры, если требуется получить определенный «классический» вид или наилучшим образом представить определенный объект наблюдателю.

1.3.19. Перспективные проекции

Все перспективные виды проецирования характеризуются наличием перспективных сокращений (diminution). Чем дальше отстоит объект от наблюдателя, тем меньше размер его изображения. Это искажение размеров и придает перспективному изображению натуральность, но оно же не позволяет использовать такое изображение для оценки действительных размеров изображенных объектов. Поэтому перспективные проекции применяются в основном в тех областях приложения компьютерной графики, где важна именно натуральность изображения с точки зрения наблюдателя — в архитектуре и кинематографии.

Рисунок 1.28. Классификация перспективных проекций

При построении перспективной проекции классическими методами предполагается, что наблюдатель расположен симметрично относительно картинной плоскости (точнее, сторон ограничивающего прямоугольника в картинной плоскости), как показано на рисунке. Таким образом, пирамида, основанием которой является ограничивающий прямоугольник в картинной плоскости, а вершиной — центр проецирования, является симметричной, или правильной пирамидой. Эта симметрия порождается геометрией конструкции большинства оптических приборов, в том числе и глаза человека. Но в некоторых фотокамерах, в частности фотокамерах с мехом, плоскость фотопластины можно поворачивать относительно плоскости объектива. В результате можно формировать перспективную проекцию общего вида. Те модели построения перспективы, которые используются в компьютерной графике, распространяются и на такой случай.

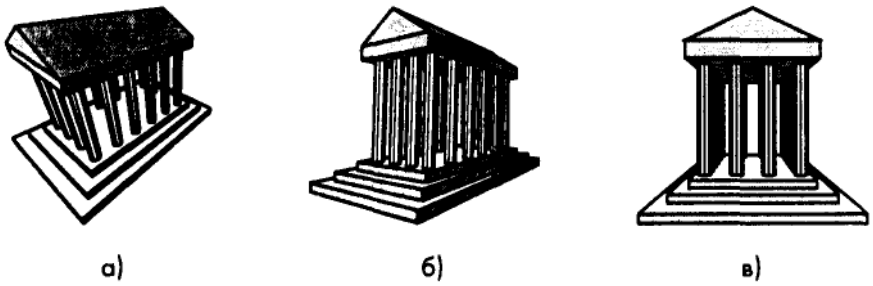


Рисунок 1.28. Виды перспективных проекций

В классической графике известны одно-, двух- и трехточечная перспективные проекции. Разница между этими тремя видами заключается в том, сколько направлений сохраняет параллельность при проецировании. Рассмотрим три разные перспективные проекции здания на рисунке. К каждому углу здания подходят линии, параллельные его главным осям. В наиболее общем случае — трехточечной перспективной проекции — линии, направленные вдоль всех трех осей, которые на объекте параллельны, становятся на проекции непараллельными и сходятся в тех разных точках схода (vanishing points.

При двухточечной перспективе линии, проходящие вдоль одной из главных осей, сохраняют параллельность и на изображении, а при одноточечной перспективе сохраняют параллельность линии, проходящие вдоль двух главных осей. В двухточечной перспективе имеются две точки схода, а в одноточечной — одна. С точки зрения прикладного программиста все опи-

санные варианты являются частным случаем обобщенной перспективной проекции, математическое описание которой будет детально рассмотрено ниже.

1.3.20. Матрица ортогонального проецирования

Ортогональная проекция — это частный случай параллельной проекции, при которой проецирующие лучи ортогональны картинной плоскости. Можно с определенной точностью считать, что ортогональная проекция получается в камере, имеющей объектив с очень большим фокусным расстоянием, причем картинная плоскость камеры ортогональна оптической оси ее объектива. Однако уравнения ортогонального проецирования можно вывести и непосредственно, не прибегая к «переносу» в бесконечность центра проецирования математической модели.

На рисунке показана ортогональная проекция на картинную плоскость описываемую уравнением $z = 0$.

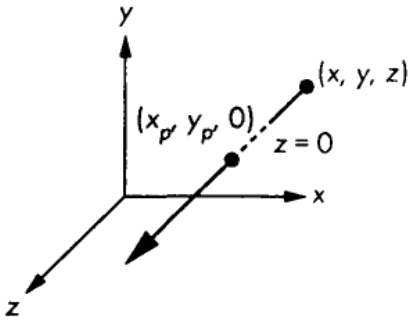


Рисунок 1.29. Расчет ортогонального проецирования

При проецировании точки на эту плоскость сохраняются значения ее компонентов x и y, а уравнения проецирования будут иметь вид:

$$x_p = x, y_p = y, z_p = 0 \tag{40}$$

Записав те же уравнения в матричной форме, получим матрицу ортогонального проецирования.

$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{41}$$

При выполнении ортогонального проективного преобразования отпадает необходимость в делении, хотя для обоих видов проективного преобразования можно использовать одну и ту же аппаратную реализацию конвейерной обработки.

Другие виды ортогонального проектирования могут быть получены при помощи изменения матрицы View, отвечающей за положение наблюдателя.

1.3.21. Матрица перспективного проецирования

Будем считать, что камера находится в начале координат, и ее ось направлена вдоль отрицательной полуоси z. Любое другое положение камеры может быть переведено в указанное при помощи преобразования матрицы View, которая и отвечает за положение камеры в пространстве.

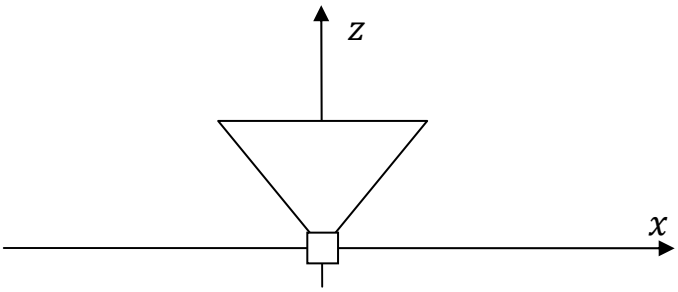


Рисунок 1.30. Расположение перспективной камеры

Картинная плоскость ортогональна оптической оси объектива. Этот вариант характерен для большинства реальных оптических приборов, в том числе и для глаза человека. Таким же способом описывается и мнимая камера наблюдателя в трехмерной сцене.

Плоскость проекции можно поместить перед центром проецирования. Если картинная плоскость перпендикулярна оси проекции, то сформируются виды, схематически представленные на рисунке.

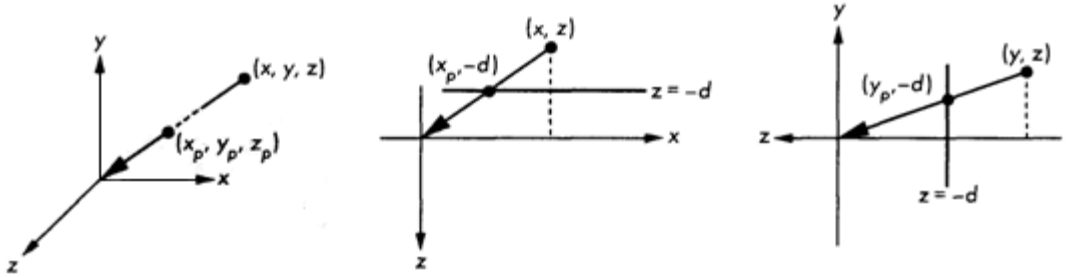


Рисунок 1.31. Расчет перспективного проецирования

Точка, имеющая в мировом пространстве координаты (x, y, z), проецируется в точку (x_p, y_p, z_p). Все проецирующие лучи сходятся в начале координат, и, поскольку картинная плоскость перпендикулярна оси z, $z_p = d$. Так как камера ориентирована в направлении отрицательной полуоси z, картинная плоскость находится в отрицательном полупространстве $z < 0$ и значение d также отрицательно.

На рисунке показаны два подобных треугольника, для которых справедливы соотношения:

$$\frac{x}{z} = \frac{x_p}{z} \implies x_p = \frac{x}{z/d} \tag{42}$$

Аналогичный результат можно получить и для y_p. Эти уравнения не линейны. Деление на «глубину» z как раз реализует перспективное сокращение — чем дальше объект, тем меньше линейные размеры его проекции.

В общем случае перспективное преобразование сохраняет линейность, но не является аффинным. Кроме того, оно необратимо. Поскольку все точки, расположенные на проецирующем луче, преобразуются в одну и ту же точку, восстановить их исходное положение нельзя.

Для работы с проективными преобразованиями нам придется слегка модифицировать используемый аппарат однородных координат. При описании однородных координат мы представляли трехмерную точку (x, y, z) в виде четырехмерной $(x, y, z, 1)$, Поскольку однородные координаты задают класс эквивалентности, мы будем рассматривать эту точку в виде (xw, yw, zw, w).

Если $w \neq 0$, то трехмерное представление точки можно получить, разделив все компоненты на w. Естественно, мы будем стараться работать с $w = 1$, поскольку это позволяет избежать дополнительных операций деления при восстановлении трехмерных координат. Однако, изменяя w, можно представить в матричном виде более широкий класс преобразований, в том числе и перспективное.

Рассмотрим матрицу M, заданную следующим образом:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix} \tag{43}$$

Преобразуем с помощью этой матрицы точку $p(x, y, z)$ в трехмерном пространстве.

$$p' = Mp = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} \equiv \begin{pmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{pmatrix} \tag{44}$$

Сравнив результат с (42), обнаружим, что в результате получилось точка, преобразованная перспективным отображением. Таким образом, ца M описывает самое простое перспективное преобразование.

В дальнейшем будем называть матрицу проектирования $Projection$, или просто $Proj$. Таким образом, с помощью этой матрицы можно перевести координаты объекта из пространства камеры в координаты экрана

Далее будет показано, как при помощи внесения небольших изменений в матрицу учесть размеры экрана пользователя, угол обзора камеры и прочие важные факторы.

1.3.22. Характеристики перспективного преобразования

В предыдущем разделе при построении матрицы перспективного проектирования не учитывались важные параметры камеры, такие как угол обзора, размер экрана, и плоскости отсечения.

Угол обзора (angle of view), иногда называемый полем обзора (field of view, FOV), определяет, под каким максимальным углом расходятся лучи, идущие от камеры.

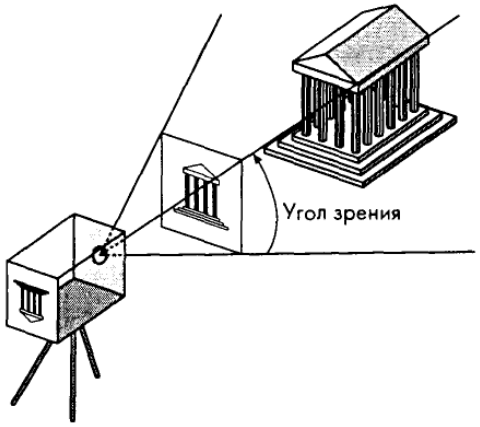


Рисунок 1.31. Угол зрения

По углам обзора объективы камер принято делить на длиннофокусные и широкоугольные. Широкоугольные объективы обладают большим углом зрения и вносят сильное перспективное искажение в изображаемые объекты. Такие объективы позволяют снять большую площадь кадра, но искажения на краях изображения, полученного при широком угле зрения, могут быть крайне велики.

Длиннофокусные объективы, наоборот, обладают малым углом обзора, и с уменьшением угла изображение приближается к такому, которое могло бы быть получено при ортогональном проектировании. Такие объективы называют «схлопывающими перспективу», и используют для визуального приближения далеких объектов.

Угол зрения отвечает также и за видимость объектов. На изображении появляются только те объекты, которые попали внутрь угла зрения. Если окно на картинной плоскости имеет вид прямоугольника, то на изображении появятся только те объекты, которые окажутся внутри бесконечной пирамиды с вершиной в центре проецирования. Эта пирамида ограничивает зону видимости (view volume). Объекты, не попавшие в эту зону, отсекаются (clipped) и не включаются в отображаемую сцену. Таким образом, приведенное выше математическое описание процесса визуализации является неполным, поскольку мы не включили в него отсечение.

В большинстве графических API параметры отсечения определяются вместе с параметрами проецирования. В системах компьютерной графики зона видимости ограничивается не только с боков, но и вдоль оси проецирования — так называемыми передней и задней отсекающими плоскостями

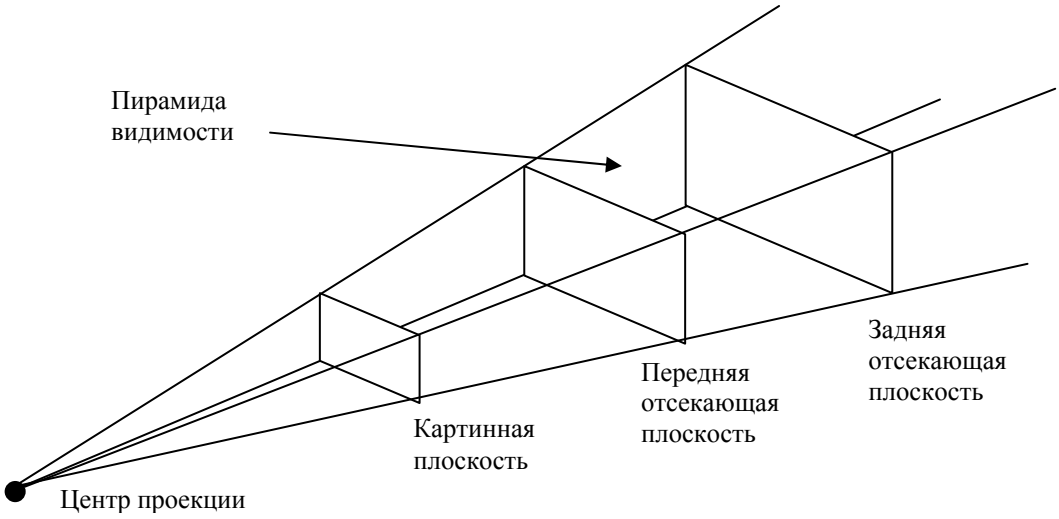


Рисунок 1.32. Пирамида видимости

В результате зона приобретает вид усеченной пирамиды видимости (frustum). В ней есть только один жестко зафиксированный параметр — центр проецирования, который находится в начале координат фрейма камеры. С помощью остальных шести параметров можно, в принципе, добиться любой ориентации и формы пирамиды, но такая гибкость используется в очень редких случаях

Рассмотрим теперь вид получаемой матрицы проецирования с учетом построения пирамиды видимости.

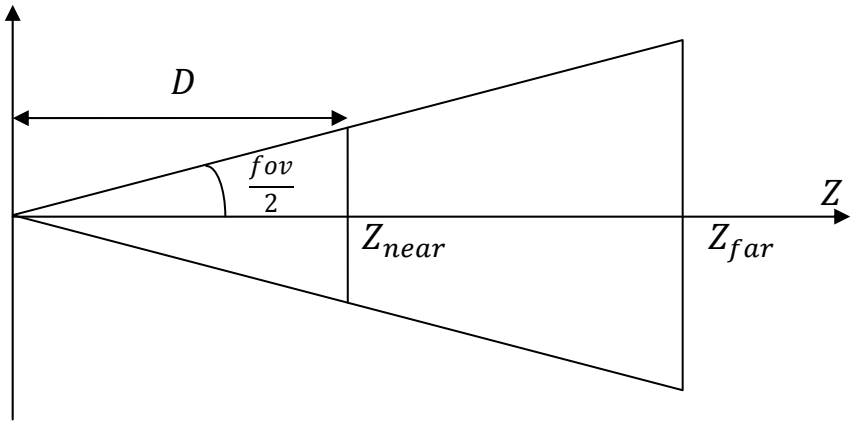


Рисунок 1.33. Построение матрицы проектирования

При построении такой матрицы необходимо учесть следующие параметры:

- размеры экрана w и h, соответствующие ширине и высоте экрана в пикселях,

- угол обзора θ. Поскольку экран имеет прямоугольную, а не квадратную форму, возникает два разных угла обзора — горизонтальны и вертикальный. В большинстве систем чаще используется горизонтальный угол обзора, поэтому мы тоже будем придерживаться этого выбор,
- расстояние от точки расположения камеры до экрана d.
- расположение плоскостей отсечения Z_{near} и Z_{far}.

Отметим, что угол θ однозначно определяет соотношение между шириной экрана w и расстоянием до экрана d следующим образом:

$$\tan\frac{\theta}{2} = \frac{w/2}{d} \tag{45}$$

Таким образом, один их этих параметров можно исключить из рассмотрения. Поскольку размер экрана и угол обзора являются наиболее понятными конечному пользователю (в частности, угол обзора — это характеристика камеры, размеры экрана — характеристика системы визуализации), мы будем рассматривать именно их. Расстояние же до экрана для пользователя является наименее интересным параметром, и поэтому будет вычисляться исходя из формулы (45).

Параметры отсечения Z_{near} и Z_{far} не имеют никакого влияния на расположение точки на экране проекции. Однако для простоты работы механизма отсечения удобно получить глубину расположения точки в качестве значения результирующей компоненты z в спроецированных координатах. Подобную глубину удобно иметь в относительных координатах, то есть точкам на расстоянии Z_{near} от камеры будет соответствовать глубина 0, а точкам на расстоянии Z_{far}, соответственно, глубина 1.

Итак, для построения финальной матрицы перспективного проецированя необходимо учесть следующее:

- проецирование ведется на прямоугольную область с размерами $w \times h$, параллельную плоскости XY. Центр расположен на оси Z и удален от начал координат на $w/2\tan\frac{\theta}{2}$
- точки, удаленные по оси Z от начала координат на Z_{near}, должны иметь глубину 0, а на $Z_{far} - 1$.

Очевидно, требуемое преобразование можно описать при помощи следующих формул:

$$x' = \frac{x}{z/d} + \frac{w}{2} = \frac{xw}{2\,z\tan\frac{\theta}{2}} + \frac{w}{2}$$
$$y' = \frac{h}{2} - \frac{y}{z/d} = \frac{h}{2} - \frac{yw}{2\,z\tan\frac{\theta}{2}} \quad (46)$$
$$z' = \frac{z - Z_{near}}{Z_{far} - Z_{near}}$$

К сожалению, достигнуть линейности в последнем уравнении при построении соответствующей матрицы не удастся, однако можно построить матрицу, которая будет сохранять порядок точек по глубине и сохранять соответствие ближней и дальней отсекающих плоскостей глубинам 0 и 1.

Вернув обозначение расстояния до картинной плоскости d для облегчения записи и введя вспомогательный параметр $Q = \frac{\mathrm{Z_{far}}}{\mathrm{Z_{far} - Z_{near}}}$, можно записать следующую матрицу.

$$\begin{pmatrix} d & 0 & w/2 & 0 \\ 0 & -d & h/2 & 0 \\ 0 & 0 & Q & -Z_{far}Q \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} xd + \frac{wz}{2} \\ -yd + \frac{hz}{2} \\ zQ - Z_{far}Q \\ z \end{pmatrix} \equiv \begin{pmatrix} \frac{xd}{z} + \frac{w}{2} \\ -\frac{yd}{z} + \frac{h}{2} \\ Q(1 - \frac{Z_{far}}{z}) \\ 1 \end{pmatrix} \quad (47)$$

Координаты x' и y' полностью идентичны требуемым значениям. Рассмотрим полученную компоненту z' подробнее:

$$z' = Q\left(1 - \frac{Z_{far}}{z}\right) = \frac{Z_{far}(z - Z_{far})}{z(Z_{far} - Z_{near})} \quad (48)$$

Из формулы хорошо видно, что при $z = Z_{far}$ значение $z' = 1$, а для ближней отсекающей плоскости при $z = Z_{near}$ можно получить $z' = 0$. Можно заметить, что зависимость не линейна, однако она сохраняет непрерывность и положительную производную на требуемом участке, а значит, сохраняет упорядоченность глубины и выполняет требуемое условие для границ отсечения.

Вообще говоря, вопрос отсечений чрезвычайно важен для построения изображений в компьютерной графике. Сцена может состоять из нескольких миллионов треугольников, но в каждый момент не все треугольники будут видны. Более того, в большинстве случаев огромная часть треугольников будет невидима из-за отсечения областью видимости, или же перекрыта расположенными ближе треугольниками.

Задача сокрытия невидимых граней имеет много различных аспектов, которые мы и рассмотрим далее.

1.4. Отсечения

В процессе отсечения производится анализ, какие примитивы или части примитивов будут включены в изображение. Примитивы, которые попадают внутрь зоны видимости, модуль отсечения пропускает дольше, а остальные отбрасывает. Возможен и третий вариант — примитив частично попадает в зону видимости. При обработке таких примитивов модуль отсечения разделяет их на части и пропускает дальше только ту из них, которая находится в зоне видимости.

В процессе конвейерной обработки процедура отсечения может выполняться поэтапно на разных стадиях. Например, модуль моделирования может взять на себя часть работы, сразу же отбросив те объекты, которые несомненно не попадут в зону видимости, и тем самым избавить остальные модули от лишних операций. Отсечение можно выполнять и после проективного преобразования, когда из трехмерного описания объектов будет сформировано двухмерное. В системах визуализации можно выполнять отсечение на уровне трехмерной зоны видимости еще до того, как система приступит к проективному преобразованию.

Ниже мы рассмотрим несколько алгоритмов отсечения. Начнем с двухмерных алгоритмов отсечения отрезков не только потому, чтобы отдать дань пионерам компьютерной графики, но и из педагогических соображений — поняв идею двухмерного отсечения, значительно проще разобраться и в трехмерных алгоритмах.

1.4.1. Алгоритм Коэна-Сазерленда

Суть задачи отсечения двухмерных отрезков поясняется на рисунке. Предположим, что уже выполнено проецирование и в нашем распоряжении имеется двухмерное описание изображения в картинной плоскости. На этой же плоскости определена и рамка отсечения, которая соответствует видовому окну на экране дисплея. Все параметры заданы вещественными числами. Как видно на рисунке, отрезок AB полностью попадает на экран, а ни один из участков отрезка CD не попадает. Отрезки EF и GH нужно укоротить перед тем, как выводить на экран. Хотя отрезок прямой однозначно описывается координатами крайних точек, на примере отрезка GH вы можете убедиться, что, хотя крайние точки отрезка и лежат вне зоны видимости, часть отрезка все-таки может попадать в эту зону.

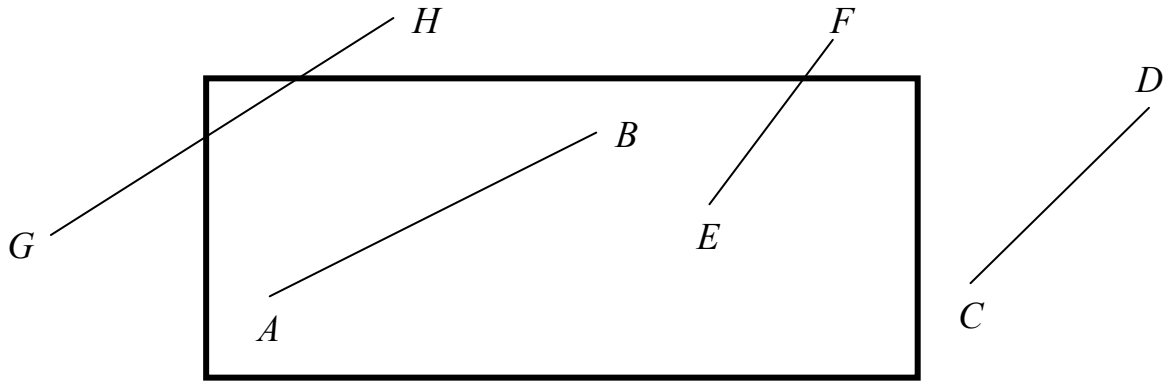


Рисунок 1.34. Задача двухмерного отсечения

Можно вычислить координаты точек пересечения прямой с рамкой видимости и использовать эту информацию для отсечения, но весь-то фокус в том и состоит, чтобы по возможности минимизировать объем вычислений и обойтись без определения точек пересечения, которое непременно включает операцию деления чисел с плавающей точкой. Исторически первым, отвечающим этим требованиям, был алгоритм Коэна-Сазерленда (Cohen-Sutherland algorithm), в котором большинство операций умножения и деления заменены операциями сложения и тельных чисел и побитовыми логическими операциями булевой алгебры.

Выполнение алгоритма начинается с продления сторон рамки отсечения в обе стороны до бесконечности, в результате чего картинная плоскость делится на девять областей, как изображено на рисунке.

1001	1000	1010	y_{max}
0001	0000	0010	y_{min}
0101	0100	0110	
x_{min}		x_{max}	

Рисунок 1.35. Расположение характеристических кодов

Каждой области присваивается четырехразрядный двоичный номер — характеристический код (outcode) — $b_0 b_1 b_2 b_3$, который формируется следующим образом. Пусть (x, y) — координаты некоторой точки на картинной плоскости. Тогда

$$b_0 = \begin{cases} 1, & y \geq y_{max} \\ 0, & y < y_{max} \end{cases} \tag{49}$$

Аналогично, b_1 приравнивается 1, если $y < y_{min}$, а значения b_2 и b_3 определяются отношением между компонентой x и абсциссами левой и правой границ рамки отсечения. В результате девяти областям присваиваются коды,

представленные на рисунке. При анализе отрезка первым делом определяется, в каких областях находятся его конечные точки, и им присваиваются соответствующие характеристические коды. Эта процедура требует выполнения восьми операций вычитания на каждый отрезок.

Рассмотрим отрезок, конечные точки которого имеют характеристические коды $o_1 = \text{outcode}(x1, y1)$ и $o_2 = \text{outcode}(x2, y2)$ Возможны четыре варианта сочетания характеристических кодов двух конечных точек, как изображено на рисунке:

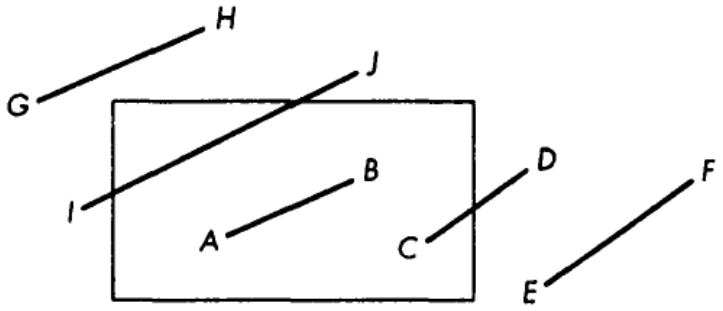


Рисунок 1.36. Варианты расположения отрезков

Варианты таковы:

1. ($o_1 = o_2 = 0$) Обе конечные точки лежат внутри рамки отсечения — этот случай представлен отрезком AB на рисунке. Весь отрезок при этом также находится внутри рамки отсечения и может быть передан дальше для выполнения растрового преобразования.
2. ($o_1 \neq 0, o_2 = 0$ или наоборот) Одна точка находится внутри рамки отсечения, а вторая — вне ее (отрезок CD на рисунке). В этом случае отрезок необходимо разделить. Отличный от нуля характеристический код указывает, в какой внешней области находится одна из конечных точек и какая именно граница (или границы) рамки пересекается отрезком. В этом случае необходимо вычислить одну или в худшем варианте две точки пересечения отрезка с границами рамки. Обращаю ваше внимание на то, что после вычисления первой точки пересечения (например, с горизонтальной границей) можно сформировать ее характеристический код и выяснить, требуется ли определять вторую точку пересечения (на этот раз с вертикальной границей).
3. ($o_1 \& o_2 \neq 0$) По результату побитовой операции AND над характеристическими кодами крайних точек можно выяснить, лежат ли они по одну сторону от границы рамки или по разные. Если результат отличен от нуля, то конечные точки лежат по одну сторону от какой-либо границы, а значит, весь отрезок лежит вне рамки отсечения и его можно спокойно отбросить (отрезок EF на рисунке).
4. ($o_1 \& o_2 = 0$) Обе конечные точки лежат вне рамки отсечения, но по разные стороны от двух ее границ. Этот вариант представлен отрезками GH и IJ на рисунке, и мы не можем с уверенностью ска-

зать, пересекает отрезок зону видимости или нет. Требуется более тщательный анализ — нужно вычислить точку пересечения с одной из границ рамки и проанализировать характеристические коды крайних точек двух новых отрезков.

Для анализа характеристических кодов достаточно только булевых побитовых операций над двоичными числами, которые выполняются очень быстро. Вычисление точек пересечения выполняется чрезвычайно редко и только там, где без этой информации не обойтись,— во втором и четвертом вариантах сочетаний характеристических кодов.

Алгоритм Коэна-Сазерленда прекрасно работает при анализе сцен, содержащих множество прямолинейных отрезков, немногие из которых действительно попадают в формируемое изображение. В этом случае большинство отрезков отбрасывается в результате логического анализа характеристических кодов конечных точек, что не занимает много времени. Алгоритм без труда можно распространить и на трехмерный случай.

Но мы не рассмотрели, как вычислять точки пересечения отрезка с границами рамки. Какой из возможных методов предпочесть, зависит от формы представления отрезка в программе, но в любом случае потребуется, как минимум, одна операция деления действительных чисел. Если используется стандартная явная форма представления прямой $y = mx + h$, где m — коэффициент наклона прямой, а h — ордината точки пересечения с осью y, то значения m и h можно вычислить по известным координатам крайних точек.

Но в такой форме нельзя представить вертикальную линию — известный всем недостаток явной формы представления. Если необходимо вычислять точки пересечения только в процессе реализации алгоритма Коэна-Сазерленда, то соответствующая функция будет довольно простой, поскольку отрезок всегда пересекается линией, параллельной одной из координатных осей. Но, скорее всего, нам потребуется функция, которая бы определяла точку пересечения прямых произвольной ориентации. Поэтому лучше ориентироваться на представление прямой в параметрической форме.

1.4.2. Алгоритм Лианга-Барского

В алгоритме Лианга-Барского (Liang-Barsky) используется параметрическая форма представления прямой. Пусть отрезок задан двумя крайними точками $p_1 = (x_1, y_1)^T$ и $p_2 = (x_2, y_2)^T$. Тогда параметрические уравнения прямой примут вид

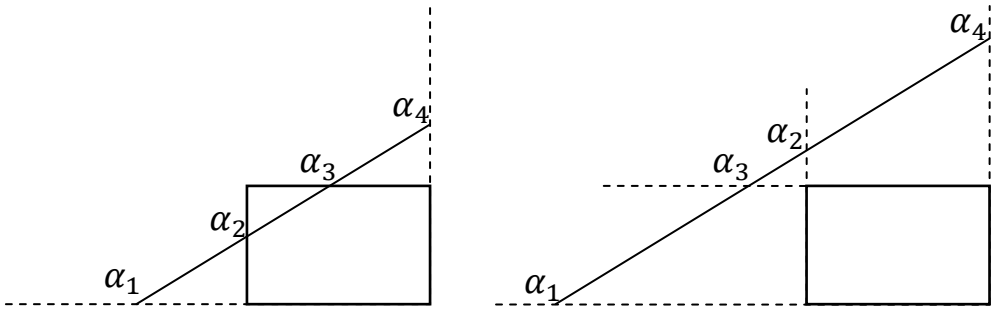
$$\begin{aligned} x(\alpha) &= (1-\alpha)x_1 + \alpha x_2 \\ y(\alpha) &= (1-\alpha)y_1 + \alpha y_2 \end{aligned} \tag{50}$$

Переходя к матричной форме записи, получим:

$$p(\alpha) = (1-\alpha)p_1 + \alpha p_2 \tag{51}$$

Параметрические уравнения такого вида описывают прямую любой ориентации, не делая исключений и для вертикальной или горизонтальной. При изменении значения параметра a от 0 до 1 точка на прямой перемещается от конечной точки отрезка p_1 до другой конечной точки p_2. Отрицательные значения параметра a задают точки, расположенные на продолжении отрезка перед конечной точкой p_2 а значения $\alpha > 1$ — точки, расположенные на продолжении отрезка после конечной точки p_2.

Рассмотрим прямую и отрезки на ней, представленные на рисунке.



а) б)

Рисунок 1.37. Алгоритм Лианга-Барского

Если прямая не параллельна одной из границ рамки (а такая ситуация очень просто выявляется), то она пересекает продолженные в бесконечность в обе стороны границы рамки в четырех точках, причем точкам пересечения соответствуют четыре значения параметра: $\alpha_1, \alpha_2, \alpha_3, \alpha_4$.

Значение α_1 соответствует точке пересечения отрезка с продолжением нижней горизонтальной границы, α_2 — левой вертикальной границы, α_3 — верхней горизонтальной границы, а α_4 — правой вертикальной границы. Если отрезок пересекает рамку видимости, то одно из этих значений соответствует точке, в которой отрезок «входит» в зону видимости, а другое — точке, в которой отрезок «покидает» зону. Не будем сейчас останавливаться на том, как определяются значения параметра в точках пересечения, а обратим внимание на порядок, в котором располагаются значения параметров для точек пересечения с горизонтальными и вертикальными границами. Для варианта, представленного на рисунке, порядок будет следующим:

$$1 > \alpha_4 > \alpha_3 > \alpha_2 > \alpha_1 > 0 \qquad (52)$$

Все четыре точки пересечения лежат на отрезке, причем значения параметров для двух из них — α_2 и α_3 — задают крайние точки того участка отрезка, который попадает в зону видимости.

Теперь рассмотрим случай, представленный на рисунке (б). При таком расположении анализируемого отрезка имеем следующий порядок значений параметров в точках пересечения с продолжением границ рамки:

$$1 > \alpha_4 > \alpha_2 > \alpha_3 > \alpha_1 > 0 \qquad (53)$$

Для эффективной реализации такой стратегии нужно разработать алгоритм, который минимизировал бы количество операций с числами в формате с плавающей точкой, необходимых для определения точек пересечения.

Множество отрезков можно отсеять, не вычисляя все четыре точки пересечения. При определении точек пересечения также желательно не использовать, где это возможно, операции деления действительных чисел. Значение параметра для точки пересечения с верхней границей рамки находится из соотношения

$$\alpha_3 = \frac{y_{max} - y_1}{y_2 - y_1} \tag{54}$$

Аналогичные соотношения имеют место и для точек пересечения с тремя остальными границами рамки. Приведенное соотношение можно представить в более удобном виде $\alpha_3 \Delta y = \Delta y_{max}$.

Практически весь логический анализ можно выполнить в терминах Δy_{max} и Δy и аналогичных им для других границ рамки видимости, и таким образом избавиться от большинства операций деления. Они понадобятся только в тех случаях, когда окажется, что отрезок пересекает зону видимости и что нужно определить точные значения параметров в точках пересечения границ.

Преимущество описанного алгоритма по сравнению с алгоритмом Коэна-Сазерленда состоит в том, что при пересечении зоны видимости можно сразу определить участок, попадающий внутрь зоны, и избежать повторного анализа.

Существует и множество других алгоритмов отсечения отрезков на картинной плоскости, но в отличие от рассмотренных, их не удается распространить на трехмерный случай.

1.4.3. Отсечение многоугольников

Необходимость в отсечении многоугольников возникает в разных ситуациях. Естественно, что отсечение необходимо выполнять перед или во время растрового преобразования, учитывая заданные размеры области экрана, отводимой для изображения. При этом иногда желательно, чтобы рамка отсечения принимала форму, отличную от прямоугольной. При формировании теней и удалении невидимых поверхностей возникает необходимость выполнить отсечение одного многоугольника другим многоугольником произвольной формы.

Например, на рисунке показана тень, сформированная отсечением многоугольника, ближайшего к источнику света, другим, более удаленным от источника многоугольником.

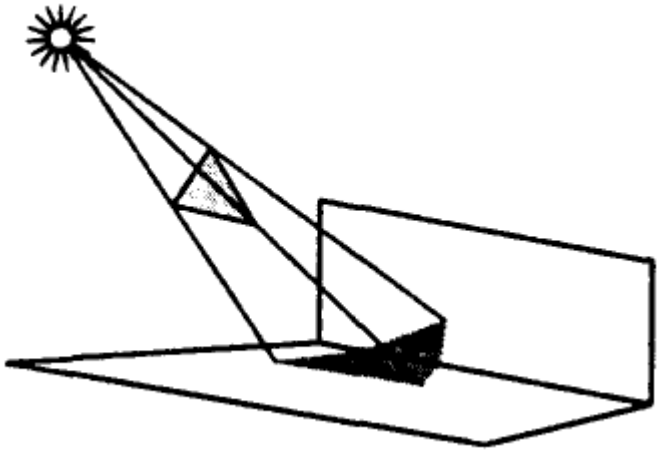


Рисунок 1.37. Построение тени отсечением многоугольника

Можно разработать алгоритм отсечения многоугольников, взяв за основу алгоритм отсечения отрезков и применив его последовательно ко всем ребрам многоугольника. Но при этом нужно не забывать, что многоугольник представляет собой единый объект и что при его отсечении может быть сформировано несколько новых объектов-многоугольников.

Рассмотрим представленный на рисунке многоугольник общего вида — «вогнутый» (concave) многоугольник. Если применить к нему отсечение прямоугольной рамкой, то получим три новых многоугольника. Но, к сожалению, реализовать модуль отсечения, который был бы способен из одного объекта сформировать несколько, довольно сложно. Поэтому чаще всего результат отсечения, подобный показанному на рисунке, представляется в программе как единый многоугольник, для чего в модуле отсечения выполняется объединение фрагментов в единый объект. В таком многоугольнике некоторые ребра накладываются друг на друга, что иногда служит источником проблем при решении других задач.

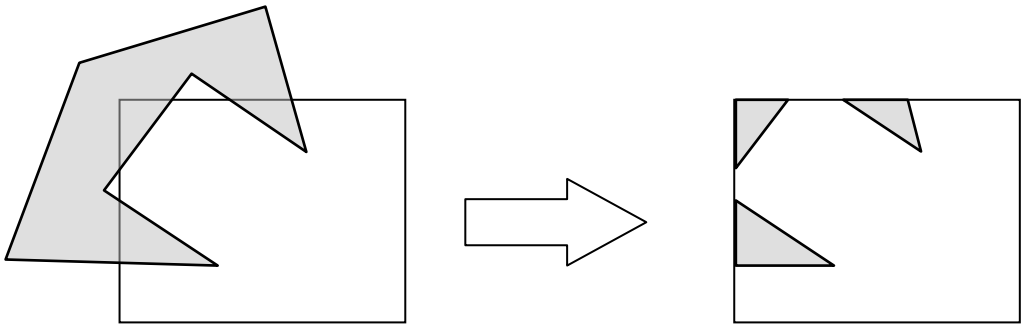


Рисунок 1.38. Отсечение многогранника

При отсечении одного выпуклого (convex) многоугольника другим выпуклым многоугольником, в частности прямоугольной рамкой зоны видимости, принципиально не может образоваться больше одного многоугольника, причем тоже выпуклого. Таким образом, в этом случае проблема «размножения» не возникает, и в большинстве графических систем либо разрешается формировать только выпуклые многоугольники, либо имеются средства рассечения (tessellation) многоугольника общего вида на выпуклые.

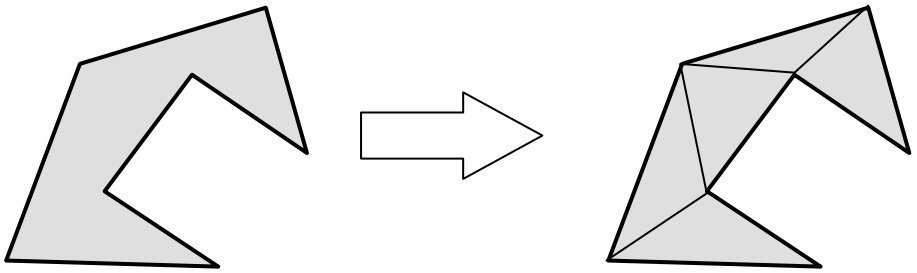
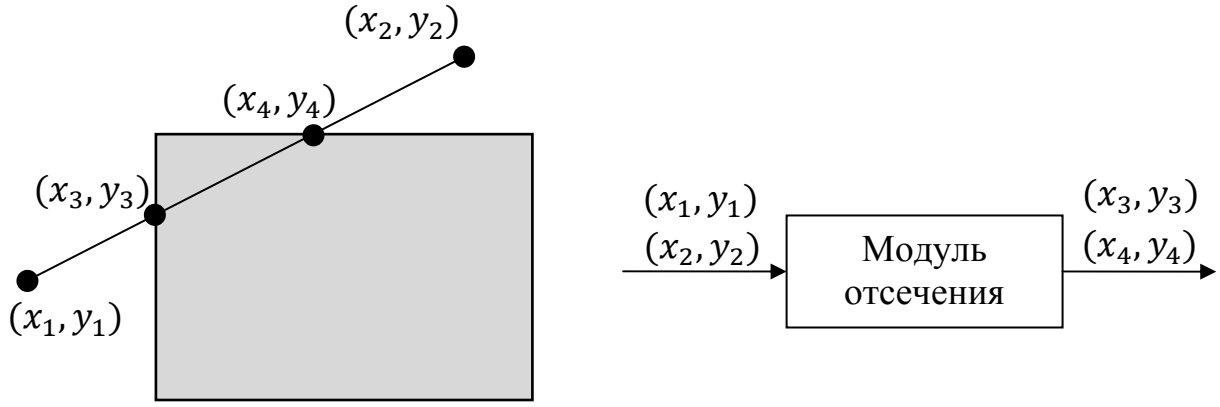


Рисунок 1.39. Тесселяция при отсечении

Для отсечения многоугольников прямоугольной рамкой можно использовать оба описанных выше алгоритма отсечения отрезков, применяя их в цикле по отношению к ребрам анализируемого многоугольника. В системах с конвейерной архитектурой более эффективным является другой подход, предложенный Сазерлендом (Sutherland) и Ходжменом (Hodgeman).

Модуль отсечения отрезка можно рассматривать как "черный ящик", на вход которого подаются две пары координат конечных точек исходного отрезка, а на выходе формируются две пары координат конечных точек того участка этого отрезка, который лежит внутри зоны отсечения, или не формируется ничего, если отрезок не пересекает зону отсечения.



а) б)

Рисунок 1.40. Модульное отсечение

Будем рассматривать рамку отсечения как объект, сформированный пересечением четырех бесконечных прямых, соответствующих верхней, нижней, правой и левой границам зоны видимости. Тогда и модуль отсечения от-

резка можно представить как последовательность четырех более простых модулей (конвейер), каждый из которых имеет дело только с одной границей.

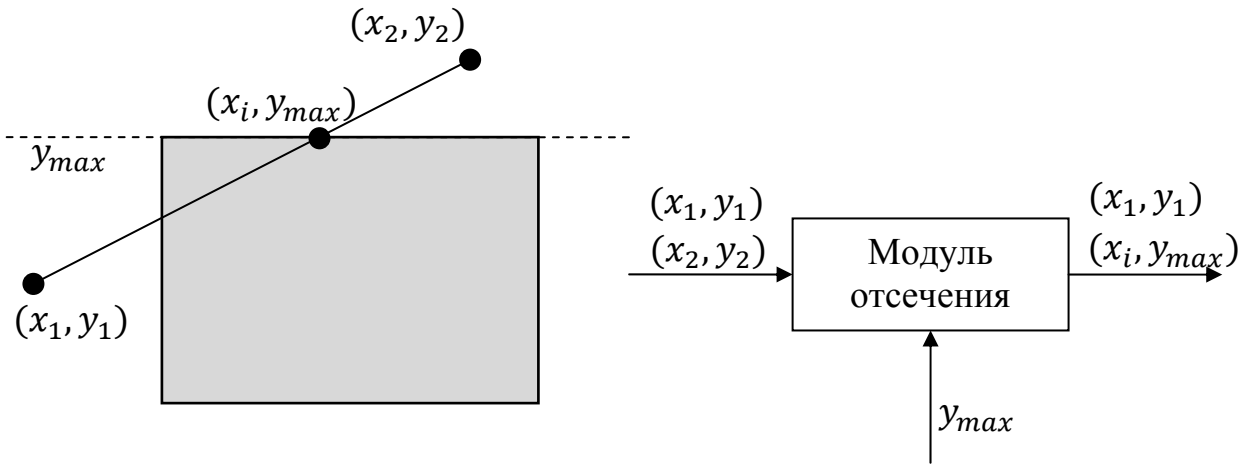


Рисунок 1.41. Модуль отсечения

Рассмотрим отсечение верхней границей рамки. Модуль, выполняющий эту операцию, получает на входе координаты крайних точек отрезка и значение y_{max} в качестве параметра, задающего зону отсечения.

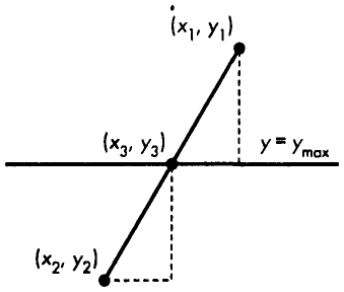
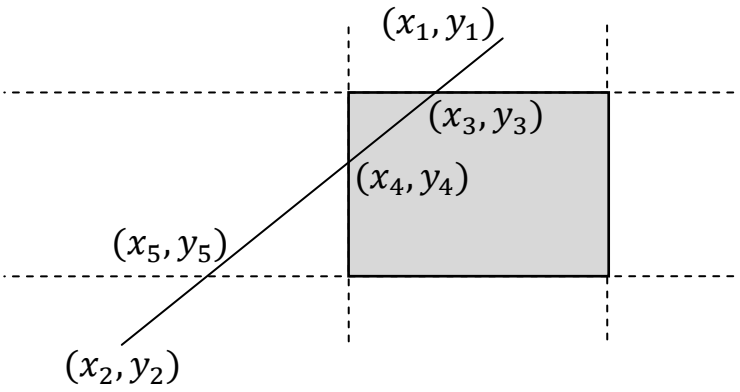


Рисунок 1.42. Расчет точки пересечения

Нетрудно показать, используя подобие треугольников на рисунке, что если отрезок пересекает границу, то координаты точек пересечения определяются уравнениями

$$x_3 = x_1 + (y_{max} - y_1)\frac{(x_2 - x_1)}{y_2 - y_1} \tag{55}$$

По этой же схеме организуются и модули отсечения нижней, правой и левой границами рамки, только в соответствующих уравнениях используются другие параметры рамки, а координаты x и y могут меняться ролями. Каждый модуль работает независимо, но целесообразнее организовать из них конвейер. Такой конвейер, реализованный аппаратно, будет корректно обрабатывать четыре вершины.



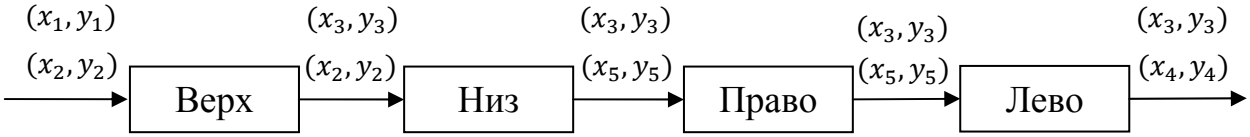


Рисунок 1.43. Объединение модулей отсечения в конвейер.

Ниже на рисунке показано, как с помощью такого аппаратного модуля реализуется отсечение многоугольника.

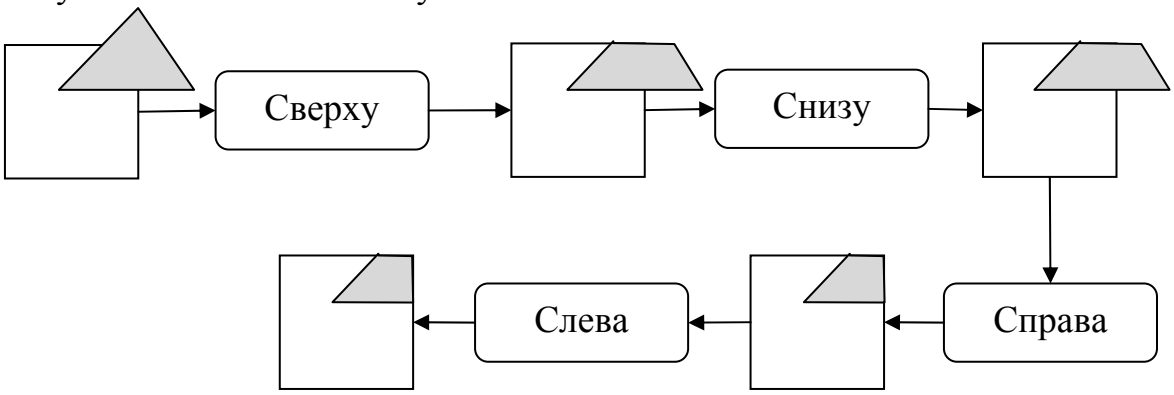


Рисунок 1.44. Конвейер отсечения многоугольника

1.4.4. Отсечение с применением прямоугольных оболочек

Предположим, что нам нужно обрабатывать многоугольники общего вида, имеющие довольно много вершин. Для этого можно воспользоваться одним из алгоритмов отсечения, рассмотренных выше, и применить его для обработки каждого из ребер многоугольника. Однако очень часто оказывается, что весь многоугольник лежит вне зоны видимости и что все усилия, затраченные на анализ его многочисленных ребер, пропали впустую.

Значительно сократить затраты поможет предварительный анализ, в котором используется прямоугольная оболочка (bounding box) сложного объекта, в нашем примере — многоугольника. Такая оболочка — это прямоугольник минимального размера со сторонами, параллельными границам рамки отсечения, в который вписывается анализируемый объект. Определение па-

раметров прямоугольной оболочки требует только последовательного просмотра всех вершин многоугольника и выявления максимального и минимального значений координат x и y.

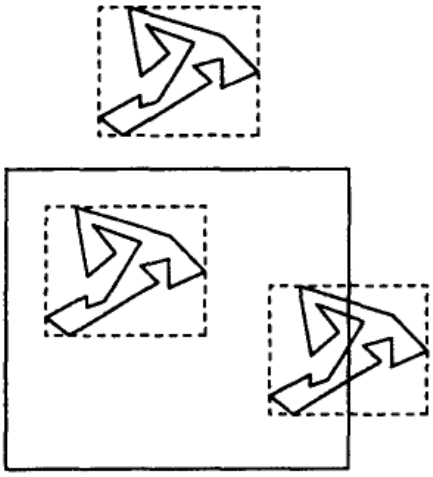


Рисунок 1.45. Отсечение при помощи прямоугольной оболочки

Далее можно применить алгоритм отсечения к оболочке, что реализуется значительно быстрее, поскольку, во-первых, оболочка имеет только четыре стороны, а во-вторых, эти стороны параллельны границам рамки.

Рассмотрим три варианта, представленные на рисунке. Тот многоугольник, который находится выше рамки, сразу отбрасывается, поскольку нижняя сторона его оболочки находится выше верхней границы рамки. Многоугольник, который расположен внутри рамки отсечения, также не имеет смысла «запускать» на конвейер отсечения, поскольку его оболочка полностью умещается в зоне видимости. Процедуру отсечения нужно выполнять только по отношению к третьему многоугольнику, так как его оболочка частично перекрывает зону видимости. Прямоугольная оболочка может использоваться и в трехмерном пространстве, причем она формируется еще на этапе моделирования и сохраняется вместе с описанием объекта.

Конечно, выполнение отсечения можно и отложить до этапа растрового преобразования и заполнения буфера кадра. Но целесообразнее каким-либо способом все-таки отсеять те объекты, которые явно не вписываются в заданную зону видимости, а на последнем этапе выполнять отсечение только тех из них, которые по самой своей природе являются растровыми, в частности блоков пикселей.

1.4.5. Отсечения иерархическими структурами

При работе с большими объемами данных весьма полезными могут оказаться различные древовидные (иерархические) структуры. Стандартными формами таких структур являются восьмеричные, тетрарные и BSP-деревья, а также деревья ограничивающих тел.

Одной из сравнительно простых структур является иерархия ограничивающих тел (Bounding Volume Hierarchy). Сначала ограничивающее тело

описывается вокруг всех объектов. На следующем шаге объекты разбиваются на несколько компактных групп, и вокруг каждой из них описывается свое ограничивающее тело. Далее каждая из групп снова разбивается на подгруппы, вокруг каждой из них строится ограничивающее тело и т. д. В результате получается дерево, корнем которого является тело, описанное вокруг всей сцены. Тела, построенные вокруг первичных групп, образуют первичных потомков, вокруг вторичных - вторичных и т. д.

Сравнения объектов начинаются с корня. Если сравнение не дает положительного ответа, то все тела можно сразу отбросить. В противном случае проверяются все его прямые потомки, и если какой-либо из них не дает положительного ответа, то все объекты, содержащиеся в нем, сразу же отбрасываются. При этом уже на ранней стадии проверок отсечение основного количества объектов происходит достаточно быстро, ценой всего лишь нескольких проверок.

Иерархические структуры можно строить и на основе разбиения пространства (картинной плоскости): каждая клетка исходного разбиения разбивается на части (которые, в свою очередь, также могут быть разбиты, и т. д. При этом каждая клетка разбиения соответствует узлу дерева).

Иерархии (как и разбиение пространства) позволяют достаточно легко и просто производить частичное упорядочение граней. В результате получается список граней, практически полностью упорядоченный, что дает возможность применить специальные методы сортировки.

1.4.6. Удаление невидимых граней

После того как вершины прошли через все этапы геометрических преобразований и процедуру отсечения, «на конвейере» остались только те геометрические объекты, которые потенциально могут попасть в формируемое изображение. Но прежде чем приступать к их растровому преобразованию, нужно решить еще одну задачу — удалить объекты, перекрываемые с точки зрения наблюдателя другими объектами.

Ниже будут описаны методы решения этой задачи для сцен, состоящих из полигональных объектов. Такое ограничение вводится по той простой причине, что в большинстве существующих на сегодняшний день графических систем все объекты, в конце концов, аппроксимируются множеством плоских многоугольников. Отрезки прямых также обрабатываются подобными методами после их незначительной модификации.

Анализ существующих на сегодняшний день алгоритмов удаления невидимых поверхностей позволяет выявить существенную разницу между подходами, ориентированными на отображаемые объекты и формируемое изображение. Особую роль в реализации этой процедуры играет возможность использовать предысторию анализа, когда результаты, полученные на предыдущем шаге, учитываются в последующих.

Пусть сцена состоит из k трехмерных непрозрачных многоугольников, каждый из которых можно считать независимым объектом. Обобщенный

подход, основанный на *анализе пространства объектов, предполагает* попарное сравнение положения всех объектов по отношению к наблюдателю (центру проецирования).

Рассмотрим два таких объекта: многоугольники A и B, изображенные на рисунке.

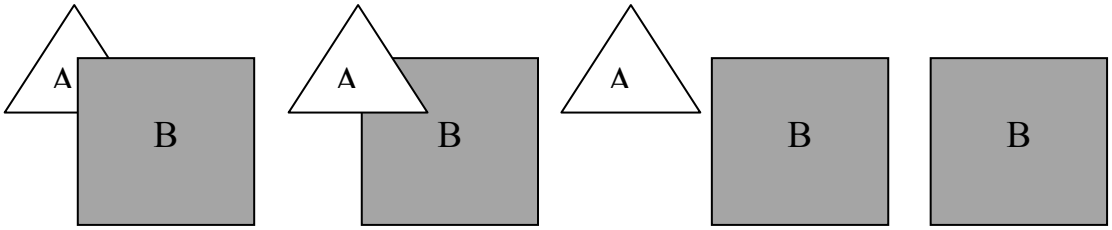


Рисунок 1.46. Два многоугольника: а — B частично закрывает A; б — A частично закрывает B; в — A и Вне перекрываются; г — B полностью закрывает A

Существует четыре варианта их взаимного положения:

- A полностью закрывает B — в изображение нужно включить только A;
- B полностью закрывает A — в изображение нужно включить только B;
- A и B не перекрываются — в изображение нужно включить оба объекта, A и B;
- A и B частично перекрываются — нужно вычислить, какая часть перекрываемого многоугольника будет видима наблюдателю.

Для того чтобы оценить сложность процесса, будем считать, что анализ, какой вариант сочетания многоугольников имеет место для текущей пары, и вычисление видимой части многоугольника представляют собой единую операцию. Будем обрабатывать множество имеющихся в сцене объектов итеративно. Берем один из k многоугольников и попарно сравниваем его с каждым из оставшихся $k-1$ многоугольников.

После выполнения этого цикла будет известно, какая часть выбранного многоугольника будет видима наблюдателю (если таковая вообще существует). Закончив с анализом одного многоугольника, перейдем к следующему, который точно так же будем попарно сравнивать с оставшимися к $-$ 2 многоугольниками.

Такой итерационный цикл продолжается до тех пор, пока не останутся только два многоугольника, которые мы и сравним в последнем итерационном цикле. Несложно показать, что оценка сложности всего процесса имеет вид $O(k^2)$. Следовательно, даже не вдаваясь в детали выполнения операций сравнения, можно прийти к выводу, что объектно-ориентированный подход лучше всего применять при отображении сцен, состоящих из относительно небольшого количества объектов.

Подход, основанный на анализе пространства изображения, имеет много общего с моделированием процесса визуализации с помощью приведения лучей.

Рассмотрим луч, исходящий из центра проецирования и проходящий через определенный пиксель на картинной плоскости. Этот луч где-то пересекается с плоскостью каждого из k многоугольников.

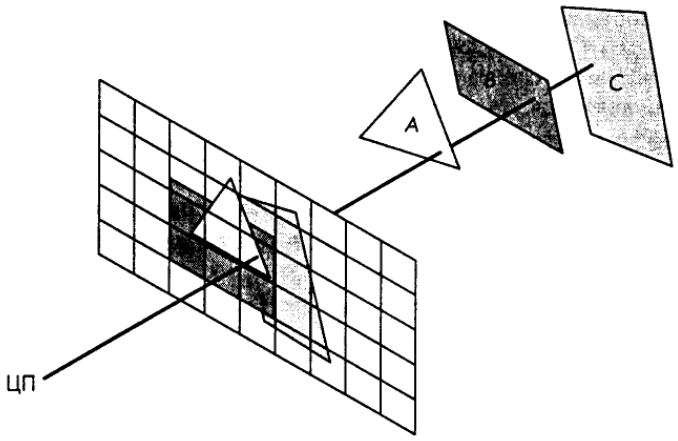


Рисунок 1.47. Удаление невидимых поверхностей в пространстве изображения

Мы можем вычислить координаты этих точек, выяснить, лежит ли точка пересечения во внутренней области многоугольника, и затем выбрать тот многоугольник, который первым пересекается этим лучом. Этот пиксель затем можно окрасить в цвет, который присвоен точке пересечения с выбранным многоугольником после выполнения тонирования. Если размер видового окна изображения $n \times m$ пикселей, то такая операция должна быть повторена nmk раз, т.е. оценка сложности имеет вид $O(k)$.

Полученная оценка является оценкой сверху, но в любом случае тот факт, что сложность процедуры при таком подходе пропорциональна количеству объектов, а не квадрату этого количества заставляет нас отдать предпочтение анализу видимости в пространстве изображения. Но следует учесть, что работа на уровне отдельных пикселей может привести к образованию зубцов на границах, разделяющих изображения отдельных объектов, что не свойственно алгоритмам, работающим в пространстве объектов.

1.4.7. Удаление нелицевых граней

Если сцена состоит только из выпуклых многогранников, то внутренние грани многоугольников никогда не будут "показываться на глаза" наблюдателю. Но у такого выпуклого многогранника часть граней будет повернута от наблюдателя (так называемые нелицевые грани) и, следовательно, тоже не будет видна наблюдателю. В конце концов, нелицевые грани будут перекрыты другими, и это сможет выявить универсальный алгоритм удаления невидимых поверхностей, но можно облегчить ему работу, сразу же отбраковав

нелицевые грани. Принцип отбраковки (culling) нелицевых граней поясняется на рисунке.

Внешняя сторона многоугольника будет видима наблюдателю только в том случае, если нормаль к плоскости многоугольника будет направлена к наблюдателю. Обозначим через θ угол между внешней нормалью плоскости многоугольника и направлением на наблюдателя, тогда многоугольник будет представлять лицевую грань объекта только в том случае, если -90° < θ < 90°, или, что то же самое, если $\cos\theta > 0$.

Второе условие анализируется проще, поскольку знак косинуса легко определяется по скалярному произведению и условие можно сформулировать следующим образом:

Можно еще упростить этот анализ, если учесть, что обычно он выполняется после преобразования в нормализованную систему координат устройства отображения. В этой системе координат все виды являются ортогональными, и, следовательно, направление на наблюдателя задается координатной осью z. Таким образом, в однородных координатах вектор v имеет вид $v = (0,0,1,0)^T$

Если уравнение плоскости, в которой лежит многоугольник, имеет вид

$$ax + by + cz + d = 0 \qquad (56)$$

то для определения, является ли грань лицевой или нелицевой, нам потребуется только проверить знак коэффициента c. Такую проверку можно выполнить как программно, так и аппаратно. При использовании отбраковки нелицевых граней нужно только тщательно проанализировать, допустимо ли это делать с учетом специфики конкретного приложения.

1.4.8. Алгоритм Z-буфера

Одним из самых распространенных алгоритмов удаления невидимых поверхностей является алгоритм z-буфера. Алгоритм довольно просто реализуется как программно, так и аппаратно, прекрасно сочетается с конвейерной архитектурой графической системы и может выполняться со скоростью, соответствующей скорости обработки вершин остальными модулями конвейера. Хотя алгоритм отталкивается от пространства изображения, он включает цикл просмотра многоугольников, а не пикселей и может быть реализован в процессе растрового преобразования.

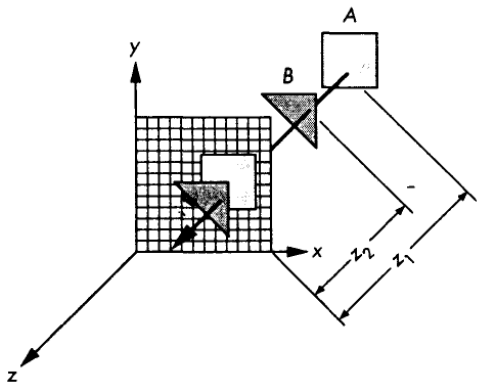


Рисунок 1.48. Алгоритм Z-буфера

Предположим, что выполняется растровое преобразование одного из двух многоугольников, показанных на рисунке. Используя принятую в графической системе модель закрашивания, можно вычислить цвет точек пересечения с каждым из многоугольников луча, исходящего из центра проецирования и проходящего через заданный пиксель картинной плоскости. Кроме того, одновременно нужно определить, является ли данная точка пересечения видимой наблюдателю,— из двух анализируемых видимой будет только точка, ближайшая к наблюдателю.

Следовательно, если преобразуется многоугольник B, его образ должен появиться на экране только в случае, если расстояние z_2 меньше расстояния z_1 до многоугольника A. И наоборот, если преобразуется многоугольник A, то его цвет не должен никак повлиять на формируемый цвет пикселя и изображение этого многоугольника в этой области экрана не должно появиться. Но есть небольшая загвоздка в реализации этой простой идеи — обработка многоугольников выполняется последовательно, а потому, преобразуя в растр один многоугольник, мы не располагаем информацией о том, как по отношению к нему расположены другие. Решается эта проблема с помощью промежуточного буфера, в который записывается информация о глубине размещения каждого многоугольника.

Предположим, что в нашем распоряжении имеется буфер — назовем его z-буфером,— который имеет такую же организацию, как и буфер кадра, а разрядность каждой ячейки достаточна для хранения информации о глубине с приемлемой для качества изображения точностью. Например, если формат изображения 1024x1280 пикселей и расстояние в графической системе представляется действительным числом с однократной точностью, то в z-буфере должно быть 1024x1280 32-разрядных ячеек.

Перед началом растрового преобразования объектов сцены в каждую ячейку заносится код, соответствующий максимальному расстоянию от центра проецирования. Буфер кадра в исходном состоянии заполняется кодами цвета фона.

В процессе растрового преобразования объектов каждая ячейка z-буфера содержит значение расстояния до ближайшего из обработанных ранее многоугольников вдоль проецирующего луча, проходящего через соответствующий пиксель пространства изображения.

Процесс заполнения z-буфера выглядит в первом приближении следующим образом. Все многоугольники в описании сцены последовательно, один за другим, подвергаются растровому преобразованию. Для каждой точки на многоугольнике, которая проецируется на определенный пиксель, вычисляется расстояние до центра проецирования. Это расстояние сравнивается с уже хранящимся в той ячейке z-буфера, которая соответствует формируемому пикселю.

Если расстояние до текущего многоугольника больше хранящегося в z-буфере, значит, ранее был обработан многоугольник, расположенный ближе к наблюдателю, который, таким образом, загораживает текущий. Следова-

тельно, данная точка текущего многоугольника не будет видима и информацию о ее цвете не нужно заносить в буфер кадра.

Если же вычисленное расстояние меньше того, что хранится в z-буфере, значит, текущий многоугольник сам загораживает те, что были обработаны ранее, и его код цвета должен заменить в буфере кадра код цвета, сформированный ранее. Одновременно вычисленное только что расстояние заносится в соответствующую ячейку z-буфера.

Объем дополнительных вычислений при использовании алгоритма z-буфера относительно невелик, поскольку в нем можно использовать предысторию обработки.

1.4.9. Сортировка по глубине

Метод сортировки по глубине (depth sort) реализует подход к решению проблемы удаления невидимых поверхностей, основанный на анализе пространства объектов. Мы рассмотрим этот метод применительно к сценам, состоящим из плоских многоугольников, но его можно распространить и на другие классы объектов. Метод сортировки по глубине является вариантом еще более простого алгоритма, получившего название алгоритма художника (painter's algorithm).

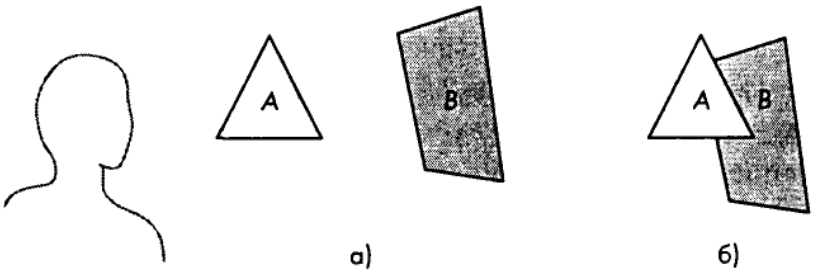


Рисунок 1.49. Алгоритм Художника. а — два многоугольника и наблюдатель, б — многоугольник А частично закрывает многоугольник В.

Предположим, что имеется набор многоугольников, отсортированных по расстоянию от них до наблюдателя. В примере, показанном на рисунке, имеются два многоугольника. С точки зрения наблюдателя они выглядят так, как показано на рисунке b),— один многоугольник частично закрывает от наблюдателя второй.

Для корректного отображения такой сцены нужно определить, какая часть дальнего многоугольника перекрыта тем, что расположен ближе, «вырезать» ее, а оставшуюся часть подвергнуть растровому преобразованию и вывести на экран. Но можно поступить и по-другому — некоторые считают, что именно так делает художник, нанося новый рисунок поверх старого.

Образы объектов можно формировать в порядке от наиболее удаленного к менее удаленным. В результате образы объектов, расположенных ближе к наблюдателю, автоматически перекроют образы объектов, расположенных дальше, и необходимость в каком-либо «искусственном» анализе перекрытия полностью отпадет.

Для такого метода придуман даже специальный термин — отображение многоугольников от дальнего к ближнему (back-to-front rendering) . Но реали-

зация описанного подхода требует решения двух проблем: как рассортировать многоугольники в пространстве сцены и что делать, если многоугольники пересекаются. Обе эти проблемы решаются методом сортировки по глубине, хотя для некоторых ситуаций можно разработать и более эффективные методы.

Предположим, что для каждого многоугольника уже вычислены параметры прямоугольной оболочки. Следующая операция — отсортировать все многоугольники по степени удаления от наблюдателя, причем в качестве основного параметра сортировки берется максимальное значение координаты z-оболочки.

Именно эта операция и дала название всему алгоритму — сортировка по глубине (depth sort). Предположим, что после сортировки многоугольники «выстроились» в порядке, показанном на рисунке.

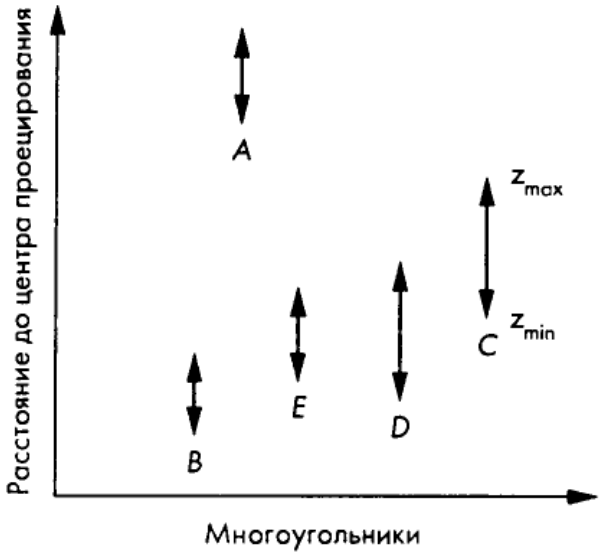


Рисунок 1.50. Z-оболочки сортируемых многоугольников

По оси ординат на этом рисунке отложены значения параметров z-оболочек. Если минимальная глубина (значение z_{min}) для данного многоугольника больше, чем максимальная глубина следующего за ним многоугольника, то после формирования их образов на экране в обратном порядке будет получено вполне корректное изображение сцены. Например, многоугольник A на рисунке расположен позади всех остальных, и его образ следует формировать первым.

Но, к сожалению, в других случаях дело обстоит гораздо сложнее, и одними значениями параметров z-оболочек не обойтись.

Если z-оболочки двух многоугольников перекрываются, то порядок формирования их образов выявляется индивидуальным сравнением каждой пары таких объектов. В алгоритме сортировки по глубине для этого используется множество довольно сложных проверок.

Рассмотрим пару многоугольников, z-оболочки которых перекрываются.

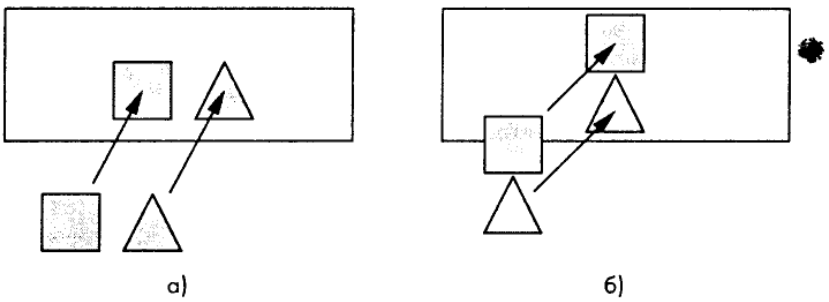


Рисунок 1.51. Анализ перекрытия оболочек

Первой выполняется самая простая проверка — сравнение x- и y-оболочек. Если либо x-, либо y-оболочки не перекрываются, многоугольники не закрывают друг друга сточки зрения наблюдателя и порядок формирования их образов не имеет значения. Но и в случае, если x- или y-оболочки перекрываются, можно отыскать порядок формирования их образов, который позволит корректно воспроизвести сцену на экране.

Остаются еще две ситуации, с которыми справиться особенно трудно.

Во-первых, три многоугольника могут перекрывать друг друга циклически, как показано на рисунке.

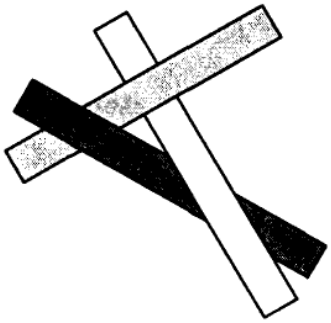


Рисунок 1.52. Циклическое перекрытие многоугольников

В таком случае вообще не существует такого порядка формирования образов многоугольников, который позволил бы получить корректное изображение. Лучшее, что можно предложить, — разделить по крайней мере один из них на две части и попытаться повторить анализ образовавшегося набора из четырех многоугольников.

Во-вторых, возможна ситуация, когда один многоугольник «пронзает» другой (или другие), как показано на рисунке.

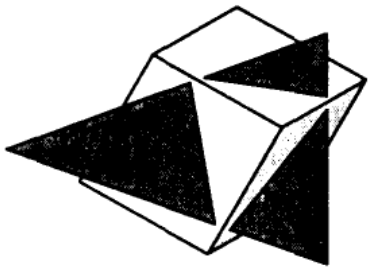


Рисунок 1.53. Многоугольник, «пронзающий» куб

В такой ситуации придется детально проанализировать пересечение, используя обобщенные методы отсечения одного многоугольника общего вида другим аналогичным многоугольником. Если пересекающиеся многоугольники имеют много вершин, можно попытаться использовать какие-либо искусственные методы, учитывающие специфику ситуации и требующие меньшего объема вычислений.

Оценить производительность метода сортировки по глубине очень сложно, поскольку многое зависит от специфики приложения. Например, работая с многоугольниками, представляющими собой грани сплошных объемных объектов, мы заранее знаем, что многоугольники не могут пересекаться. Но как бы там ни было, необходимость в предварительной сортировке явно указывает на то, что оценка производительности, как функции от сложности отображаемой сцены, не может быть линейной.

2. Методы реалистичного закрашивания

2.1. Освещение и текстурирование

Изучив материал предыдущих глав, вы уже можете формировать графические модели трехмерных объектов и выводить на экран их изображения. Но вряд ли вас устроит качество этого изображения — все объекты на нем кажутся плоскими, картинка не передает пространственного характера смоделированных объектов. Такое качество изображения объясняется тем, что при его формировании мы заливали каждую поверхность равномерно одним и тем же цветом. В результате ортогональная проекция сферы выглядит на экране как равномерно закрашенный круг, а куб — как плоский шестиугольник.

Фотография той же сферы выглядит совсем по-другому — на ней внутренняя область окружности, в которую проектировалась сфера, окрашена неравномерно, и в результате явно чувствуется выпуклая поверхность криволинейного геометрического тела. О таком изображении говорят, что цвет в нем имеет множество градаций или оттенков (shades). Именно таким способом передается объемность, пространственность объекта на плоском изображении.

В этом разделе мы попытаемся восполнить этот пробел. Будут рассмотрены по отдельности модели разных источников света и модели наиболее распространенных физических явлений, возникающих при взаимодействии света и материальной среды. Нашей целью является показать, как в уже описанную выше конвейерную архитектуру графической системы включить еще одну очень важную функцию — закрашивание отображаемой сцены с учетом характеристик источников света и материала моделируемых объектов.

2.1.1. Локальные модели

Мы сосредоточим внимание только на локальных моделях распределения света. Такие модели, в противоположность глобальным моделям освещения, позволяют вычислить оттенок отдельной точки на некоторой поверхности, независимо от других поверхностей как этого, так и других объектов сцены. В процессе вычислений учитываются только характеристики материала, ассоциированного с данной поверхностью, локальная геометрия этой поверхности, расположение и свойства источников света.

С точки зрения физики поверхность материального тела может либо излучать световую энергию, например поверхность электрической лампочки, либо отражать свет, падающий на нее от внешнего источника. Некоторые тела одновременно и отражают свет, и излучают его вследствие внутренних физических процессов, происходящих в материале. Когда мы смотрим на не-

которую точку материального объекта, то ее цвет определяется множеством элементарных актов взаимодействия со светом, падающим на объект как непосредственно от источников света, так и от других отражающих поверхностей. Последовательность этих элементарных актов можно представить в виде рекурсивного процесса.

Рассмотрим простую сцену, представленную на рисунке. Часть света от источника освещения, которая попадает на поверхность объекта А, отражается. Часть этого отраженного света попадает на поверхность объекта Б, причем часть этого отраженного света отражается и попадает вновь на объект А. Далее процесс повторяется.

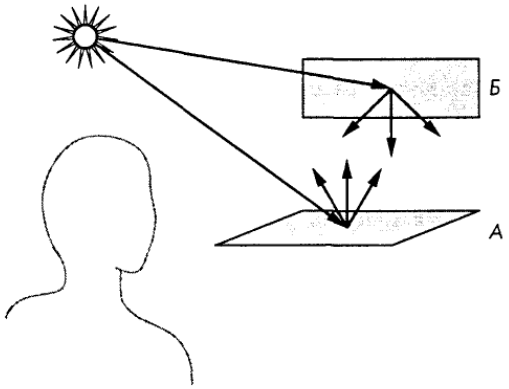


Рисунок 2.1. Отражающие поверхности.

Такое рекурсивное отражение света от двух поверхностей приводит к определенным цветовым эффектам, в частности появлению на поверхностях дополнительных окрашенных бликов. Математически этот рекурсивный процесс описывается интегральными уравнениями, которые называются глобальными уравнениями заполнения (rendering equation). В принципе, их можно было бы использовать для определения распределения цвета по всем поверхностям объектов сцены, но, к сожалению, такие интегральные уравнения в общем случае не поддаются решению даже с использованием численных методов.

Существует множество подходов, основанных на разных вариантах их аппроксимации (например, метод трассировки лучей или метод излучательности), каждый из которых позволяет найти решение уравнений для определенного типа объектов. Но и здесь имеется одно существенное препятствие— и метод трассировки лучей, и метод излучательности требуют большого объема вычислений, которые на существующей технологической базе не удается выполнить в реальном масштабе времени для сцен даже средней сложности.

Поэтому основное внимание мы уделим более простым локальным моделям заполнения, основанным на модели отражения Фонга (Phong), которая на сегодняшний день представляет собой вполне приемлемый компромисс между физической корректностью и объемом необходимых вычислений.

Вместо того чтобы рассматривать уравнения общего баланса световой энергии в сцене, эта модель анализирует световые лучи, испускаемые свето-излучающими поверхностями — источниками света (light sources), — и их взаимодействие с отражающими поверхностями объектов сцены. В чем-то такой подход сходен с трассировкой лучей, но в отличие от последнего метода он анализирует только один акт отражения от отдельной поверхности, а не рекурсивный процесс. Предлагается разделить проблему на две независимые части:

- в модель сцены нужно включить описания источников света
- сформировать модель процесса отражения, которая будет адекватно передавать взаимодействие материала поверхностей объектов сцены и света.

Для того чтобы представить себе этот процесс, попробуем проследить, как распространяются лучи от точечного источника света к сцене, представленной на рисунке.

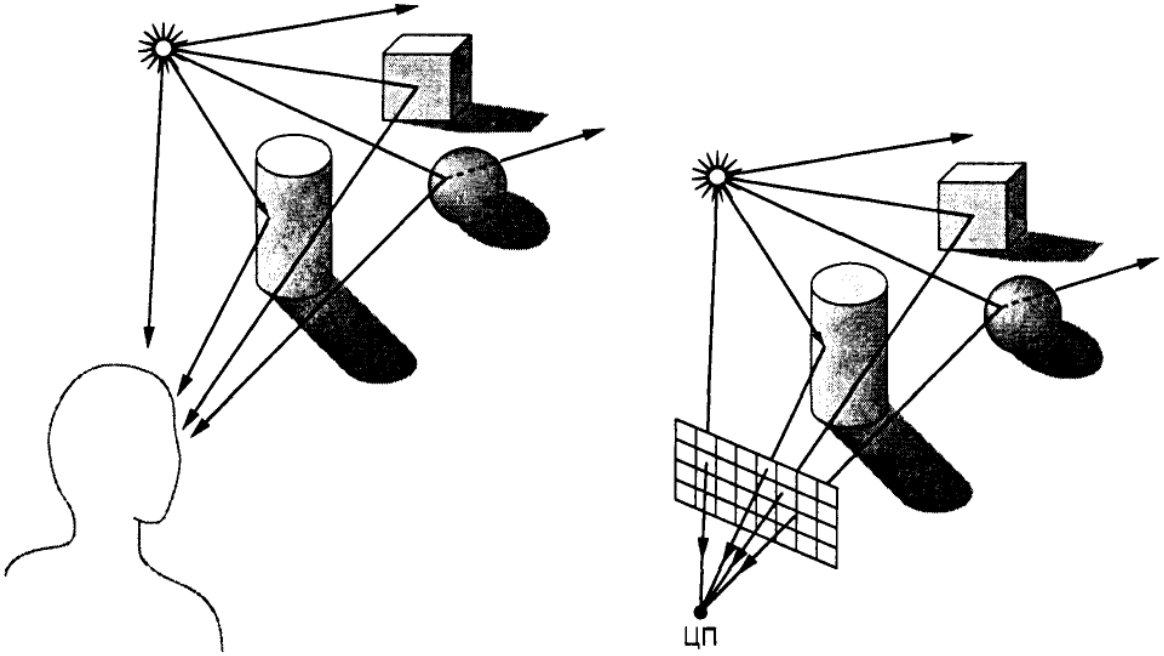


Рисунок 2.2. Световые лучи для человека и экрана компьютера.

Как отмечалось ранее, наблюдатель видит только те лучи, которые отразились от поверхностей объектов и достигли его глаз, «совершив», возможно, довольно сложное «путешествие» в пространстве сцены и «зацепив» по пути не один объект. Если луч попал в глаз непосредственно от источника, то наблюдатель будет воспринимать цвет света, испускаемого этим источником. Если луч по пути отразился от поверхности какого-либо объекта, то наблюдатель видит цвет, зависящий от характера взаимодействия света, падающего на поверхность, и материала поверхности.

В компьютерной графической модели в роли глаза наблюдателя выступает картинная плоскость, и изображение формируют только те лучи, кото-

рые достигли центра проецирования, пройдя внутри зоны, ограниченной рамкой отсечения картинной плоскости. Обратите внимание на то, что большинство лучей, испускаемых источником, не попадает на картинную плоскость, а потому не представляет для нас никакого интереса. Мы воспользуемся этим выводом позже.

Будем считать, что картинная плоскость разделена на прямоугольники, каждому из которых соответствует отдельный пиксель экрана, причем цвет пикселя зависит от цвета луча, попавшего на этот пиксель.

На рисунке показаны варианты и однократного, и многократного взаимодействия луча с объектами. Именно характер этого взаимодействия и определяет, что же увидит наблюдатель — красный объект или коричневый, темный или светлый, матовый или блестящий.

Когда луч попадает на поверхность, часть его энергии поглощается, а часть — отражается. Если поверхность непрозрачна, то вся энергия луча распределяется между процессами отражения и поглощения. Если же поверхность прозрачна, то часть энергии луча проходит через эту поверхность и может взаимодействовать с другими объектами. Характер взаимодействия зависит от длины световой волны. Объект, облученный белым светом, кажется наблюдателю красным, потому что большая часть энергии падающего света поглощена материалом объекта, а та, что отразилась, имеет максимум интенсивности в красной части спектра. Объект кажется наблюдателю блестящим, если его поверхность гладкая. И наоборот, объект, поверхность которого шероховата, кажется наблюдателю матовым.

Распределение теней на поверхности объекта зависит также от ориентации отдельных участков его поверхности, т.е. от направления вектора нормали в каждой точке этой поверхности.

Можно выделить три основных типа характера взаимодействия света и материала поверхности.

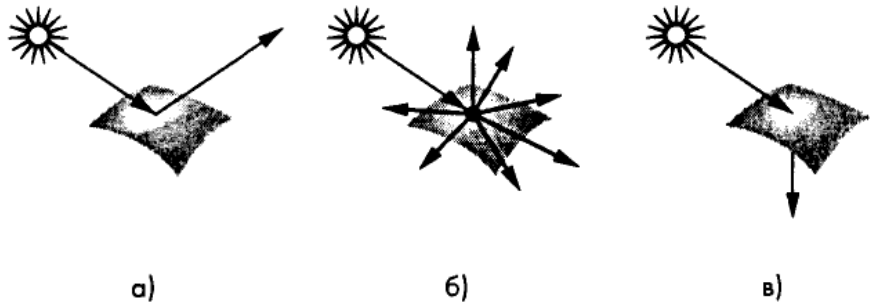


Рисунок 2.3. Взаимодействие света и материала: (а) — зеркальное отражение, (б) — диффузное отражение, (в) — преломление.

Три основных типа взаимодействия света с поверхностью можно описать следующим образом:

1. **Зеркальное отражение**. Поверхности, таким образом взаимодействующие с падающим на них светом, выглядят блестящими, поскольку большая часть световой энергии отражается или рассеивается в узком диапазоне углов, близких к углу отражения. Зеркало — это идеально отражающая поверхность. Хотя небольшая часть энергии падающего луча и поглощается, остальной свет отражается под одним углом, причем этот угол равен углу падения луча.
2. **Диффузное отражение**. При диффузном отражении падающий свет рассеивается в разных направлениях. Такой тип взаимодействия характерен для поверхности равномерно окрашенной стены или поверхности Земли, как она представляется пилоту самолета или космического летательного аппарата.
3. **Преломление**. В этом случае луч света, падающий на поверхность, преломляется и проникает в среду объекта под другим углом. Этот процесс рефракции характерен для стекла и воды. Как правило, при этом отражается часть падающего света.

Все указанные типы взаимодействия и описывающие их модели будут рассмотрены позднее. Для описания этих моделей необходимо сначала рассмотреть параметры самих источников света.

2.1.2. Источники света

Свет может исходить от поверхности объекта в двух случаях: либо объект является излучающим, либо объект отражает свет, падающий на него извне. Как правило, источником света мы считаем излучающие объекты, хотя, строго говоря, это и не так. Например, в ночных сценах источником может быть Луна, которая сама свет не излучает. Кроме того, некоторые объекты, которые излучают свет, одновременно и отражают свет, падающий на них от других источников, например колба электрической лампочки. Но в нашей упрощенной модели, не претендующей на абсолютную адекватность реальному физическому миру, мы не будем учитывать эти нюансы.

Рассматривая источник света, такой, как показан на рисунке, мы не должны забывать о том, что этот объект имеет определенную поверхность.

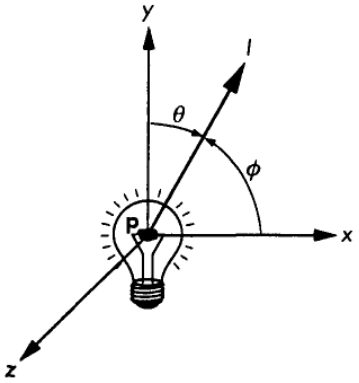


Рисунок 2.4. Источник света.

Каждая точка этой поверхности (x, y, z) может испускать световой луч, который характеризуется направлением эмиссии (θ, ϕ) и распределением световой энергии по длинам волн λ. Таким образом, элементарный источник света в общем случае характеризуется функцией излучения (illumination function), $I(x, y, z, \theta, \phi, \lambda)$, зависящей от шести параметров.

Следует обратить внимание на то, что направление излучения описывается двумя углами и что физический источник полихромного излучения представляется как множество независимых элементарных источников монохроматического света. Рассматривая освещаемую таким источником поверхность, можно получить освещенность каждой ее точки, интегрируя функцию излучения по поверхности источника света.

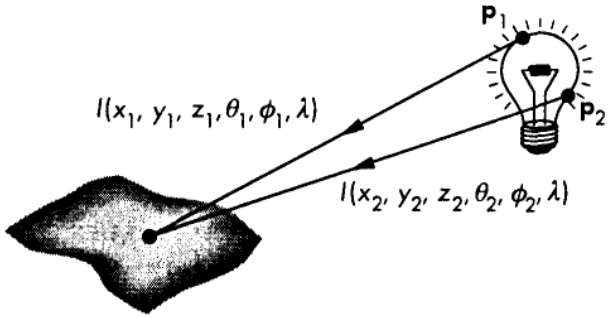


Рисунок 2.5. Суммирование света от разных источников

При интегрировании нужно учитывать, во-первых, только те углы излучения, которые обеспечивают попадание лучей на анализируемую точку освещаемой поверхности, а во-вторых, расстояние между элементарным источником и этой точкой. Для распределенного источника света, каковым является электрическая лампа, вычислить такой интеграл довольно сложно как аналитически, так и численно. Для упрощения подобный источник часто моделируется многоугольниками, каждый из которых представляет собой элементарный источник, или аппроксимацией реального источника множеством точечных.

Мы рассмотрим четыре основных типа источников света: фонового освещения (ambient lighting), точечные источники (point sources), прожекторы (spotlights) и удаленные источники света (distant light). Этих четырех типов вполне достаточно для моделирования освещения в большинстве простых сцен.

2.1.3. Цвет излучения

Создать адекватную модель источника довольно сложно, поскольку в ней нужно учитывать, что от длины волны зависит не только интенсивность, но и направление излучения. Однако мы с самого начала остановились на трехцветной модели зрения человека, которая предполагает, что для получе-

ния всей цветовой гаммы достаточно учитывать при анализе три основные (первичные) цветовые составляющие — красного, зеленого и синего цветов.

Поэтому в дальнейшем мы будем рассматривать любой источник как состоящий из трех независимых источников первичных цветов и соответственно описывать его трехкомпонентной функцией излучения:

$$I = (I_R, I_G, I_B)^T. \quad (57)$$

Следовательно, для вычисления распределения красного цвета в изображении будет использоваться «красный» компонент функции излучения. Поскольку вычисления для всех трех цветов выполняются по единой схеме, мы в дальнейшем для краткости будем рассматривать только по одному скалярному уравнению каждого типа, и будем считать, что его можно применять к любому из первичных цветов.

2.1.4. Фоновое освещение

В некоторых помещениях, например учебных аудиториях или в кухне, система освещения организуется таким образом, чтобы обеспечить равномерный свет по всему пространству. Чаще всего для этого используются источники большого размера с рассеивателями.

Смоделировать такую систему освещения можно, по крайней мере, в принципе, сформировав большое множество маленьких источников, направление излучения которых выбирается случайно, а затем проинтегрировав освещение некоторой точки от всех этих источников. Но при реализации такого подхода на практике придется выполнять массу вычислений, что не позволит графической системе формировать изображение в реальном масштабе времени.

Можно поступить и по-другому — подобрать такие характеристики источника, которые обеспечили бы равномерное освещение по всему пространству сцены. Такой источник принято называть источником фонового света (ambient light). Следуя второму подходу, мы можем при моделировании просто считать, что каждая точка на поверхности объектов этой сцены освещена одинаково. Таким образом, функция освещенности каждой точки поверхности характеризуется только заданной интенсивностью I_A.

Как уже отмечалось выше, будем считать, что источник фонового света состоит из трех источников первичных цветов, а его функция интенсивности — трехкомпонентная:

$$I_A = (I_{AR}, I_{AG}, I_{AB})^T. \quad (58)$$

Любой из компонентов функции I_A — красный, зеленый или синий — является скаляром I_{AX}, но в дальнейшем нужно учитывать, что хотя каждая точка на поверхности любого из объектов сцены получает от источника свет одинаковой интенсивности, отражают они его по-разному.

2.1.5. Точечный источник света

Идеальный точечный источник света (point source) излучает свет одинаково во всех направлениях. Такой источник, размещенный в точке p_0, характеризуется трехкомпонентным вектором цвета:

$$I(p_0) = \begin{pmatrix} I_R(p_0) \\ I_G(p_0) \\ I_B(p_0) \end{pmatrix} \tag{59}$$

Освещенность некоторой точки поверхности светом от этого источника обратно пропорциональна квадрату расстояния между этой точкой и источником.

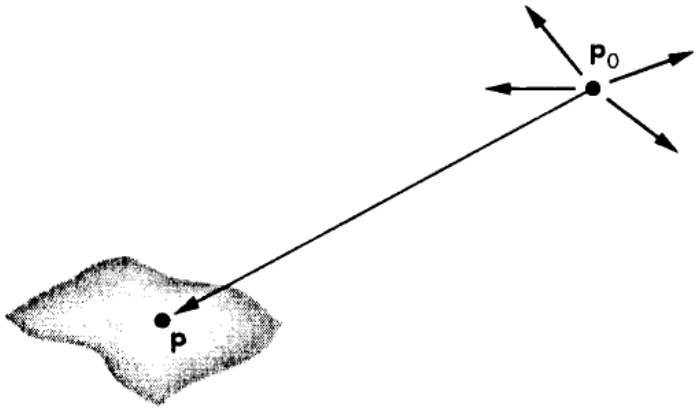


Рисунок 2.6. Освещение поверхности точечным источником света.

Следовательно, освещенность в точке p светом от некоторого точечного источника может быть представлена матрицей-столбцом:

$$I(p, p_0) = \frac{1}{|p - p_0|^2} I(p_0) \tag{60}$$

Использование точечных источников в большинстве приложении определяется скорее простотой работы с ними, чем желанием точно передать характеристики реальных физических осветительных приборов.

Изображение сцены, сформированное с учетом только точечных источников, получается очень контрастным (как говорят фотографы, жестким), все объекты оказываются либо очень яркими, либо слишком темными.

Реально каждый физический осветительный прибор имеет конечные размеры, что приводит к более плавному переходу от полностью светлых участков к полностью затененным, как показано на рисунке.

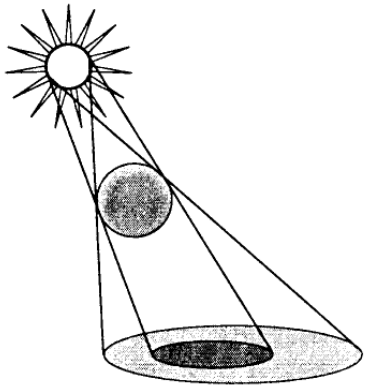


Рисунок 2.7. Формирование теневого перехода источником света, имеющим конечные размеры.

Эта зона перехода оказывается частично затененной — находится в полутени. В графической программе можно сымитировать снижение контраста от точечного источника, добавив источник фонового света (фотографы называют его источником заполняющего света).

Учет расстояния от конкретной точки до точечного источника света также вносит свою долю в излишнее «ужесточение» контраста. Хотя с чисто физической точки зрения обратно пропорциональная зависимость между освещенностью и расстоянием до источника вполне корректна, в компьютерной графике нередко используется модифицированная зависимость

$$\frac{1}{a + bd + cd^2} \tag{61}$$

где d — расстояние между p и p_0. Константы a, b и c выбираются из условия «смягчения» светотеневого перехода. Следует учитывать, что если точечный источник находится достаточно далеко от всех объектов сцены, то можно считать, что отличием в расстоянии до разных точек сиены можно смело пренебречь и что этот источник освещает их одинаково.

2.1.6. Прожекторы

Источники света типа прожектор (spotlight) отличаются тем, что испускают свет направленным пучком, то есть каждая точка излучающей поверхности посылает свет в одном и том же направлении. Проще всего смоделировать прожектор с помощью точечного источника света, ограничив для него направление, в котором распространяются световые лучи. Таким образом, формируется конус с вершиной в точке p_s, ось которого направлена вдоль вектора I_s, а угол наклона образующей к оси равен θ, как изображено на рисунке.

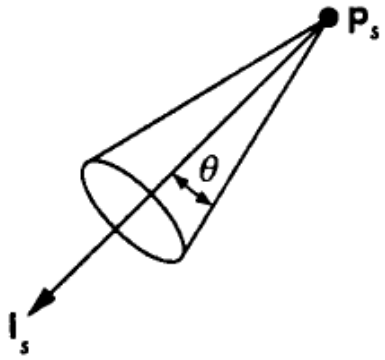


Рисунок 2.8. Источник освещения типа «прожектор»

Следует обратить внимание на то, что при $\theta = 180^0$ такой квазипрожектор превращается в точечный источник.

Модель прожектора, более близкая к реальному физическому прибору, характеризуется функцией распределения интенсивности в конусе излучения. Естественно, что наибольшей интенсивностью обладают лучи, направленные вдоль оси конуса. Интенсивность излучения прожектора является функцией от угла ϕ между осью конуса излучения прожектора и вектором s, направленным на определенную точку освещаемой поверхности, если этот угол меньше θ.

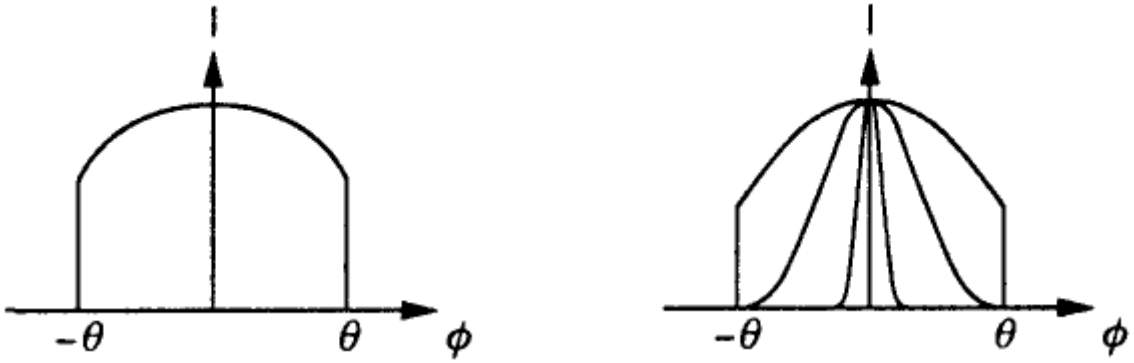


Рисунок 2.9. Простое и экспоненциальное распределение интенсивности в конусе излучения прожектора

Хотя функцию распределения интенсивности можно аппроксимировать по-разному, чаще всего ее задают в виде $\cos^e \phi$ где показатель e определяет, насколько быстро убывает интенсивность по мере увеличения угла ϕ, как приведено на втором рисунке. Как будет показано в дальнейшем, тригонометрическая функция косинус очень часто используется при математическом моделировании освещения. Если и s, и I являются векторами единичной длины (ортами), то значение косинуса можно вычислить с помощью скалярного произведения этих векторов:

$$\cos \phi = s \cdot I \tag{62}$$

Вычисление косинуса этим методом не требует никаких таблиц или разложения в ряд — достаточно выполнить три умножения и два сложения.

2.1.7. Удаленный источник света

Характерной особенностью удаленного источника света является то, что все испускаемые им лучи можно считать параллельными. Использование такого источника в сцене избавляет от необходимости рассчитывать направления лучей, освещающих разные точки отображаемой поверхности, а значит, существенно повышает скорость формирования изображения.

Прекрасным примером такого источника является солнце. На рисунке показано, что в этом случае можно заменить точечный источник света параллельным пучком лучей.

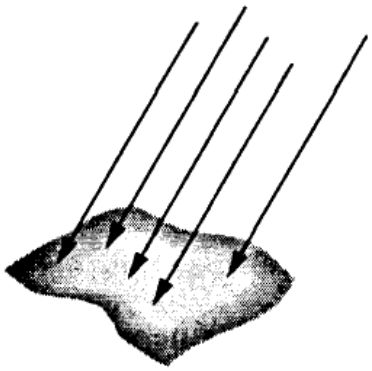


Рисунок 2.10. Параллельные лучи от удаленного источника света.

На практике обработка удаленного источника выполняется почти так же, как параллельное проецирование — вместо положения источника света учитывается направление его лучей. Следовательно, если использовать математический аппарат однородных координат, точечный источник света p_0 можно представить четырехмерной матрицей-столбцом $p_0 = (x, y, z, 1)^T$, а удаленный источник света — вектором направления в виде $p_0(x, y, z, 0)^T$.

В графической системе вычисления, связанные с обработкой уда ленного источника света, выполняются значительно быстрее, чел обработка близко расположенных точечных источников и прожекторов. Естественно, что одна и та же сцена, освещенная точечным источником или удаленным, выглядит совершенно по-разному. В составе практически всех реальных систем визуализации имеются средства представления и манипулирования как точечными, так и прожекторами и удаленными источниками света.

2.1.8. Модель отражения Фонга

Итак, перед нами стоит довольно сложная задача: с одной стороны, нам нужна модель взаимодействия света и материала, которая бы соответствовала реальным физическим процессам, а с другой стороны, реализация этой модели в графической системе должна хорошо встраиваться в конвейерную архитектуру и не слишком перегружать компьютер.

Ниже будет рассмотрена модель процесса отражения света объектами сцены, предложенная Фонгом (Phong). Практика использования этой модели во множестве графических систем доказала ее эффективность и достаточно близкое приближение к реальным физическим процессам, что позволяет формировать полутоновые изображения высокого качества при самых различных условиях освещения и свойствах материалов.

Для вычисления цвета произвольной точки p на поверхности объекта в модели используется четыре вектора, как приведено на рисунке.

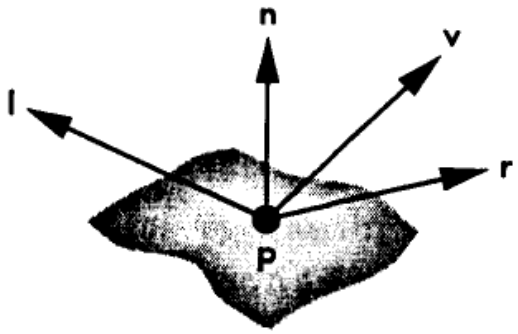


Рисунок 2.11. Векторы, используемые в модели Фонга

Если поверхность криволинейная (а это наиболее распространенный случай в реальном мире вещей), все четыре вектора могут изменяться при переходе от одной точки к другой. Вектор n является нормалью к поверхности в точке p. Вектор v направлен от точки p к наблюдателю или центру проецирования, а вектор l задает направление от точки p к произвольной точке источника распределенного света или в нашем случае — к точечному источнику света. И наконец, вектор r — это направление идеального отражения луча, падающего на поверхность вдоль вектора l. Следует учитывать, что вектор r однозначно определяется векторам n и l.

Модель Фонга адекватно передает три из четырех описанных выше типов взаимодействия света и материала — зеркальное и диффузное отражение и фоновое освещение. Предположим, что имеется множество точечных источников света. Можно считать, что для каждого из трех основных цветов свет от этого источника, влияющий на формируемое изображение, имеет три составляющие, которые назовем компонентами фонового (ambient), зеркального (specular) и диффузного (diffuse) света. Соответственно в математических выражениях компонент фонового цвета будет иметь индекс a, зеркального — индекс r, а диффузного — d.

С физической точки зрения такое предположение выглядит несколько странным, но наша цель состоит в том, чтобы, в конце концов, сформировать в графической системе световые эффекты, близкие к реальным, уложившись при этом в достаточно жесткие временные рамки. Поэтому для имитации эффектов, которые по самой своей природе носят глобальный характер, мы пытаемся использовать локальную модель. Итак, модель источников освещения включает компоненты фонового, диффузного и зеркального типов, и для каждой точки p отображаемой поверхности можно вычислить матрицу освещенности размером 3x3 для i-го источника света:

$$L_i = \begin{pmatrix} L_{ira} & L_{iga} & L_{iba} \\ L_{ird} & L_{igd} & L_{ibd} \\ L_{irs} & L_{igs} & L_{ibs} \end{pmatrix} \tag{63}$$

Элементы первой строки матрицы представляют интенсивности фонового света соответственно для красного, зеленого и синего цветов, поступающего от i-го источника. Элементы второй строки представляют цветовые компоненты интенсивности для диффузного света, а третьей — для зеркального. Предполагается, что в этих элементах не учитываются эффекты ослабления интенсивности освещения, связанные с расстоянием до источника.

Модель формируется в предположении, что существуют методы вычисления интенсивности света, отражаемого любой точкой поверхности, на основе значений элементов матрицы L, для этой точки. Например, зная значение диффузной составляющей L_{ird} красного цвета, поступающего от i-го источника, и определив каким-то способом коэффициент отражения R_{ird} для соответствующего компонента, можно вычислить значение составляющей интенсивности света, отраженного точкой p, в виде произведения $R_{ird}L_{ird}$.

Значение коэффициента отражения R_{ird} зависит от свойств материала поверхности, ее ориентации, направления на источник света и расстояния между источником и освещаемой точкой. Для каждой точки можно вычислить свой набор коэффициентов и объединить их в матрицу:

$$R_i = \begin{pmatrix} R_{ira} & R_{iga} & R_{iba} \\ R_{ird} & R_{igd} & R_{ibd} \\ R_{irs} & R_{igs} & R_{ibs} \end{pmatrix} \tag{64}$$

Для каждой цветовой составляющей света, приходящего от определенного источника, можно по этой же схеме вычислить и интенсивность отраженного света для фонового и зеркального компонентов, а затем все их сложить. Например, для i-го источника интенсивность составляющей красного цвета, отраженной от поверхности в точке p, равна

$$I_{ir} = R_{ira}L_{ira} + R_{ird}L_{ird} + R_{irs}L_{irs} = I_{ira} + I_{ird} + I_{irs} \tag{65}$$

Интегральную интенсивность отраженного света несложно найти, просуммировав составляющие от всех источников, имеющихся в сцене, и, в общем случае, глобального фонового освещения:

$$I_r = I_{ar} + \sum_i (I_{ira} + I_{ird} + I_{irs}) \tag{66}$$

Мы упростим в дальнейшем вид выражений, опустив индекс цвета, поскольку все сформулированные соотношения в равной степени справедливы для каждого из трех основных цветов. Но вычисление коэффициентов отражения для составляющих фонового, диффузного и зеркального света будет существенно отличаться. Отбросив индексы цвета, получим приведенное выше выражение в более компактном виде:

$$I = I_a + I_d + I_s = R_a L_a + R_d L_d + R_s L_s \tag{67}$$

Вычисления по приведенной формуле нужно выполнить для каждого основного цвета и каждого из имеющихся в сцене источников, затем просуммировать результаты по всем источникам и прибавить к ним член, зависящий от глобального фонового освещения.

2.1.9. Отражение фонового света

Интенсивность падающего на поверхность фонового света I_a одинакова во всех точках этой поверхности. Частично энергия этого света поглощается материалом поверхности, а частично — отражается. Отражение фонового света характеризуется коэффициентом $R_a = k_a$. Во внимание нужно принимать только положительные значения интенсивности отраженного света, то есть $0 \leq k_a \leq 1$. Следовательно, получим

$$I_a = k_a L_a \tag{68}$$

Здесь L_a может принимать индивидуальное значение для каждого из источников или быть единым для всех источников в сцене.

Стоит напомнить, что поверхность характеризуется тремя такими коэффициентами для каждого из основных цветов — k_{ar}, k_{ag}, k_{ab}, причем их значения могут отличаться. В результате изображение сферы будет выглядеть желтым, если коэффициент k_{ab}для синего цвета мал, а коэффициенты k_{ar} и k_{ag} для красного и зеленого цветов — велики.

2.1.10. Диффузное отражение

Поверхность с идеальным диффузным отражением отражает падающий на нее свет одинаково во всех направлениях. Следовательно, такая поверхность выглядит одинаково для всех наблюдателей. Однако количество отра-

женного света зависит от материала поверхности, поскольку часть энергии падающего светового потока поглощается.

Способность поверхности к диффузному отражению характеризуется ее шероховатостью. Если рассмотреть профиль такой поверхности при сильном увеличении, то, скорее всего, вы увидите картинку, подобную приведенной на рисунке.

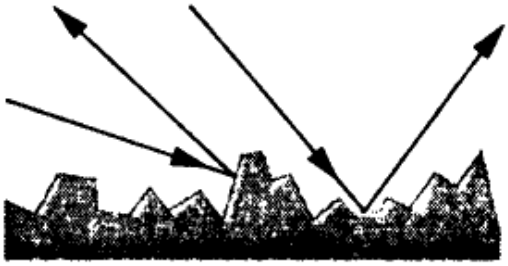


Рисунок 2.12. Шероховатая поверхность

Лучи света, падающие на поверхность под практически одинаковыми углами, чаще всего отражаются в различных направлениях. Поверхность с идеальным диффузным отражением настолько шероховата, что не существует какого-либо предпочтительного направления отраженных лучей, и они с равной вероятностью могут отразиться в любом направлении в верхнем по отношению к поверхности полупространстве. Такие поверхности иногда называют Ламбертовыми поверхностями (Lambertian surfaces) и математически характер отражения описывается законом Ламберта.

Рассмотрим плоскую шероховатую поверхность, которая освещается солнечными лучами. Самой яркой поверхность будет в полдень, а затем, ближе к вечеру, будет казаться все более темной, поскольку, в соответствии с законом Ламберта, для наблюдателя яркость поверхности определяется только вертикальной составляющей отраженного от нее света. Это явление можно представить себе следующим образом.

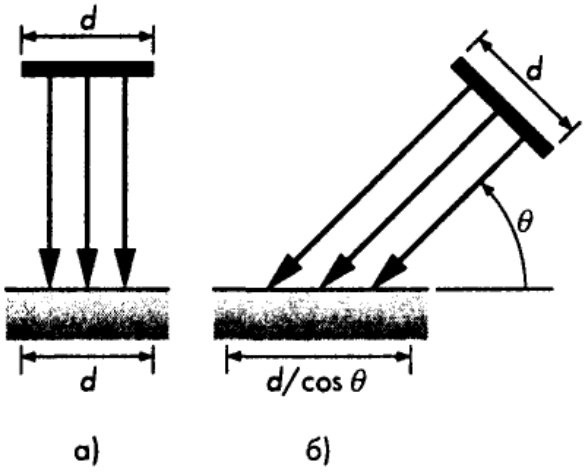


Рисунок 2.13. Графическая иллюстрация закона Ламберта

Рассмотрим, как отражаются от такой поверхности лучи от небольшого источника параллельного света. По мере того как источник света будет двигаться по воображаемому небосводу, наклоняясь все ниже и ниже, то же количество световой энергии будет распределяться по все большей площади, и поверхность будет восприниматься как более темная. Возвращаясь к точечному источнику, можно сформулировать математическую модель диффузного отражения.

Закон Ламберта утверждает, что

$$R_d \sim \cos\theta \tag{69}$$

Где знаком ~ обозначается пропорциональность, а θ — это угол между нормалью n к поверхности в анализируемой точке и направлением на источник света l. Если l и n являются единичными векторами, то

$$\cos\theta = l \cdot n \tag{70}$$

Введем в рассмотрение коэффициент пропорциональности k_d, который определяет, какая часть падающего на поверхность света отражается, и получим соотношение для составляющей диффузного отражения в общем отраженном световом потоке:

$$I_d = k_d(l \cdot n)L_d \tag{71}$$

В это соотношение можно ввести и множитель, зависящий от расстояния, который позволит учесть ослабление светового потока по мере удаления источника от анализируемой точки поверхности. Мы будем использовать множитель, имеющий в знаменателе квадратный трехчлен от расстояния d:

$$I_d = \frac{k_d}{a + bd + cd^2}(l \cdot n)L_d \tag{72}$$

Последняя формула и будет представлением диффузного освещения в модели Фонга.

2.1.11. Зеркальное отражение

Включив в модель только фоновое освещение и диффузное отражение, мы получим изображение, на котором уже будут видны тени и, таким образом, передана трехмерная форма объектов, но все поверхности будут выглядеть матовыми, как будто на них лежит слой пудры. Ни на одном объекте вы не увидите яркого пятна, которое в природе бывает на блестящих поверхностях. Обычно такое яркое пятно на объекте имеет цвет, отличный от того, который порождается фоновым светом или диффузным отражением. Например, на поверхности красного шара, освещенного источником белого света, вы

увидите яркое белое пятно, которое появилось в результате зеркального отражения части светового потока источника от поверхности шара в сторону наблюдателя.

В то время как диффузное отражение свойственно шероховатым поверхностям, зеркальное отражение появляется на гладких поверхностях. Чем более гладкой является поверхность, тем больше она напоминает по своим оптическим свойствам зеркало. На рисунке показано, почему по мере сглаживания поверхности все большая часть отраженного от нее светового потока концентрируется в определенном направлении, т.е. разброс углов отражения от различных точек поверхности все более сужается. Точное моделирование явления зеркального отражения — задача не простая, поскольку рассеивание света носит несимметричный характер и зависит от длины волны падающего света, а также меняется при изменении луча отражения.

Фонг предложил приближенную модель, в которой учет зеркального отражения очень незначительно увеличивает объем вычислений по сравнению с моделью, учитывающей только диффузное отражение. В этой модели вводится дополнительный аддитивный член, учитывающий зеркальное отражение. Таким образом, поверхность представляется шероховатой для составляющей диффузного отражения и гладкой — для составляющей зеркального отражения.

Интенсивность светового потока, который достигает наблюдателя, зависит от угла ϕ между вектором r, характеризующим направление идеального зеркального отражения, и вектором v, направленным к наблюдателю. В модели Фонга используется уравнение:

$$I_s = k_s L_s \cos^{\alpha} \phi \tag{73}$$

Коэффициент $k_s (0 \le k_s \le 1)$ определяет, какая часть светового потока отражается. Показатель степени α — это коэффициент резкости бликов (shininess coefficient). На рисунке показано, как при увеличении значения a отраженный свет концентрируется в зоне, близкой к углу идеального зеркального отражения.

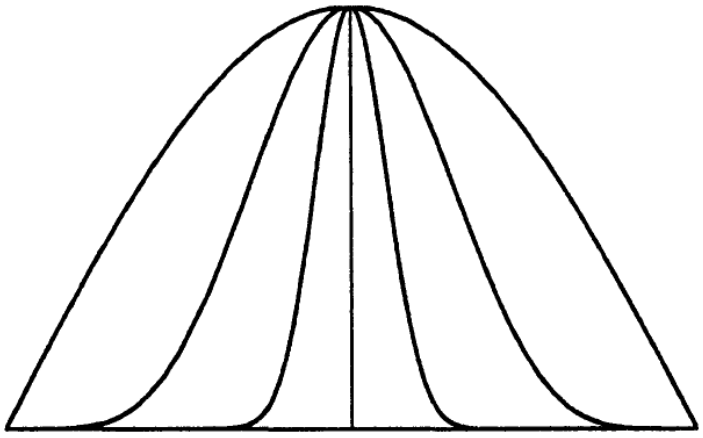


Рисунок 2.14. Зависимость резкости бликов для коэффициентов α

В пределе, при возрастании α до бесконечности, получим идеальное зеркало; значения в диапазоне от 100 до 500 соответствуют характеру отражения от большинства металлических поверхностей, а малые значения ($\alpha < 100$) соответствуют наиболее распространенным материалам с обычными оптическими свойствами.

Располагая данными о нормализованных векторах r и n, можно использовать в модели Фонга операцию скалярного произведения, и таким образом несколько сократить объем вычислений. В этом случае член, учитывающий зеркальное отражение, принимает вид

$$I_s = k_s L_s (r \cdot v)^{\alpha} \tag{74}$$

В это выражение можно добавить множитель, учитывающий расстояние до источника света, который имеет такой же вид, как и в выражении для диффузного отражения. В результате получим выражение для модели Фонга (Phong model), учитывающей это расстояние:

$$I = \frac{1}{a + bd + cd^2}(k_d L_d (l \cdot n) + k_s L_s (r \cdot v)^{\alpha}) + k_a L_a \tag{75}$$

Вычисления по этой формуле выполняются для каждого источника света, причем для всех трех основных цветов раздельно.

На первый взгляд кажется нецелесообразным связывать с каждым источников свою интенсивность фонового света или раздельно рассматривать составляющие диффузного и зеркального отражения. Но дело в том, что мы не можем позволить себе роскошь использовать в графической системе полную интегральную модель распределения освещения, а потому вынуждены прибегать к различным искусственным приемам, пытаясь получить результат, близкий к реальному.

Рассмотрим, например, сцену, в которой имеется множество объектов. Если включить в нее и источники света, то некоторые объекты будут освещены прямым светом от источников. Изображение таких объектов можно сформировать в графической системе, пользуясь составляющими зеркального и диффузного отражения. Но большая часть светового потока от источников достигнет объектов в результате многократного отражения.

Этот свет можно смоделировать в графической системе с помощью составляющей фонового освещения, специфической для каждого источника, причем цветовая характеристика этого фонового освещения должна учитывать как цветовую характеристику источника, так и цветовые характеристики множества объектов сцены. В этом и состоит особенность приближенной модели, в которой мы вынуждены усреднять реальные свойства компонентов.

В несколько расширенном виде такой же анализ можно провести и для составляющей диффузного отражения. Эта составляющая должна неявно учитывать тот факт, что в реальной обстановке рассеянный свет многократно

отражается от разных объектов и в результате его цвет может существенно измениться. В модели это можно отобразить, варьируя коэффициенты по основным цветам в составляющих диффузного и зеркального отражения, и таким образом свести все к локальному анализу отдельных поверхностей.

2.1.12. Закрашивание многоугольников

Располагая средствами вычисления векторов нормали, можно при заданном расположении источников света и наблюдателя применить рассмотренные модели ко всем точкам поверхностей объектов сцены. Но, к сожалению, использование уравнений вычисления нормали, которые мы рассмотрели ранее применительно к сферической поверхности, приводит к неприемлемо большим вычислительным затратам.

Раньше мы неоднократно подчеркивали достоинства полигональной модели для представления объектов отображаемой сцены. Использование этой модели для выполнения тонирования изображения сцены, состоящей из множества криволинейных объектов, существенно уменьшает объем вычислений. Именно такая модель, предполагающая аппроксимацию криволинейных поверхностей множеством маленьких плоских многоугольников, и используется в большинстве графических систем.

Рассмотрим полигональную сеть, подобную представленной на рисунке.

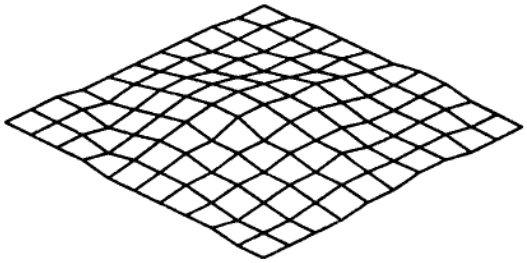


Рисунок 2.15. Полигональная сетка

Каждый многоугольник в такой сети — плоский, и вычислить компоненты вектора нормали к нему не представляет особого труда. Ниже мы рассмотрим три метода закрашивания многоугольников: плоское, интерполяционное, или закрашивание по методу Гуро (Gouraud), и закрашивание по методу Фонга.

2.1.13. Плоское закрашивание

При перемещении от одной точки на поверхности к другой в общем случае могут изменяться три вектора — l, n и v. Однако, если поверхность плоская, вектор n остается постоянным для всех точек этой поверхности. Если наблюдатель расположен достаточно далеко от этой поверхности, то изменением вектора v при переходе от точки к точке на поверхности плоского многоугольника небольшого размера также можно пренебречь и считать его постоянным. И, наконец, если в сцене используется удаленный источник све-

та, то вектор l также считается постоянным для всех точек поверхности, ограниченной закрашиваемым многоугольником. В данном случае термин «удаленный» имеет прямой смысл, т.е. считается, что источник бесконечно далеко удален от освещаемой поверхности.

Для реализации алгоритма закрашивания нужно в этом случае вместо расположения источника задавать направление на источник. Но термин «удаленный» можно рассматривать и в относительном смысле, сравнивая размеры закрашиваемого многоугольника с расстоянием до наблюдателя или до источника. Именно такая интерпретация этого термина используется в большинстве графических систем.

Если три указанных вектора постоянны для всех точек многоугольника, то все необходимые вычисления для его закрашивания можно выполнить только один раз и применить результаты ко всем точкам этого многоугольника. Этот метод получил название плоского (flat), или равномерного закрашивания (constant shading).

На изображении, сформированном алгоритмом плоского закрашивания, четко видна разница в оттенках цвета отдельных многоугольников сети, причем если источник света или наблюдатель расположены достаточно близко к объектам сцены, то это отличие порождается не только разной ориентацией многоугольников (значениями векторов нормалей n), но и разным направлением векторов l и v. Но при аппроксимации таким способом гладкой поверхности качество изображения может удовлетворить только самого непритязательного пользователя, поскольку переход между отдельными многоугольниками сети на изображении кажется очень резким.

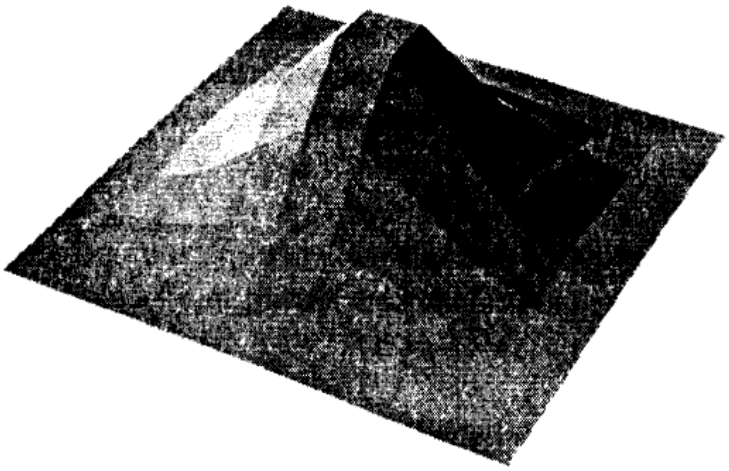


Рисунок 2.16. Плоское закрашивание полигональной сетки

Одна из особенностей зрительной системы человека заключается в том, что она очень чувствительна к малым изменениям оттенка соседних участков изображения вследствие так называемой вторичной задержки (lateral inhibition). Если рассматривать шкалу с дискретно изменяющейся интенсивностью участков, то каждый переход будет восприниматься таким образом, что с одной стороны перехода кажется, будто интенсивность участка понизилась по

сравнению с номинальной, а с другой стороны перехода участок вначале кажется более темным, чем номинальный.

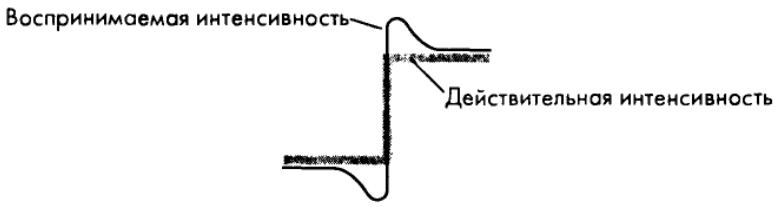


Рисунок 2.17. Различие между воспринимаемой и действительной интенсивностью в области перепада

Таким образом, в области светотеневого перехода глаз воспринимает полосы Маха (Mach bands). Это явление объясняется характером связи между колбочками и оптическим нервом, и единственное, что может сделать разработчик компьютерной графической системы для устранения такого неприятного эффекта — постараться по возможности сгладить контраст между оттенками соседних участков на границе многоугольников.

2.1.14. Закрашивание по методу Гуро

Возвращаясь к полигональной сети, отметим, что идея использования в вычислениях нормалей, ассоциированных с вершинами такой сети, должна вызвать у любого математика возражения, поскольку с математической точки зрения она абсолютно некорректна. Поскольку вершина является точкой пересечения, как минимум, двух по-разному ориентированных многоугольников, то в ней происходит разрыв непрерывности функции вектора нормали. Хотя такая ситуация и может серьезно осложнить математику алгоритма.

Гуро (Gouraud) пришел к выводу, что нормаль в точке вершины может быть определена таким способом, который позволит сгладить закрашивание. Рассмотрим одну из вершин в середине сети, в которой пересекаются четыре многоугольника.

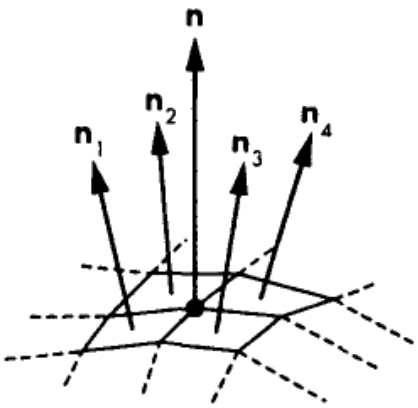


Рисунок 2.18. Определение нормали вершины полигональной сетки

Каждый многоугольник имеет свой вектор нормали. Метод закрашивания Гуро состоит в том, что с вершинами связываются нормали, которые получаются в результате усреднения нормалей многоугольников, Пересекающихся в этой вершине. Для примера, представленного на рисунке, с выделенной вершиной ассоциируется нормаль, вычисленная в соответствии с соотношением

$$n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|} \tag{76}$$

Далее, для вычисления освещенности каждой точки результирующей картинки вычисляется освещенность всех вершин той грани, на которой находится требуемая точка. Затем значения освещенности всех образующих вершин усредняется.

2.1.15. Закрашивание по методу Фонга

Но даже при использовании метода Гуро в ряде случаев не удается избежать появления на изображении полос Маха. Фонг предложил интерполировать не цвет точек от вершины к вершине, а направление нормалей последовательных точек на ребрах каждого многогранника.

Рассмотрим отдельный многоугольник полигональной сети, представленный на рисунке.

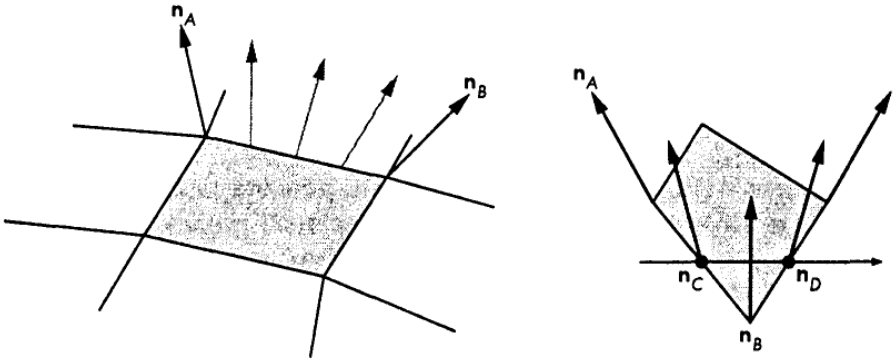


Рисунок 2.19. Интерполяция нормалей по методу Фонга

С каждой вершиной можно связать нормаль, которая формируется усреднением нормалей к граням, пересекающимся в этой вершине. Далее с помощью *билинейной* интерполяции, которую мы уже рассматривали ранее, можно интерполировать нормали по всей внутренней области многоугольника. Рассмотрим рисунок. Векторы нормалей в вершинах A и B используются для интерполяции вдоль ребра, связывающего эти две вершины:

$$n(\alpha) = (1 - \alpha)n_A + \alpha n_B \tag{77}$$

Аналогичную процедуру можно выполнить и для других ребер многоугольника. Затем можно вычислить нормаль в любой точке внутренней об-

ласти этого многоугольника, зная распределение нормалей на его ребрах. Получив вектор нормали в определенной точке, можно выполнить все необходимые вычисления, определяемые используемой моделью отражения. Обычно этот процесс реализуется вместе с растровым преобразованием многоугольника, в результате чего отрезок между точками проецируется на участок строки развертки в буфере кадра.

Метод Фонга позволяет получить более гладкое тонирование изображения, но за это приходится платить весьма изрядную цену — объем вычислений резко возрастает. Существует множество вариантов аппаратной реализации метода Гуро, которые позволяют получать изображение приемлемого качества, практически не увеличивая время закрашивания, чего не скажешь о методе Фонга. В результате в настоящее время метод Фонга используется только в тех системах, где не требуется формировать изображение в реальном масштабе времени.

2.1.16. Текстурирование

Текстурирование (mapping) применяется в процессе тонирования изображения и позволяет создавать иллюзию поверхности со сложной текстурой, хотя в геометрической модели эта поверхность может быть представлена самым заурядным многоугольником. Методы наложения основаны на использовании массива пикселей который определяет, как модифицируются параметры алгоритма тонирования, чтобы создать иллюзию сложной рельефной поверхности. Далее будут рассмотрены средства, позволяющие прикладной программе модифицировать отдельные пиксели в буфере, виды дискретных буферов и их назначение, методы доступа к ним.

Алгоритм наложения проективной текстуры (texture mapping) использует некоторый шаблон (или текстуру) для формирования цвета фрагмента. Текстура может иметь регулярный характер и храниться как фиксированный массив — именно такого типа текстуры часто используются для заполнения внутренних областей многоугольников. Иногда применяются методы динамического формирования текстур или в качестве текстуры используется внешнее изображение. В любом случае можно считать, что наложение проективной текстуры на поверхность, как это показано на рисунке, является частью процесса закрашивания этой поверхности.

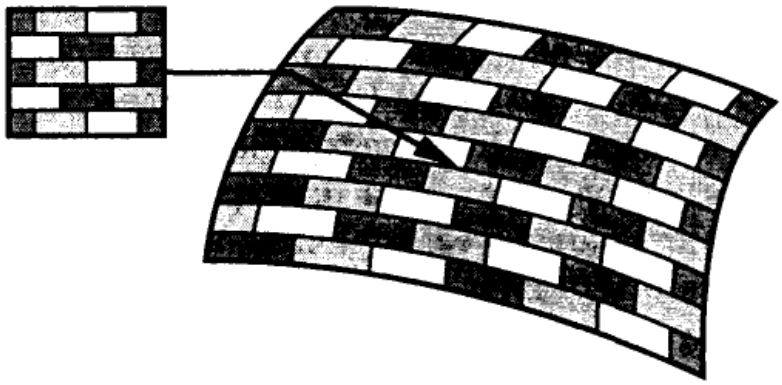


Рисунок 2.20. Наложение текстуры на поверхность

Наложение проективных текстур позволяет «проработать» детали образа гладкой поверхности. Другой метод — наложение микрорельефа (bump mapping) — позволяет сделать поверхность менее гладкой, наложив на нее «пупырышки» — карту микрорельефа (bump map), и превратить баскетбольный мяч в некое подобие апельсина. Наложение карт отражения (reflection maps) или карт среды (environmental maps) позволяет сформировать изображение, напоминающее то, которое создается при трассировке лучей, хотя сама процедура трассировки и не выполняется. В этом случае изображение окружающих предметов (среды) накладывается на поверхность, и создается иллюзия зеркального отражения.

Три перечисленные группы методов имеют много общего. Все они модифицируют результат тонирования, никак не затрагивая форму объекта (точнее, его геометрическую модель). Все методы реализуются синхронно с тонированием как часть заключительного этапа этого процесса. Все методы базируются на некоторых образцах — картах, которые хранятся в виде одно-, двух- или трехмерного дискретизированного изображения. Все методы требуют принятия специальных мер для сглаживания ступенек или зубцов на границах областей.

2.1.17. Проективные текстуры

Образцы текстур могут иметь самый разный вид — от полос и клеток до сложных изображений, воспроизводящих срез натуральных материалов, таких как мрамор, гранит, дерево. Очень часто мы распознаем материал реального объекта именно по его текстуре. Поэтому, если перед разработчиком графического приложения стоит задача создать иллюзию натурального объекта, ему никак не обойтись без наложения на объект соответствующей текстуры. В этом разделе мы рассмотрим только работу с двухмерными текстурами, хотя используемые при этом методы вполне применимы и к одно-, трех- и четырехмерным текстурам.

Процесс наложения проективных текстур выполняется в несколько этапов. Перед началом процедуры мы располагаем двухмерным образцом (шаблоном) текстуры — функцией интенсивности $T(s,t)$. Независимые перемен-

ные s и t называются координатами текстуры (texture coordinates). Сейчас можно считать образец текстуры массивом бесконечного (или, по крайней мере, очень большого) размера, хотя реально он, естественно, хранится в памяти конечного объема из $n \times m$ ячеек. Элементы массива образца текстуры иногда по аналогии с пикселями называют текселями (texels). Не теряя общности, можно изменить масштаб представления координат текстуры и выйти за диапазон $(0, 1)$.

Карта наложения (texture map) ассоциирует с каждой точкой геометрического объекта интенсивность T соответствующей точки образца, причем точки поверхности, в свою очередь, отображаются на пространство координат экрана. Если объект представлен в пространственных (геометрических) координатах ((x, y, z) или (x, y, z, w)), можно далее рассуждать в терминах математической функции отображения пространства координат текстуры на пространство геометрических координат и функции проективного преобразования, отображающей пространство геометрических координат на пространство координат экрана.

Если описать геометрический объект в параметрической форме в виде функции параметров (u, v), то в этой цепочке отражений появится еще одно звено. В этом случае нужно учитывать существование двух параллельных процессов отражения: первый отражает координаты текстуры на геометрические координаты, а второй — параметрические координаты на геометрические, как это схематически показано на рисунке. Третья функция отражения переносит нас в пространство координат экрана.

Если не вдаваться в подробности, то процесс наложения текстуры выглядит достаточно простым. Малая область образца отображается на область геометрической поверхности, соответствующую одному пикселю окончательного изображения. Предполагается, что значения интенсивности образца T представляют собой цветовой код RGB и их можно использовать либо для модификации цвета поверхности, определяемого из других соображений (источники света, отражение и т.п.), либо для назначения поверхности цвета только на основании образца текстуры. Этот процесс выполняется синхронно с алгоритмом тонирования или закрашивания поверхности.

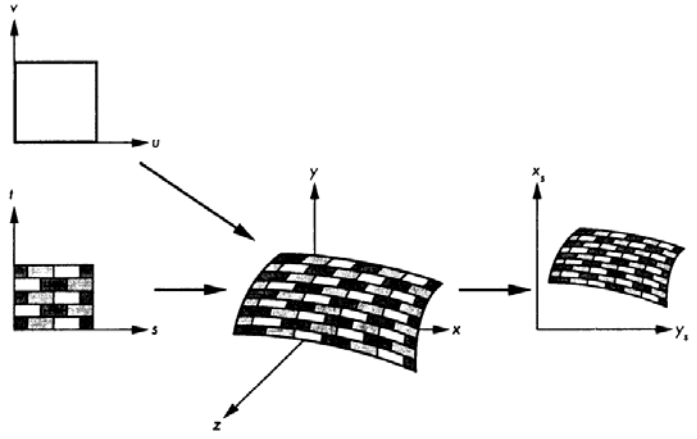


Рисунок 2.21. Карты наложения при параметрическом задании поверхности

Но при более внимательном взгляде обнаруживается множество «подводных камней». Во-первых, нужно определиться с методом сопоставления координат текстуры с геометрическими координатами. Двухмерный образец обычно определяется на прямоугольной области в пространстве текстуры. Функция отображения этого прямоугольника на область произвольной формы в трехмерном пространстве может быть достаточно сложной или обладать нежелательными свойствами. Например, если желательно отобразить прямоугольную область на сферу, это невозможно сделать, не искажая форму образца или пропорции его компонентов (отношения между расстояниями).

Во-вторых, сама организация процесса тонирования (он выполняется последовательным перебором пикселей) заставляет нас интересоваться скорее обратной функцией отображения, т.е. отображением координат экрана на координаты текстуры. Нас интересует при этом, какая точка на образце соответствует определенному пикселю образца поверхности, а это требует пересчета координат экрана в координаты текстуры.

В-третьих, поскольку мы вычисляем коды засветки пикселей, каждый из которых определяет цвет элементарной прямоугольной области изображения, то нас интересует не функция отображения точки одного пространства на точку другого, а области в одном пространстве на область в другом. Здесь мы вновь сталкиваемся с проблемой сглаживания границ между областями, которой следует уделить особое внимание. В противном случае появляются совершенно неожиданные эффекты типа муара.

Теперь на время оставим задачу определения функции отображения и обратим взор на метод определения засветки пикселей. Одно из возможных решений — использовать точку, полученную при обратном отображении центра пикселя, и извлечь соответствующее значение функции окраски образца. Этот метод прост, но приводит к появлению на изображении дефектов квантования, которые особенно заметны в том случае, когда изображение образца носит периодический характер. На рисунке показано, в чем причина подобных искажений.

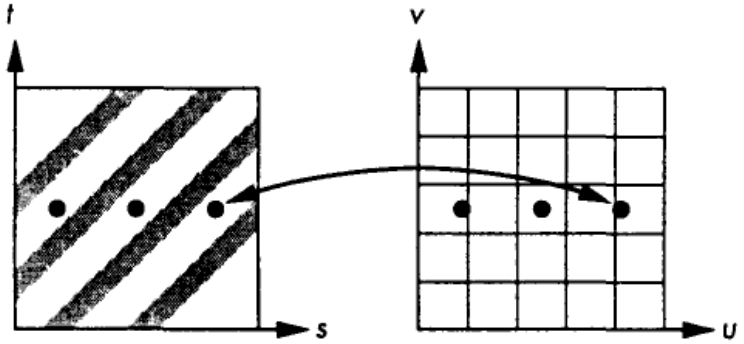


Рисунок 2.21. Эффект дискретизации при наложении текстуры

На этом рисунке слева можно видеть регулярную текстуру в виде полос, которую нужно наложить на плоскую поверхность. Обратная проекция цен-

тра каждого пикселя попадает как раз в промежуток между темными полосами, и в результате вся поверхность будет светлой. В общем случае, если не принимать во внимание конечные размеры пикселя, на изображении появится муар. Лучшие результаты можно получить другим методом (но его сложнее реализовать) — формировать возвращаемое значение интенсивности образца, усредняя значения интенсивности T по области, обратной проекции пикселя. Этот метод дает лучший результат, но и он не идеален.

Например, на рисунке показано, что происходит при его применении. На сей раз мы опять не получим на изображении полос, поскольку оно будет иметь усредненный цвет — ни светлый, ни темный. Таким образом, нам и на этот раз не удалось избавиться от дефектов, связанных с ограниченным разрешением буфера кадра и образца. Наиболее заметны эти дефекты при регулярной структуре образца текстуры.

2.2. Растеризация и наложения

Теперь перейдем к анализу методов выполнения завершающей стадии создания изображения — алгоритмам растрового преобразования примитивов и заполнения буфера кадра. Мы детально рассмотрим только два вида примитивов — отрезки и многоугольники, причем и те и другие заданы своими вершинами. Считаем, что все необходимые геометрические преобразования, в том числе и отсечение, уже выполнены и образы всех оставшихся примитивов должны включаться в конечное изображение сцены. Чтобы не отвлекаться, пренебрежем необходимостью удаления невидимых поверхностей и преобразуем каждый объект индивидуально, не используя информацию о его пространственных отношениях с другими объектами сцены. Будем также считать, что в результате проективного преобразования все вершины представлены только двумя координатами, причем в системе координат экрана.

2.2.1. Растровое преобразование

Рассмотрим такую организацию буфера кадра: массив из $n \times m$ элементов, каждый из которых соответствует одному пикселю, причем элемент с адресом в массиве $(0, 0)$ соответствует пикселю, расположенному в левом нижнем углу экрана. Для установки определенного кода цвета пикселя используется единственная функция графической системы, формат вызова которой имеет вид `setpixel( x, y, color )`.

Аргумент value обычно представляет собой закодированный цвет, требуемый для выставления. Адрес в буфере кадра, естественно, является дискретной величиной, но экранные координаты могут быть представлены и действительными числами. Мы вправе говорить о том, что точка имеет координаты на экране, например (63.4, 157.9), но центр ближайшего к ней пикселя находится в точке (63, 158) или (63.5, 157.5). В разных устройствах отображения пиксели могут иметь разную форму и размеры, но сейчас будем

считать, что все они являются квадратиками, причем координатами пикселя является центр этого квадратика, а размер в точности равен шагу размещения пикселей на экране. Будем также считать, что извлечение информации из буфера кадра производится другим процессом параллельно с его заполнением модулем растрового преобразования. Такое предположение, которое справедливо для большинства графических систем, использующих так называемую двухпортовую память, избавляет нас от необходимости обращать внимание на детали вывода изображения из буфера на экран.

Простейший алгоритм растрового преобразования линейного отрезка известен как алгоритм ЦДА (DDA algorithm). Этот алгоритм «позаимствован» у специализированных вычислителей — цифровых дифференциальных анализаторов, — которые в прежние времена использовались для численного решения систем дифференциальных уравнений. Поскольку прямая описывается дифференциальным уравнением вида $dy/dx = m$, где m — коэффициент наклона, формирование прямолинейного отрезка есть не что иное, как решение этого простейшего уравнения численным методом.

Пусть отрезок задан крайними точками (x_1, y_1) и (x_2, y_2).

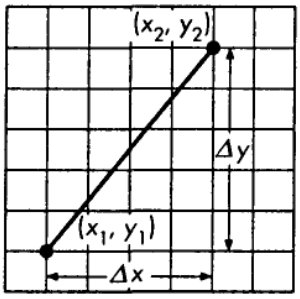


Рисунок 2.22. Прямолинейный отрезок в системе координат экрана

Будем считать, что значения координат округлены до ближайшего целого, т.е. отрезок начинается и заканчивается точно в центре соответствующего пикселя. Коэффициент наклона отрезка определяется соотношением:

$$m = \frac{y_2 - y_1}{x_2 - y_1} = \frac{\Delta y}{\Delta x} \tag{78}$$

Также будем считать, что $0 < m < 1$. При других вариантах значения m можно использовать симметричную модификацию алгоритма. В основе алгоритма лежит запись кода цвета в ячейку буфера, представляющую определенный пиксель, для каждого значения ix с помощью функции setpixel, по мере того как x изменяется от x_1 до x_2. Между элементарными приращениями δx и δy координат отображающей точки, которая «перемещается» от одной конечной точки отрезка к другой, существует связь:

$$\delta y = m\, \delta x \tag{79}$$

Если на каждом шаге увеличивать значение x на 1, т.е. принять $\delta x = 1$, то координату y отображающей точки нужно увеличить на величину $\delta y = m$. Хотя каждое очередное значение x представляет целое число, очередное значение y таковым не является, поскольку m в общем случае — дробь. Поэтому значение y нужно округлить и найти таким образом подходящий пиксель, представляющий текущее положение отображающей точки.

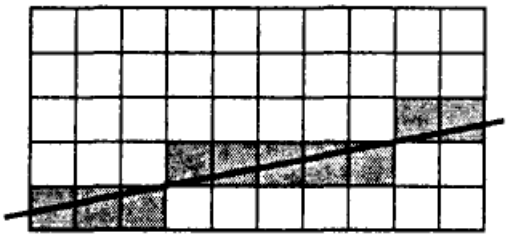


Рисунок 2.23. Формирование линии при использовании алгоритма ЦДА

В результате наш алгоритм будет выглядеть следующим образом:

```
for ( ix = x1, ix <= x2, ix++ )
{
    y += m ;
    set_pixel ( x, round(y), color );
}
```

Здесь функция round () выполняет округление действительного числа до ближайшего целого. Причина, по которой мы ограничили значение коэффициента наклона единицей, становится ясной при взгляде на рисунок.

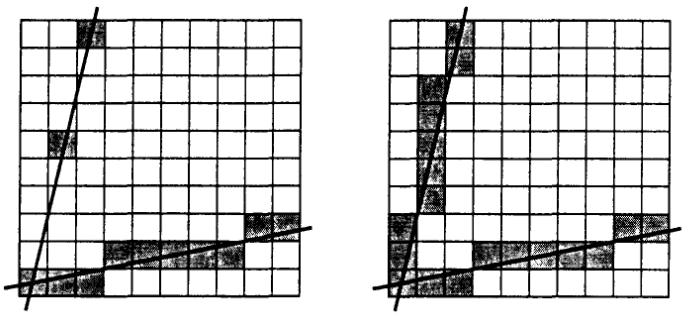


Рисунок 2.24. Искажение при больших углах наклона и симметричная модификация алгоритма.

Тот алгоритм, который мы сейчас рассматриваем, отыскивает значение координаты y отображающей точки, задаваясь приращениями ее координаты x. При больших значениях коэффициента наклона координаты x и y должны «поменяться ролями». Задаваясь значениями координаты y отображающей точки, будем определять подходящее значение координаты x. Заодно обращаю ваше внимание на то, что использование такой симметричной модификации алгоритма устраняет потенциальные проблемы с горизонтальными и вертикальными отрезками.

Поскольку прямолинейный отрезок определяется вершинами, то при назначении кода цвета каждому очередному пикселю можно также использовать интерполяцию, но не пространственных координат, а «цветовых». Кроме того, в процессе интерполяции отрезка можно, управляя цветом, накладывать на образ отрезка штрихи, точки и т.п. Все эти дополнительные эффекты не рассматриваются как часть алгоритма растрового преобразования, но часто реализуются синхронно с ним.

2.2.2. Алгоритм Брезенхема

Описанный алгоритм ЦДА при всей его простоте и наглядности обладает одним недостатком — при формировании координат каждой отображающей точки необходимо выполнять сложение действительных чисел. Более простой целочисленный алгоритм предложил Брезенхэм (Bresenham) в 1965 году, и с тех пор этот алгоритм используется практически во всех графических системах, причем существуют как программные, так и аппаратные его реализации.

Как и при анализе алгоритма ЦДА, будем считать, что конечные точки отрезка представлены целочисленными координатами (x_1, y_1) и (x_2, y_2), а коэффициент наклона удовлетворяет условию $0 < m < 1$.

Это условие достаточно критично для работоспособности алгоритма. Суть алгоритма в том, что при построении растрового образа отрезка выбирается пиксель, ближайший по вертикали к соответствующей прямой. Рассмотрим промежуточный шаг растрового преобразования отрезка после того, как будет сформирован пиксель $(i + \frac{1}{2}, j + \frac{1}{2})$

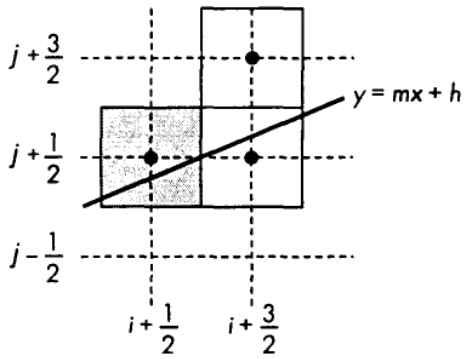


Рисунок 2.25. Алгоритм Брезенхема.

Известно, что прямая описывается уравнением $y = mx + h$. При $x = i + \frac{1}{2}$ прямая проходит от центра пикселя — точки с координатами $(i + \frac{1}{2}, j + \frac{1}{2})$ — не далее, чем на полшага между пикселями. В противном случае образ прямой вследствие округления не прошел бы через этот пиксель. При переходе к следующей точке, для которой $x = i + 3/2$, учитывая ограничения, наложенные на значений коэффициента наклона, образ отрезка

должен пройти либо через пиксель с центром в $\left(i + \frac{3}{2}, j + \frac{1}{2}\right)$, либо через пиксель с центром $\left(i + \frac{3}{2}, j + \frac{3}{2}\right)$.

Проблема выбора решается с помощью характеристической переменной $d = b - a$, где a и b — это расстояния между прямой в точке $x = i + \frac{3}{2}$ и верхним и нижним пикселями-кандидатами.

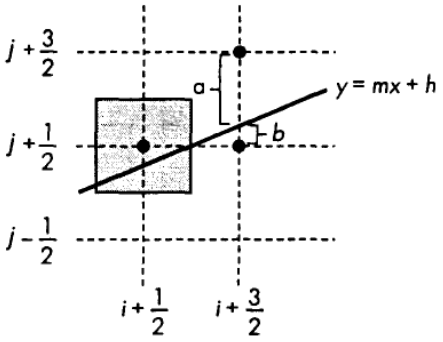


Рисунок 2.26. Характеристическая переменная алгоритма Брезенхема

Если *d* отрицательно, прямая проходит ближе к центру нижнего пикселя и в качестве следующего пикселя образа отрезка выбирается в $\left(i + \frac{3}{2}, j + \frac{1}{2}\right)$, в противном случае выбирается пиксель с центром в в $\left(i + \frac{3}{2}, j + \frac{3}{2}\right)$. Можно было бы определить значение *d,* вычислив $y = mx + b$, но тогда теряется одно из заявленных достоинств алгоритма — использование только целочисленных операций, поскольку *m* является действительной дробью.

Поэтому, во-первых, операции над числами с плавающей точкой заменяются операциями над числами с фиксированной точкой, а во-вторых, все вычисления выполняются в приращениях.

Изменим определение характеристической переменной. Будем использовать

$$d = (x_2 - x_1)(b - a) = \Delta x(b - a) \tag{80}$$

Это изменение не влияет на выбор пикселей-кандидатов, поскольку для принятия решения используется только знак характеристической переменной, а он в новой формулировке будет таким же, как и в прежней. Подставляя в выражение d значения a и b, полученные из уравнения прямой, и принимая во внимание, что

$$m = \frac{\Delta x}{\Delta y} \text{ и } h = y_2 - mx_2 \tag{81}$$

приходим к выводу, что *d* принимает только целочисленные значения. Итак, нам удалось избавиться от операций над действительными числами, но

непосредственное вычисление d все же требует большого количества вычислений, хотя и целочисленных.

Поэтому несколько изменим подход. Пусть d_k— значение d при $x = k + \frac{1}{2}$. Попробуем вычислить d_{k+1} в приращениях относительно d_k. Значение приращения будет зависеть оттого, выполнялось ли на предыдущем шаге приращение координаты y пикселя. Возможные варианты представлены на рисунке.

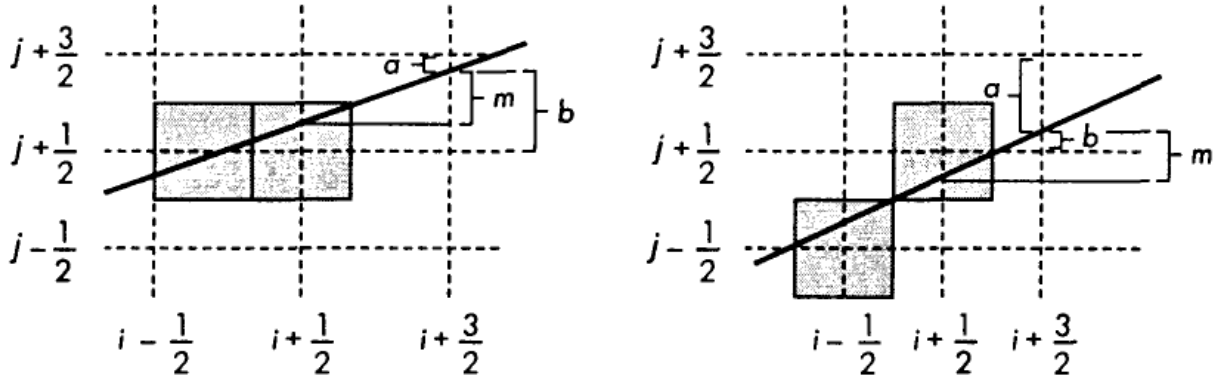


Рисунок 2.27. Приращение значений параметров a и b

Для параметров a и b — расстояний между прямой и пикселями-кандидатами — справедливы следующие соотношения. Если на предыдущем шаге произошло приращение y, то

$$\begin{aligned} a_{k+1} &= a_k - m + 1 \\ b_{k+1} &= b_k + m - 1 \\ b_{k+1} - a_{k+1} &= (b_k - a_k) + 2m - 2 \end{aligned} \tag{82}$$

В противном случае:

$$\begin{aligned} a_{k+1} &= a_k - m \\ b_{k+1} &= b_k + m \\ b_{k+1} - a_{k+1} &= (b_k - a_k) + 2m \end{aligned} \tag{83}$$

Умножив полученные выражения на Δx, получим, что при переходе к следующему значению x значение d изменяется либо на $2\Delta y$, либо на $2(\Delta y - \Delta x)$. Этот результат можно описать в таком виде:

$$d_{k+1} = d_k + \begin{cases} 2\Delta y, & d_k < 0 \\ 2(\Delta y - \Delta x), & d_k \geq 0 \end{cases} \tag{84}$$

В результате вычисление каждого очередного пикселя растрового образа отрезка потребует только одной операции сложения целых чисел и анализа

знака. Алгоритм настолько эффективен, что в некоторых системах он реализован в виде одной инструкции.

2.2.3. Сглаживание ступенек

После растрового преобразования образы прямолинейных отрезков и ребер многоугольников выглядят как бы состоящими из мелких ступенек. Даже на экране ЭЛТ с разрешением выше 1024x1280 можно невооруженным глазом заметить такие искажения. Природа их кроется в самом принципе преобразования непрерывной функции в дискретную, которая представляется на сетке пикселей с конечным разрешением.

Появление ступенек объясняется тремя причинами. Во-первых, объем буфера кадра, в котором формируется дискретизированный образ графического объекта, конечен, и представление в нем прямолинейного отрезка в принципе не является взаимно однозначным, т.е. одному и тому же растровому образу может соответствовать множество исходных отрезков. Во-вторых, все пиксели имеют строго фиксированное положение на экране и образуют сетку с постоянным шагом. В-третьих, пиксели имеют фиксированный размер и форму.

На первый взгляд кажется, что, поскольку все эти причины органически присущи дискретному способу представления непрерывного сигнала, вряд ли что-нибудь удастся сделать и что отмеченные погрешности изображения неустранимы. Алгоритмы, подобные предложенному Брезенхэмом, оптимальны в том смысле, что позволяют определить на дискретной сетке позиции, ближайшие к исходной линии. Но если конструкция устройства отображения позволяет воспроизводить оттенки цветов или даже градации одного цвета, эффект ступенчатости изображения можно значительно ослабить. Как известно, линия как математический объект является одномерным многообразием — характеризуется только длиной, но не имеет ширины (толщины). В отличие от исходного математического объекта, его растровый образ имеет конечную ширину, иначе он просто не был бы различим визуально. Каждый пиксель, из которых этот образ состоит, имеет вполне определенные размеры — размер пикселя принят в наших рассуждениях за некую метрическую единицу. Поэтому можно считать, что идеализированный образ математической прямой линии на экране имеет толщину 1 единицу.

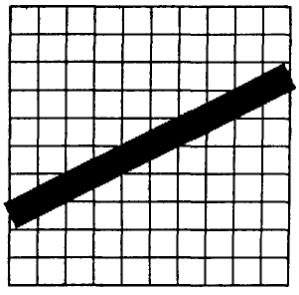


Рисунок 2.28. Идеальный образ прямой на экране

Вычертить на экране растрового устройства отображения такой идеальный образ прямой невозможно, поскольку он не состоит из отдельных квадратных пикселей. Алгоритм Брезенхэма можно считать способом приближенного представления этого идеального образа на сетке из реальных пикселей.

Если присмотреться к идеальному образу прямой, то нетрудно подметить, что многие пиксели он перекрывает частично. В «чистом» виде алгоритм Брезенхэма предполагает, что при коэффициенте наклона, меньшем 1, за каждый шаг перемещения по оси x засвечивается точно один пиксель. Но можно поступить и по-другому — засвечивать и те пиксели, которые частично перекрываются идеальным образом линии, но задавать им не полную интенсивность, а частичную, приблизительно пропорциональную степени перекрытия пикселя.

В результате сформируется сглаженное изображение прямой, примерно такое, как на рисунке.

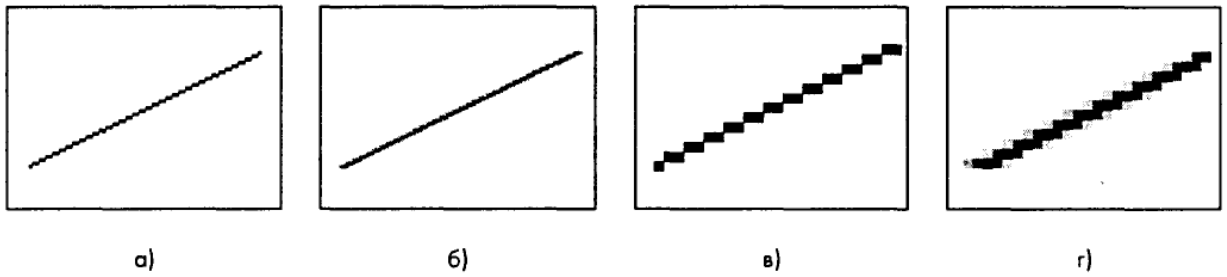


Рисунок 2.29. Сглаживание изображения прямолинейного отрезка

Этот способ получил название сглаживание усреднением по области (antialiasing by area averaging). Для реализации такого способа необходимо выполнить операции, весьма схожие с теми, которые выполняются при отсечении многоугольника. Существуют и другие алгоритмы сглаживания ступеней на изображении, которые применяются при отображении примитивов других типов, в частности многоугольников.

2.2.4. Наложение

Ранее были рассмотрены алгоритмы формирования изображений прозрачных объектов. Описанные методы могут применяться совместно с технологией трассировки лучей, которая сопряжена с достаточно большим объемом вычислений. Реальные системы вывода предоставляет в распоряжение прикладного программиста иной механизм формирования прозрачных объектов, позволяющий обойтись без трассировки лучей в явном виде. Этот механизм получил название *альфа-смешивание (alpha-blending,* или *a-blending).*

Интенсивность альфа-канала входит в качестве четвертой компоненты в вектор цвета в формате RGBA (или RGBα). Прикладная программа может управлять интенсивностью альфа-канала точно так же, как и интенсивностью каждого из основных цветов, т.е. задавать значение интенсивности A для ка-

ждого пикселя. Но исполняющая система визуализации трактует это значение по-иному. Для нее заданное значение интенсивности компоненты A определяет коэффициент пересчета значений основных цветов при записи их в буфер кадра. В результате оказывается, что в код засветки пикселя вносит свой вклад несколько геометрических объектов и формируется смешанное, или комбинированное, изображение.

Коэффициент поглощения α определенной среды есть количественная мера, позволяющая оценить, какая доля светового потока, падающего на поверхность раздела сред, проходит дальше, а какая поглощается. Если коэффициент поглощения равен 1 ($\alpha = 1$), то свет полностью поглощается на поверхности раздела. Поверхность, для которой коэффициент α равен 0, является идеально прозрачной — весь падающий на нее световой поток проходит дальше. Коэффициент прозрачности τ (transparency) поверхности (фактически среды) связан с коэффициентом поглощения простой зависимостью $\tau = 1 - \alpha$.

Рассмотрим три равномерно окрашенных многоугольника, показанных на рисунке.

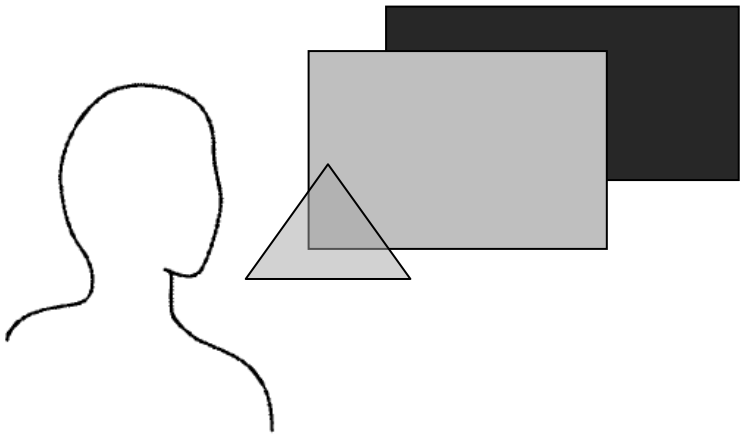


Рисунок 2.30. Наложение полупрозрачных объектов

Предположим, что средний многоугольник непрозрачен, а ближайший к наблюдателю — прозрачен. Если он будет сделан из идеально прозрачного материала, то наблюдатель вообще его не увидит — он увидит только расположенные за ним непрозрачные предметы. Однако если ближайший многоугольник только частично прозрачен (часто говорят — *полупрозрачен),* как солнцезащитные очки, то наблюдатель увидит в этом месте некоторый совмещенный цвет, в котором присутствует и цвет полупрозрачного объекта, и цвет расположенного за ним непрозрачного. Поскольку средний многоугольник непрозрачен, то третьего многоугольника наблюдатель за ним не увидит. Если ближайший многоугольник красный, а расположенный за ним — синий, то комбинированный цвет будет иметь фиолетовый оттенок вследствие наложения *(смешивания)* цветов. Если же и средний многоугольник полупрозрачен, то наблюдатель увидит еще более сложное сочетание цветов всех трех объектов.

В системах компьютерной графики образы многоугольников формируются последовательно, причем порядок их размещения по отношению к наблюдателю в типовой системе не влияет на порядок формирования экранных образов. Следовательно, если желательно получить такое смешанное изображение, нужно изобрести способ, позволяющий учитывать коэффициент прозрачности объектов на стадии растрового преобразования. Мы будем в дальнейшем использовать понятия пикселя-источника (source pixel) и пикселя-приемника (destination pixel) в том же смысле, что и понятия буфера-источника и буфера-приемника.

При обработке очередного многоугольника вычисляется фрагмент изображения размером с пиксель, и если этот фрагмент видим, то пикселю присваивается цвет, сформированный в соответствии с принятой моделью закрашивания. До сих пор мы использовали вычисленный алгоритмом тонирования цвет фрагмента, на который при необходимости еще и была наложена определенная текстура, для замены кода засветки соответствующего пикселя в буфере кадра. Если рассматривать вычисленный код засветки в качестве интенсивности пикселя-источника, а текущий код засветки в этой ячейке буфера кадра как интенсивность пикселя-приемника, то эти две интенсивности можно скомбинировать самыми разными способами.

Использование α-канала является одним из способов выполнить наложение (смешивание) цветов на уровне отдельных пикселей. Такое наложение моделирует замену двух полупрозрачных цветных стекол одним, которое будет иметь цвет и коэффициент поглощения, отличные от соответствующих характеристик каждого из исходных компонентов.

Если представить пиксель-источник и пиксель-приемник в виде массивов из четырех элементов (RGBA): $s = [s_r, s_g, s_b, s_\alpha]$ и $d = [d_r, d_g, d_b, d_\alpha]$, то результатом смешивания будет новый код засветки:

$$d' = [b_r s_r + c_r d_r, b_g s_g + c_g d_g, b_b s_b + c_b d_b, b_\alpha s_\alpha + c_\alpha d_\alpha] \qquad (85)$$

Массивы констант b и c называются соответственно коэффициентами смешивания (blending factors) источника и приемника. Как и базовые цветовые составляющие RGB, значение параметра α не может превышать 1.0 и быть меньше 0.0. Подбирая значения компонента α и коэффициентов смешивания, можно получить массу интересных эффектов при смешивании изображений.

Основная сложность в управлении процессом формирования смешанного изображения состоит в том, что на результат влияет порядок формирования образов многоугольников, который очень сложно контролировать из прикладной программы. Например, в некоторых приложениях используется режим, когда коэффициент смешивания источника приравнивается к значению α, а коэффициент смешивания приемника — к значению $(1 - \alpha)$. В результате после выполнения смешивания очередного изображения получим:

$$\begin{pmatrix} R \\ G \\ B \\ \alpha \end{pmatrix} = \begin{pmatrix} \alpha_s R_s + (1-\alpha_s) R_d \\ \alpha_s G_s + (1-\alpha_s) G_d \\ \alpha_s B_s + (1-\alpha_s) B_d \\ \alpha_s \alpha_s + (1-\alpha_s) \alpha_d \end{pmatrix} \quad (86)$$

Эта формула показывает, что результат будет корректен как для прозрачных, так и для непрозрачных объектов, но он все-таки существенно зависит от того, в каком порядке будут обрабатываться объекты при растровом преобразовании. Следовательно, в отличие от других случаев, когда прикладной программе не было никакого дела до порядка обработки объектов при растровом преобразовании, в данном случае желаемый эффект может быть получен только при условии соблюдения определенного порядка.

При использовании смешивания изображений возникает проблема и с механизмом удаления невидимых поверхностей. Обычно в такой ситуации этот механизм нужно отключать, поскольку он не делает различия между непрозрачными и прозрачными объектами, а следовательно, любой объект, который расположен за другим объектом, пусть он даже имеет коэффициент поглощения 0.01, просто не будет подвергаться растровому преобразованию и никак не «проявится» в результирующем изображении.

Нам же нужно, чтобы те объекты, которые расположены за полупрозрачными объектами, включались в изображение. Этого можно достичь достаточно простым приемом, управляя маской обновления z-буфера. Дело в том, что для определенных объектов, в данном случае — прозрачных, можно при работе с z-буфером выборочно настраивать режим «только для чтения».

Когда буфер глубины работает в режиме «только для чтения», прозрачный объект, расположенный за непрозрачным, образ которого уже сформирован, растровому преобразованию подвергаться не будет. Если же прозрачный объект расположен перед непрозрачным, то он будет подвергаться растровому преобразованию, но в результате сформируется смешанный цвет. При этом данные о глубине размещения прозрачного объекта в буфер глубины занесены *не будут.* При обработке непрозрачных объектов восстанавливается нормальный режим работы z-буфера, и процесс идет как обычно.

3. Реальные системы визуализации

В приведенных ранее разделах рассматривались основные проблемы визуализации трехмерной графики безотносительно реально существующих системы вывода. Однако в настоящее время системы визуализации реального времени получили очень большое распространение — не в последнюю очередь благодаря компьютерным играм, которые на данный момент являются одними из самых ресурсо- и трудоемких приложений компьютерной графии на домашних компьютерах.

И именно благодаря компьютерным играм технологии визуализации трехмерных изображений развиваются семимильными шагами. Главным таким шагом было появление в 90-х годах XX века выделенного устройства под названием «графический ускоритель», встроенного в плату формирования видеоизображения.

Основной задачей такого ускорителя было формирование собственного конвейера визуализации трехмерных изображений. Если ранее вся ответственность за вывод подобного рода графики возлагалась на центральный процессор (CPU, Central Processing Unit), то теперь этой задачей начал заниматься отдельный модуль под названием GPU (Graphics Processing Unit).

Появление подобного устройства позволило в десятки раз повысить быстродействие компьютерной графики реального времени, однако появились и некоторые ограничения. Так, если ранее на CPU можно было реализовывать практически любые допустимые алгоритмы, то теперь возможности вывода ограничивались возможностями графического ускорителя. До настоящего времени все производители графических ускорителей и подсистем вывода занимаются как раз тем, что расширяют подобные возможности вывода.

Естественным образом, первоначально все графические ускорители имели разный программный интерфейс, чем значительно усложняли жизнь разработчикам программного обеспечения. По этой причине не замедлили появиться программные системы, предоставляющие обобщенный интерфейс для работы с графическими ускорителями. На данный момент очень широкое распространение получили две такие системы — открытая OpenGL (Эволюционировавшая из системы IrisGL компании Silicon Graphics) и DirectX (собственность компании Microsoft). Следует отметить, что последняя представляет собой целый пакет для разработки мультимедийных приложений, и в дальнейшем под DirectX мы будем подразумевать только DirectGraphics, составную часть DirectX, отвечающую за визуализацию графики.

Обе системы получили настолько широкое распространение, что их стандарты уже поддерживаются видеокартами аппаратно и стали де-факто основой для разработки самих графических ускорителей.

3.1. Устройство систем визуализации

В настоящем разделе рассматривается устройство и методы программирования системы визуализации DirectGraphics, составной части DirectX. Обычно для иллюстрации работы графических алгоритмов применяют более академическую систему OpenGL, однако ее процедурная архитектура на настоящий момент, на взгляд автора, требует значительного пересмотра. Объектно-ориентированная система DirectX может показаться чуть более сложной к освоению, однако на практике она зачастую показывает более высокое быстродействие, и по распространенности на настоящий момент является лидирующей по сравнению с OpenGL.

Вообще говоря, сложно рассуждать о превосходстве одной системы над другой — решающим голосом всегда является производительность видеокарты. И не существует такого функционала DirectX, который не мог бы быть реализован на OpenGL, и наоборот.

Таким образом, все сказанное ниже хоть и относится в подсистеме визуализации DirectGraphics, с легкостью может быть перенесено на систему OpenGL, зачастую с минимальными изменениями.

3.1.1. История DirectX

Изначально нацеленный на разработку видеоигр, DirectX стал популярен и в других областях разработки программного обеспечения. К примеру, DirectX, наряду с OpenGL, получил очень широкое распространение в инженерном/математическом ПО.

В 1994 году Microsoft была практически готова выпустить следующую версию Windows — Windows 95. Главным фактором, определяющим, насколько популярна будет новая ОС, являлся набор программ, которые можно будет запускать под её управлением. В Microsoft пришли к выводу, что, пока разработчики видят DOS более подходящей для написания игровых приложений, коммерческий успех новой ОС весьма сомнителен.

DOS позволяла разработчику получить прямой доступ к видеокарте, клавиатуре/мыши/джойстику и прочим частям системы, в то время как Windows 95, с её защищённой моделью памяти, предоставляла более стандартизованный, но в то же время весьма ограниченный доступ к устройствам. Microsoft нуждалась в новом способе дать разработчику всё, что ему необходимо. Айслер (Eisler), Сэйнт Джон (St. John), и Энгстром (Engstrom) решили эту проблему, назвав само решение DirectX.

Первый релиз DirectX был выпущен в сентябре 1995 года, под названием «Windows Game SDK».

Ещё до появления DirectX, Microsoft включила OpenGL в ОС Windows NT. Direct3D позиционировался как замена OpenGL в игровой сфере. Отсюда берёт своё начало «священная война» между сторонниками кроссплатформенной OpenGL и доступной лишь в Windows (в т.ч. Windows NT) Direct3D. Так или иначе, остальные части DirectX очень часто комбинируют-

ся с OpenGL в компьютерных играх, так как OpenGL как таковой не подразумевает функциональность уровня DirectX (например, доступ к клавиатуре/джойстику/мыши, поддержка звука, игры по сети и т. д.).

В 2002 году Microsoft выпустила DirectX 9 с улучшенной и расширенной поддержкой шейдеров. С 2002 года DirectX неоднократно обновлялся. В августе 2004 года в DirectX была добавлена поддержка шейдеров версии 3.0 (DirectX 9.0c).

3.1.2. COM-объекты

Как уже упоминалось выше, DirectX является объектно-ориентированной системой, в отличие от процедурной OpenGL. Фактически, DirectX реализуется в виде набора динамических библиотек (dll) для операционной системы Windows.

Как и любая объектно-ориентированная система, DirectX оперирует объектами, вызывая у них некоторые методы. Для более абстрактного представления данных, в DirectX используется интерфейсная модель. То есть конечный пользователь оперирует не самими объектами, а методами интерфейсов, которые реализует каждый конкретный объект.

Для работы с объектами и интерфейсами DirectX использует технологию объектной модели компонентов (COM). Эта технология позволяет создавать объекты, получать их интерфейсы и вызывать их методы, не используя детали конкретной реализации самих объектов.

Основным понятием, которым оперирует стандарт COM, является COM-компонент. Программы, построенные на стандарте COM, фактически не являются автономными программами, а представляют собой набор взаимодействующих между собой COM-компонентов. Каждый компонент имеет уникальный идентификатор (GUID) и может одновременно использоваться многими программами. Компонент взаимодействует с другими программами через COM-интерфейсы — наборы абстрактных функций и свойств.

Технология COM не зависит от языка программирования — постулируются лишь стандарты создания объектов и вызова функций. Таким образом, можно указать следующие важные детали организации COM-компонент:

- Весь требуемый функционал реализуется при помощи объектов.
- Каждый объект имеет свой глобальный уникальный идентификатор (GUID), позволяющий идентифицировать его среди прочих.
- Все объекты зарегистрированы в операционной системе под своими идентификаторами
- Операционная система предоставляет функцию создания произвольного объекта по его идентификатору.
- Каждый объект реализует один или несколько интерфейсов. Выражение «объект реализует интерфейс» означает, что в объекте реализованы все методы данного интерфейса.

- Все интерфейсы унаследованы от базового IUnknown. Таким образом, все объекты реализуют этот интерфейс.
- Любой интерфейс объекта может быть получен при помощи метода QueryInterface интерфейса IUnknown.

Из всего вышеперечисленного можно описать следующую структуру COM-компонент. Каждый компонент представляет собой объект, реализующий один или несколько интерфейсов. Детали этой реализации, равно как и объявление самого класса, сокрыты от глаз конечного пользователя (чаще всего они хранятся в скомпилированной форме в некоторой dll).

Когда пользователь хочет получить доступ к некоторому компоненту, он создает его при помощи некоторой функции CreateInstance, которой в качестве аргумента передается ассоциированный с объектом глобальный идентификатор (GUID).

Глобальный идентификатор позволяет с точность идентифицировать ассоциированный с ним объект. Система генерации ключей GUID гарантирует, статистическую уникальность подобного ключа. В текущей реализации GUID имеет вид 128-битного ключа и часто записывается в шестнадцатеричной форме:

```
6F9619FF-8B86-D011-B42D-00CF4FC964FF
```

Все компоненты регистрируются под своими уникальными ключами в системном реестре. Ключи обычно генируются при помощи специальной программы.

Каждый добавленный в систему компонент обязан быть зарегистрирован в системе. Впрочем, обычно этим занимается программа инсталляции.

Таким образом, реализующий COM-объект класс может быть записан следующим образом

```
[uuid(529c43ff-cb2b-4aee-8ce5-3d0bad967674)]
class ComObjectSample
  : public IUnknown
{
};
```

Строчкой вида [uuid(…)] мы будем ассоциировать класс с глобаным идентификатором

Зарегистрированный в системе объект может быть создан при помощи общесистемной функции CreateInstance. Получив на вход ассоциированный с объектом GUID, функция возвращает указатель на интерфейс IUnknown, реализуемый новосозданным объектом. Заметим, что никаких деталей реализации ни в этот, ни в какой другой момент пользователю не доступно.

В общем виде функция может выглядеть примерно следующим образом:

```
HRESULT CreateInstance ( UUID & guid, IUnknown** ppUnk ) {
if ( guid == __uuidof(ComObjectExample) ) {
    *ppUnk = new ComObjectExample();
    *ppUnk->AddRef();
    return S_OK ;
```

```
    }
    return E_FAIL ;
}
```

Здесь функция __uuidof возвращает ассоциированный с объектом идентификатор. Подробнее об этом можно узнать из [9] и [10].

При помощи метода QueryInterface интерфейса IUnknown пользователь может получить указатель на любой другой интерфейс, поддерживаемый объектом. Каждый объект, являющийся COM-компонентом, обязан реализовывать этот метод и корректно отдавать указатели на требуемые интерфейсы.

Для идентификации интерфейсов также используется система глобальной идентификации GUID. Так, например, интерфейс может быть объявлен следующим образом:

```
[uuid(49e6dca4-9cba-4a75-b38f-ede9132a4a06)]
interface ISample : public IUnknown
{
    virtual void DoSomeThings () = 0 ;
};
```

Таким образом, полное объявление COM-объекта из приведенного ранее примера будет следующим:

```
[uuid(529c43ff-cb2b-4aee-8ce5-3d0bad967674)]
class ComObjectSample
  : public IUnknown
  , public ISample
{
  void DoSomeThings (){ /*...*/ }


  HRESULT QueryInterface ( UUID & iid, void ** p ) {
    if ( iid == uuidof(ISample) ){
       *p = (ITeacher *)this ; AddRef (); return S_OK ;
    }
    if ( iid == uuid(IUnknown) ){
       *p = (IUnknown *)this ; AddRef (); return S_OK ;
    }
    return E_NOINTERFACE ;
 }
};
```

В этом коде присутствуют функции AddRef и Release, а также неизвестные еще коды возврата E_NOINTERFACE и S_OK. Они будут рассмотрены далее.

3.1.3. Подсчет ссылок

Как упоминалось выше, создать COM-объект можно при помощи функции CreateInstance. Однако каждый созданный объект должен быть удален — как же это осуществить для объектов, точка создания которых на неизвестна.

Для решения этой проблемы Microsoft предлагает технологию подсчета ссылок. Заключается она в следующем.

Каждый раз, когда создается новый объект, с ним ассоциируется некоторый счетчик, в который помещается единица. Как только коду, использующему этот объект, потребуется его кому-то передать, этот счетчик увеличивается. А когда ссылка или указатель на объект становятся больше не нужны, счетчик уменьшается на единицу. Когда счетчик достигает нуля, это означает, что объект больше никем не используется, и в этом случае он удаляется.

В COM-объектах подсчет ссылок реализован следующим образом. В интерфейс IUnknown кроме метода QueryInterface вводятся еще два метода — AddRef и Release. Первый инкрементирует значение внутреннего счетчика, второй уменьшает.

Реализация этих методов отдается на откуп разработчику COM-объекта. В самом деле, изменение счетчика может быть реализовано как с учетом потоковой безопасности, так и вообще с учетом того, что объект должен быть один на все вызовы CreateInstance.

Обычно все же счетчик представляет собой целое число, которое размещается в памяти самого объекта. В этом случае методы будут выглядеть следующим образом:

```
unsigned AddRef () {
     return ++count_ ;
}
unsigned Release () {
     if ( count_-- == 0 )
        delete this ;
     return count_ ;
}
```

В приведенном коде count_ — переменная типа int, размещенная в памяти класса. Из кода видно, что при достижении счетчиком ссылок нуля объект будет уничтожен.

Также стоит отметить, что созданный объект находится в подвешенном состоянии — счетчик ссылок равен нулю, но объект существует. Поэтому у созданного объекта необходимо сразу же вызвать метод AddRef, что и делается в примере функции CreateInstance.

Наиболее важным выводом из приведенного служит тот факт, что созданный объект должен быть удален. Если память под объект выделсяется при помощи оператора new, где-то должен быть сопутствующий delete. Соответственно, если некоторый объект DirectX (например, эффект, или текстура) создаются при помощи функции CreateInstance (или других функций из семейства Create*), память из-под него обязательно должна быть высвобождена при помощи вызова метода Release().

3.1.4. Коды ошибок

При рассмотрении вышеприведенного кода также можно было отметить, что везде используется некоторый тип HRESULT и коды возврата. Этот тип

используется в Windows повсеместно и позволяет сообщить первичные данные об успехе функции или о ее ошибке.

Тип HRESULT объявляется как DWORD и является 32-битным беззнаковым целым. Коды возврата с установленным последним битом являются кодами ошибок, со сброшенным — кодами успеха.

Так, например, код ошибки S_OK равный 0, соответсвует полному успеху выполнения функции. Код S_FALSE (равный 1) соответствует штатному завершению функции, однако свидетельствует о некотором прогнозируемом неуспехе.

Коды ошибок обычно начинаются префикса E_. Так, код E_NOINTERFACE означает отсутствие подходящего интерфейса в вызове QueryInterface и ей подобным. E_NOIMPL — функция еще не реализована. E_FAIL — некоторый общий сбой.

Более подробно о кодах ошибок можно узнать из [9].

3.2. Архитектура DirectX

Выше были изложены причины использования DirectX в настоящем пособии, а также принципы функционирования DirectX как COM-компоненты в операционной системы Windows. Далее в настоящей лекции будет описана архитектура Direct3D, принципы и основные методы его работы, а также основы использования фиксированного конвейера при построении изображений.

В настоящей и нижеследующих лекциях речь пойдет, как правило, о DirectX версии 9.0c. В момент написания актуальными версиями являются DirectX 10 и DirectX 11, однако последние имеют ряд существенных аппаратных зависимостей, и в настоящий момент мало употребимы и распространимы. Описываемая девятая версия является на данный момент самой распространенной и широко используемой, что и обуславливает ее описание в настоящем пособии.

Как упоминалось ранее, Direct3D предоставляет аппаратно-независимый интерфейс для работы с графическими ускорителями (и с системой вывода изображений на монитор вообще). Как известно, в ОС Windows используется собственная система вывода изображений GDI. На рисунке ниже представлена схема взаимодействия между аппаратной частью и описанными системами.

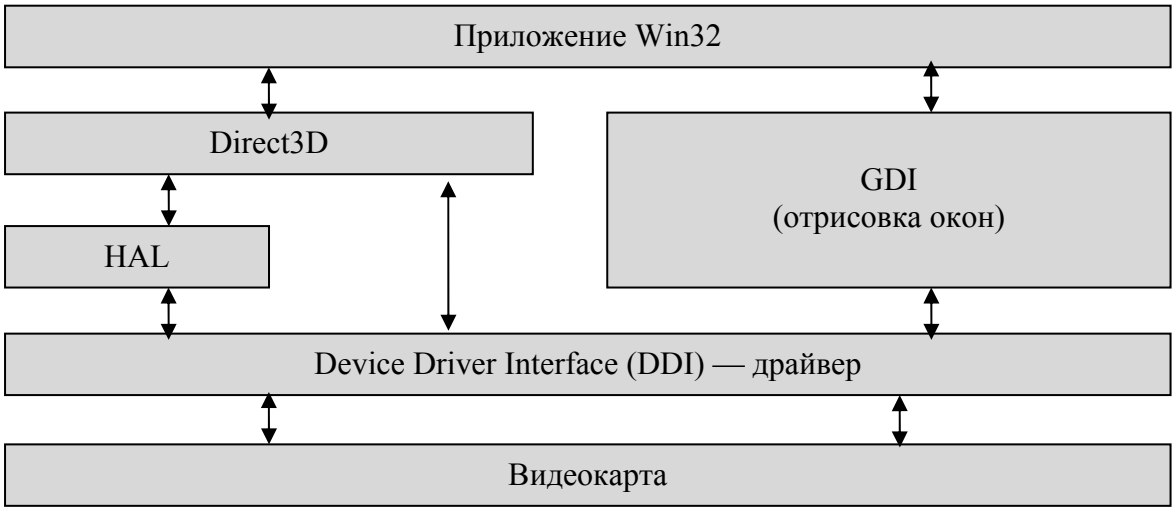


Рисунок 3.1. Связь систем вывода изображений в Windows

Как видно из рисунка, Windows взаимодействует с графической аппаратурой посредством драйвера (DDI). Как GDI, так и Direct3D могут взаимодействовать с этой прослойкой напрямую.

Однако для каждого устройства при некоторой общности интерфейсов драйвера сами драйверы могут существенно различаться, особенно в отношении реализации графического конвейера. По этой причине для Direct3D предоставляется некая аппаратно-независимая прослойка, называемая HAL (Hardware Abstraction Layer).

Эта прослойка описывает некоторое абстрактное устройство визуализации, имеющие одинаковую внутреннюю структуру для всех видов графических адаптеров. Если некоторый функционал, предоставляемый абстрактным устройством, поддерживается графическим адаптером, то он реализуется аппаратно. Если же функционал (capability) не поддерживается, то его реализация осуществляется программно при помощи самой библиотеки Direct3D.

Таким образом, для конечного пользователя все взаимодействие с аппаратной частью происходит при помощи абстрактной прослойки, и пользователю нет необходимости учитывать конкретные детали реализации каждого аппаратного ускорителя.

3.2.1. Устройства Direct3D

Как можно заключить из вышесказанного, основной компонентой Direct3D является некоторое визуализационное «устройство» (device). Это устройство отвечает за выполнение всего конвейера визуализации, включая шаги:

- Преобразование (Transform)
- Освещение (Lighting)
- Растеризация (Rasterization)

В общем случае работу устройства можно представить как совокупность трех указанных модулей.

Рисунок 3.2. Архитектура «устройства» Direct3D

Изначально видеокарты занимались только вопросами растеризации и отображения изображений на экране. Большим прорывом в области построения графических ускорителей стал перенос фазы преобразования и освещения/закрашивания на аппаратную часть. Изначально эта технология носила имя hardware T&L (аппаратное преобразование и освещение), однако позднее, когда весь конвейер визуализации был перенесен на аппаратную часть, смысл этого термина стерся.

Устройство Direct3D также отвечает за сохранение и изменение так называемых состояний (States) — далее мы будем называть их «стэйтами», чтобы избежать путаницы с другими состояниями. Стэйты представляют собой не что иное как настройки визуализации. К стэйтам относятся режимы освещения, наложения, текстурирования, преобразования, использования z-буфера и многое другое. В двух словах работу визуализатора можно описать следующим образом:

1. Установка стейтов
2. Запуск конвейера

Рисование всей сцены обычно заключается в разбиение ее отдельные фрагменты, для каждого из которых устанавливаются все стейты, а затем запускается процедура конвейерной отрисовки. Далее это процесс будет описан более подробно.

Direct3D предоставляет два основных вида устройств:

- абстрактное (HAL),
- опорное (REF).

Абстрактное устройство использует всю мощь лежащего под ней графического ускорителя, опорное же реализуется полностью программно и не зависит от используемой видеокарты. Оба вида устройств разработаны таким образом, что замена их друг другом не должна влиять на работоспособность программы. Однако использование опорного устройства вместо абстрактного существенно снижает быстродействие.

Опорное устройство часто используется для демонстрации возможностей (и особенно нововведений) Direct3D, еще не поддерживаемых ускорителями. В большинстве случаев в реальной жизни необходимо использовать именно абстрактное устройство для получения максимального быстродействия.

3.2.2. Характеристики устройства

Устройство визуализации имеет ряд настроек и характеристик, которые позволяют использовать его в разных случаях по-разному. Рассмотрим основные характеристику устройства.

Основной характеристикой устройства является его тип, описанный ранее. Обычно используется абстрактный тип, однако для операций без видеокарты или для демонстрации полного спектра возможностей может использоваться и опорное (REF) устройство.

Также для устройства необходимо указать идентификатор видеовыхода. Обычно видеовыход соответствует монитору, на который осуществляется вывод. Поскольку большинство систем визуализации являются одномониторными, в качестве идентификатора видеовыхода используется нулевое значение.

Также важным параметром для устройства является окно вывода. Поскольку Direct3D взаимодействует с видеоадаптером напрямую, параллельно с системой вывода GDI, операционной системе необходимо указать в терминах GDI, в каком именно месте будет осуществляться вывод. В этом месте в системе вывода GDI будет создана «дырка» (то есть вывод ОС в это место осуществляться не будет), и вывод Direct3D будет осуществляться непосредственно в эту «дырку».

Важно отметить, что вывод Direct3D может осуществляться как в оконном, так и в полноэкранном режиме. В последнем случае операционная система сама переключит разрешение и глубину цвета видеовыхода на требуемые. Однако в обоих случаях требуется указать окно вывода (HWND в терминах Windows API), чтобы ОС могла осуществлять взаимодействие этого окна с другими.

Упомянутые свойства входят в еще одну характеристику устройства, называемую режимом вывода, в который входят:

- Разрешение окна вывода
- Формат (глубина цвета)
- Оконный/полноэкранный вывод
- Количество вторичных буферов
- Метод (цепочка) обмена

Рассмотри эти параметры подробнее. Поскольку в системе визуализации осуществляется вывод растрового изображения, область вывода представляет собой прямоугольную матрицу из растров. Соответственно, эта матрица обладает свойствами ширины (width) и высоты (height), которые также являются характеристиками устройства. Также, наряду с шириной и высотой, вводится понятие отношения сторон (aspect ratio).

Поскольку каждый растр обладает еще и цветом, характеристической особенностью устройства является еще и формат задания этого цвета. Этот формат также часто называют форматом пикселя, или глубиной цвета экрана. Direct3D поддерживает следующие форматы буферов:

- A2R10G10B10 (32bit)

- A8R8G8B8 (32bit)
- X8R8G8B8 (24bit)
- A1R5G5B5 (16bit)
- X1R5G5B5 (16bit)
- R5G6B5 (16bit)

Как можно отметить, часть форматов кроме цветовых координат R,G и B используют еще либо канал прозрачности A, либо незадействованные биты, отмеченные каналом X.

Все цветовые представления, использующие альфа-канал, могут быть использованы только для вторичных буферов. Рассмотрим подробнее, что это значит.

3.2.3. Экранные буферы и презентация

Экранным буфером называется последовательность байт, описывающая представленное на экране изображение. Очевидно, размер этого буфера будет составлять $width \times heigth \times depth$, где $depth$ — количество байт, требуемое для кодирования цвета одного растра. Иногда, впрочем, размер буфера может отличаться — зачастую строки выравниваются в памяти по некоторой границе (4 или 16 байт) для увеличения быстродействия, и в этом случае размер экрана составит $stride \times height \times depth$, где величина $stride$ определяется как выровненная по соответствующей границе ширина экрана.

Буфер, который в данный момент соответствует выводимому на экран изображению, называется первичным буфером. Зачастую, наряду с первичным буфером, создается один или несколько вторичных буферов.

Обычно размер видеопамяти на графической плате существенно превышает объем памяти, требуемый для вывода экрана, что позволяет размещать на ней дополнительные данные — буферы, текстуры, данные о выводимых вершинах и т.п. Расположение в памяти первичного буфера не фиксировано, и может задаваться некоторым смещением относительно начала адресации.

Таким образом, изменение выводимой картинки на экран может осуществляться как изменением байт в первичном буфере, так и сменой адреса первичного буфера. При первом способе вывода существенным осложнением может стать мерцание (flicker) экрана при последовательном изменении его содержимого — редко вся картинка обновляется за одно рисование.

Второй способ называется двойной буферизацией и позволяет избавиться от мерцания за счет дополнительной памяти, называемой вторичными буферами. Рисование всегда осуществляется во вторичный буфер, и когда рисование окончено, происходит перемещение указателя вывода на экранный буфер, как изображено на рисунке.

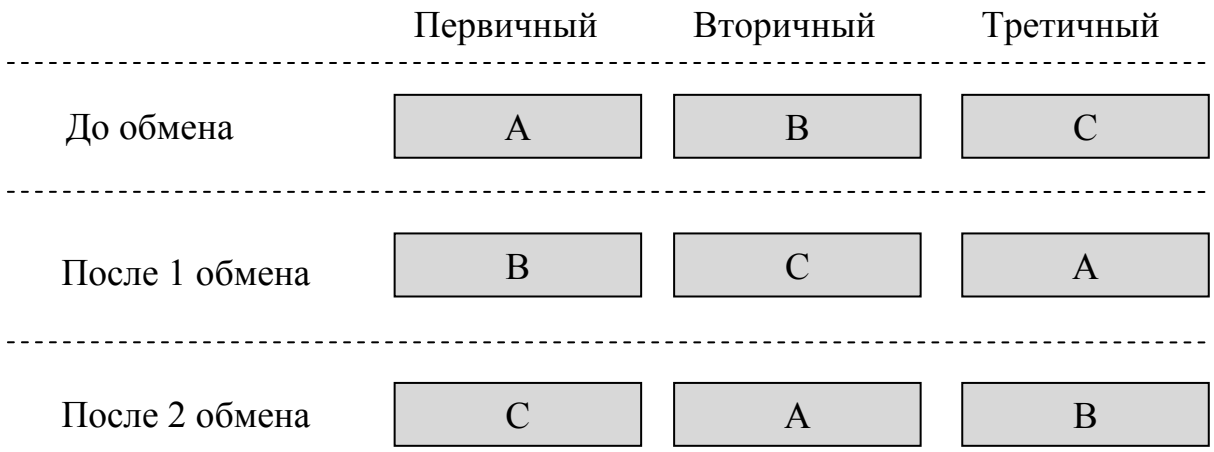


Рисунок 3.3. Многобуферный вывод

Таким образом, вывод осуществляется, по сути, обменом указателей на первичный и вторичный буфер. Такой обмен в терминах Direct3D называется «презентацией».

Для осуществления двойной буферизации заводится, соответственно, два или более буферов. Один из них всегда является выводимым, а рисование осуществляется во второй. Затем осуществляется презентация, указатели на буферы меняются местами, и теперь уже второй буфер является отображаемым, а рисование осуществляется в первый.

Важно отметить, что при презентации осуществляется обмен не содержимого буферов, а только указателей на них, то есть сам процесс вывода построенного кадра осуществляется практически мгновенно.

Для управления методом презентации существуют так называемые «цепочки обмена», о которых пойдет речь далее.

3.2.4. Цепочки обмена

Система Direct3D поддерживает несколько способов осуществления презентации вторичного буфера на экран. Рассмотрим их подробнее.

Самый простой способ — «сброс» (discard). При презентации сбросом осуществляется только обмен указателей на выводимые буферы. При этом ввиду особенностей работы Direct3D и видеокарты содержимое буфера, ранее бывшего первичным, сбрасывается. Таким образом, при отрисовке нового кадра это буфер должен быть очищен, так как система не гарантирует, что его содержимое после сброса будет осмысленным. Такой способ презентации является самым быстрым, так как не осуществляет никакого копирования данных.

Способ презентации, называемый «обменом» (flip), исправляет проблему потери и гарантирует сохранение содержимого в освобождаемом буфере. К сожалению, из-за особенностей реализации, использование системы обмена порождает наличие третьего, невидимого буфера, в который осуществляется копирование данных. По этой причине метод не является быстрым, хотя и сохраняет содержимое буфера.

Третьим методом презентации является прямое копирование (copy), при котором содержимое вторичного буфера копируется в первичный. Никакого обмена не происходит, поэтому не требуются дополнительные буферы. Однако этот способ является одним из самых медленных по быстродействию. Более того, при копировании может быть задан только один вторичный буфер.

В большинстве случаев наиболее оптимальной является указание обмена сбросом, как обеспечивающего наибольшее быстродействие. Содержимое вторичных буферов используется крайне редко, поэтому о его сохранности можно не беспокоиться.

3.2.5. Вертикальная синхронизация

Еще одним параметром устройства является наличие вертикальной синхронизации. Во всех электронно-лучевых, а также во многих жидкокристаллических мониторах вывод осуществляется построчно — луч скользит по поверхности монитора, высвечивая нужные люминофоры (или активизируя нужные кристаллы). Если в процессе вывода содержимое буфера вывода изменится, часть выведенного изображения будет соответствовать предыдущему состоянию буфера, а часть — новому, как изображено на рисунке.

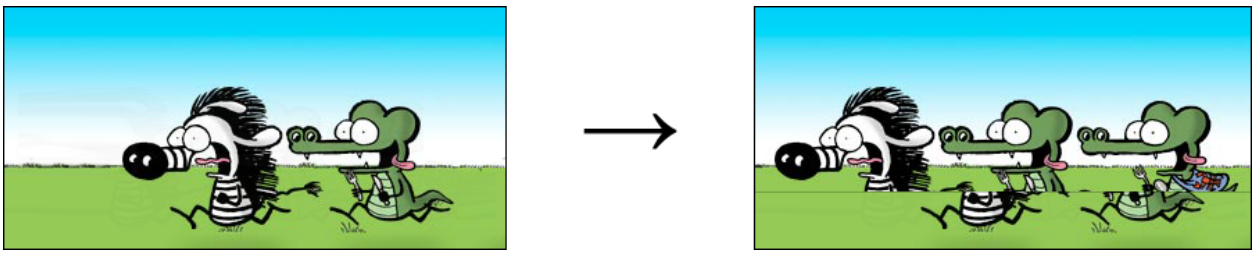


Рисунок 3.4. Проблема вертикальной синхронизации

Для устранения подобной проблемы вводится понятие вертикальной синхронизации. Суть ее в следующем. Во время «презентации» кадра система вывода ожидает

Существует еще ряд других характеристик устройства — параметры сглаживания, флаги поведения и т.п., однако их рассмотрение выходит за рамки настоящего пособия. Подробнее эти и другие характеристики рассмотрены в [9].

3.2.6. Состояния устройства

Обычно видеокарта в каждый момент может выполнять конвейерную отрисовку только для одного набора вершин, то есть для одной программы. В связи с этим устройство может находиться в двух разных состояниях (необходимо различать эти состояния и «стэйты») — рабочем и потерянном.

В рабочем (operational) состоянии устройство находится в «прямом» соединении с графическим ускорителем, и он напрямую выполняет команды устройства. Это штатный режим функционирования для устройства.

Однако переключение пользовательского контекста (например, преход на другую программу, выход в системное меню вызовом Ctrl+Alt+Del и т.п.) заставляет устройство отключиться от видеоадаптера и перейти в так называемое «потерянное» (lost) состояние. В этом состоянии практически любая операция с устройством будет завершаться с ошибкой, так как отсутствует соединение с реальной системой визуализации.

В этом состоянии сбрасываются все данные видеопамяти, включая текстуры, вершинные данные, шейдера и т.п. Ряд данных может не сбрасываться при потере устройства. Подробнее о необходимости восстановления для каждого типа данных можно изучить в [9].

Для возобновления работы необходимо обновить (Reset) устройство и восстановить все потерянные данные на видеокарте.

3.3. Фиксированный конвейер Direct3D

Ранее мы рассмотрели устройство вывода Direct3D и характеристики, необходимые при его создании. После того, как устройство создано, необходимо осуществить саму визуализацию.

В общем случае визуализация разбивается на кадры. Каждый кадр представляет собой последовательное заполнение экранного буфера требуемым изображением. Заполнение изображения может быть разбито на следующие этапы:

1. Очистка буфера
2. Последовательный вывод в буфер примитивов (треугольников, отрезков или точек)
3. Презентация буфера

Шаг 2 подразумевает вывод всей наличествующей в сцене геометрии. Свойства вывода этой геометрии (способы закрашивания, отсечения и т.п.), называемые «стейтами», задаются непосредственно перед ее выводом. Таким образом, шаг 2 раскрывается в следующую процедуру

```
Для всех моделей
    установить стейты
    вызвать отрисовку с указанными вершинными данными
```

В рамках Direct3D описанная процедура будет выглядеть так:

```
device->Clear( /* параметры очистки */ );

device->BeginScene();

// для каждой модели
device->SetRenderState ( /* параметры */ );
device->SetRenderState ( /* параметры */ );
device->Draw ( /* вершины, индексы и т.п. */ );

device->EndScene();
device->Present();
```

Здесь предполагается, что device — указатель на соответствующее устройство Direct3D. В дальнейших разделах будет описаны конкретные способы задания состояний, моделей, методов очистки и т.п.

Остается открытым вопрос — когда производить перерисовку кадра. В системах визуализации «по запросу» (on demand) обычно вызов перерисовки производится после внесения пользователем каких-то корректив — изменения точки зрения, параметров выводимых объектов и т.п. Для систем вывода реального времени чаще используется перерисовка по таймеру, или в моменты простоя (idle) обработчика оконных сообщений. Последний способ оказывает максимальную нагрузку на процессор, однако позволяет достичь максимальной частоты обновления экрана, если таковая задача стоит у разработчика программы.

3.3.1. Библиотека DXUT

Настоящее пособие не ставит своей целью обучить учащегося всем тонкостям разработки приложений под Windows с использованием Direct3D, поэтому для уменьшения объемов кода далее будет использоваться ряд упрощений и вспомогательных библиотек.

Одной из таких библиотек является набор вспомогательных файлов DXUT (DirectX Utility), позволяющее упростить все элементы разработки, не касающиеся DirectX непосредственно. К таковым относятся создание окна, реализация цикла обработки сообщений, получение и обработка пользовательского ввода и т.п. Библиотека DXUT позволяет сосредоточиться непосредственно на использовании DirectX, что является ее неоспоримым преимуществом.

Точкой входа для программы под Windows является функция WinMain. Для инициализации работы библиотеки DXUT предполагается следующий вид этой функции:

```
int WINAPI WinMain( HINSTANCE, HINSTANCE, LPSTR, int )
{
    DXUTInit( true, true );
    DXUTCreateWindow( L"SimpleSample" );
    DXUTCreateDevice( true, 640, 480 );
    DXUTMainLoop();
    return DXUTGetExitCode();
}
```

Приведенный код позволяет инициировать библиотеку, создать окно с именем «SimpleSample», инициировать устройство в оконном режиме и разрешении 640×480, и запустить цикл обработки сообщений.

Теперь, чтобы осуществить покадровое рисование, необходимо установить обработчик рисования. Для этого необходимо перед вызовом DXUTInit осуществить следующий вызов:

```
DXUTSetCallbackD3D9FrameRender( OnFrameRender );
```

Где OnFrameRender — функция обработки кадра, прототип которой выглядит следующим образом:

```
void CALLBACK OnFrameRender( IDirect3DDevice9* device
            , double fTime float fElapsedTime, void* );
```

Первый аргумент этой функции соответствует устройству, при помощи которого осуществляется вывод, второй и третий аргумент — время в секундах, прошедшее от запуска программы и от предыдущего кадра соответственно. Последний аргумент — вспомогательные пользовательские данные, которые могут быть использованы разработчиком по его усмотрению.

Также могут быть установлены обработчики на потерю и восстановление устройства, на пользовательский ввод с мыши и с клавиатуры, и т.п. Разобраться с этими возможностями библиотеки DXUT учащемуся предлагается самостоятельно.

3.3.2. Очистка экрана

Непосредственно перед выводом изображения содержимое буфера необходимо очистить. Осуществляется это при помощи метода Clear устройства Direct3D

```
device->Clear( 0, 0, D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER
    , 0xFFFF00FF, 1, 0 );
```

Опустив значение маловажных аргументов функции, можно указать, что она состоит из флагов очистки и значений. Флаги могут быть комбинацией трех различных буферов — буфера изображений, буфера глубины и буфера стенсила. Описание последнего выходит за рамки настоящего пособия.

Для тех буферов, для которых указаны флаги очистки, необходимо указать значение, которым будет заполнено содержимое буфера. Для буфера изображения указывается цвет в шестнадцатеричной кодировке (например, черному цвету соответствует 0x000000). Для буфера глубины указывается максимальная возможная глубина 1.

После того, как буфер очищен, можно инициировать заполнение сцены, что и будет описано далее.

3.3.3. Конвейер визуализации

Большинство систем визуализации основывается на конвейерной архитектуре. Подобная архитектура позволяет обрабатывать одновременно несколько потоков данных, и является одной из самых оптимальных с точки зрения быстродействия.

Как было описано ранее, процедура визуализации модели строится из двух шагов:

1. Установка стейтов
2. Вызов метода отрисовки

При вызове отрисовки списки вершин и индексов, образующие модель, подаются на вход конвейера. Структура конвейера изображена на рисунке.

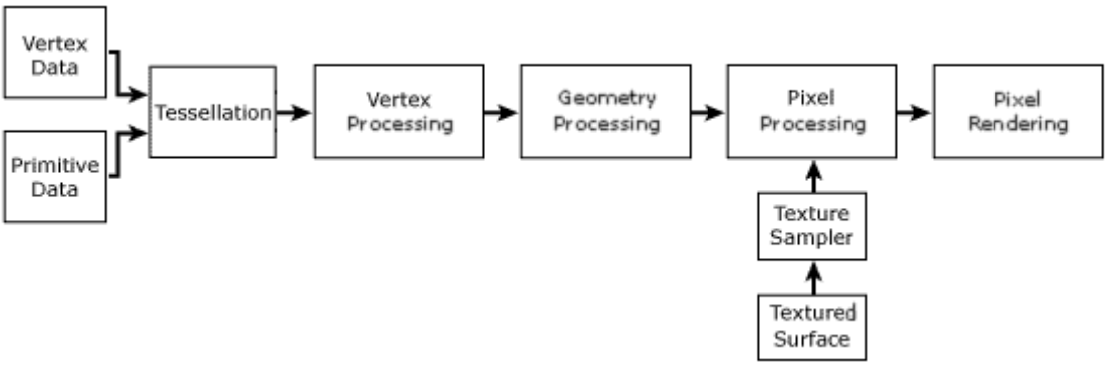


Рисунок 3.5. Конвейер визуализации.

Первым этапом конвейера является этап тесселяции (подразбиения). На этом этапе могут быть сформированы дополнительные вершины и соединяющие из ребра, позволяющие получить более подробную модель. В более

поздних версиях DirectX этот этап получил большое развитие — в частности, стало возможным написание геометрических шейдеров, позволяющих получить очень подробную геометрию модели на аппаратном уровне. Однако при визуализации простых моделей этот этап используется редко.

Следующий этап — геометрическая обработка (называемая также вершинным процессированием). На этом этапе осуществляется преобразование вершинных данных из мировых координат в экранные. Если координаты изначально заданы в экранных координатах, преобразование носит тожественный характер (проще говоря, пропускается). В противном случае преобразование осуществляется домножением на матрицу преобразования, которая стоится как композиция мировой матрицы, матрицы наблюдателя и матрицы проектирования.

В рамках фиксированного конвейера иные преобразования невозможны. При использовании программируемого, а не фиксированного конвейера, существует возможность заменить этот этап на программу (вершинный шейдер), вычисления в которой могут задаваться самим разработчиком.

Следующим этапом является этап растеризации, при котором модель преобразовывается во «фрагмент» — набор точек экранного буфера, соответствующих расположению модели на экране. На этом этапе для каждой точки экранного буфера определяется, будет ли ее цвет изменен при выводе, или же нет.

Также на этапе растеризации проводятся все необходимые отсечения — отсечение по пирамиде видимости, отсечение нелицевых граней и т.п.

Растеризованный фрагмент попадает на этап, называемый пиксельной обработкой. На этом этапе для каждой точки фрагмента определяется ее цвет. Цвет определяется исходя из интерполированных параметров вершин, к растеризованной грани которых принадлежит точка, а также из загруженных в видеопамять текстур. Выборкой из текстур занимается так называемый текстурный сэмплер, о котором пойдет речь позднее.

Наконец, на последнем этапе осуществляется совмещение закрашенного фрагмента и буфера изображения. На этом этапе учитываются стейты, отвечающие за режимы наложения.

Далее рассмотрим, каким образом задаются и представляются примитивы в Direct3D

3.3.4. Описание примитивов

В первом разделе пособия указывалось, что все модели в современных системах визуализации представляют собой наборы из примитивов. Такими примитивами могут быть треугольник, отрезок или точка.

Таким образом, для описания модели достаточно указать список вершин и образующих примитивы индексов, а также метод формирования примитивов по индексам.

В Direct3D для описания вершины используются атрибуты. Основными атрибутами являются:

- Координата
- Нормаль
- Цвет
- Текстурная координата

Описание вершины может выглядеть, например, так:

```
struct Vertex
{
    D3DXVECTOR3 pos ;
    D3DXVECTOR3 nrm ;
    D3DCOLOR    clr ;
};
```

Порядок и размер данных внутри структуры очень важен, так как выводящие функции рассматривают вершинные данные как набор байт. Для того, чтобы указать, какие данные используются при описании вершины, необходимо задать так называемый формат вершин.

Формат вершин строится как комбинация флагов, указывающие тип и размер задаваемых данных. При помощи флагов невозможно описать порядок, поэтому порядок атрибутов вершины жестко фиксируется.

Для задания флагов используется функция SetFVF.Флаги могут быть следующими:

- D3DFVF_XYZ — трехмерные координаты
- D3DFVF_XYZW — четырехмерные однородные координаты
- D3DFVF_XYZRHW — экранные координаты (вершинное преобразования не осуществляется)
- D3DFVF_NORMAL — нормаль
- D3DFVF_DIFFUSE — цвет вершины
- D3DFVF_TEX1 — первые текстурные координаты

Для более подробного описания форматов вершин можно обратиться к [9]

Вывод осуществляется при помощи четырех функций вида DrawPrimitive*. Две из них используют метод загрузки вершин в видеопамять, и не рассматриваются в данном пособии как излишне сложные.

Функции DrawPrimitiveUP и DrawIndexedPrimitiveUP оперируют непосредственно со списками вершин и индексов. Безындексный формат задания предполагает, что последовательность индексов совпадает с последовательностью вершин в вершинном списке.

При вызове этих функций необходимо указать адреса вершинного и индексного (если требуется) списков, и форматы представления в них данных.

Как упоминалось ранее, вызов функции Draw* приводит к запуску конвейера отрисовки с указанными в данными.

3.3.5. Описание преобразований

Преобразования, как упоминалось выше, задаются при помощи трех матриц — мировой, наблюдателя и проективной. Для работы с матрицами

вместе в DirectX SDK поставляется библиотека D3DX, (Direct3D eXtension), предоставляющая ряд функций и интерфейсов, существенно упрощающих работу с математическими функциями, а также рядом других объектов Direct3D, таких как шрифты, текстуры, эффекты и т.п.

Матрица мира для каждого объекта может быть задана произвольно. Матрица обязательно должна быть инициирована единичной (например, при помощи функции D3DXMatrixIdentity). В дальнейшем мировые матрицы могут преобразовываться при помощи композиции матриц поворота, перемещения масштабирования и т.п.

Матрица наблюдателя может быть задана рядом специальных функций. Например, одной из таких функций является

```
D3DXMatrixLookAtLH ( &mtView, &eye, &at, &up );
```

Первым ее аргументом является матрица для заполнения, остальные же задают положение наблюдателя, точку обзора (на которую смотрим наблюдатель), а также направление «верха» камеры. Последнее нужно по следующей причине: положение наблюдателя и точка обзора задают ось, на которой находится (и вдоль которой направлена) гипотетическая «камера». Однако эта камера может свободно вращаться по этой оси. Для того, чтобы камеру зафиксировать, необходимо указать еще одно направление ориентации камеры. Таким направлением является вектор «верха», по которому ориентация камеры в пространстве определяется однозначно.

Для указания матрицы перспективного проектирования используется функция

```
D3DXMatrixPerspectiveFovLH ( &mtProj, .5, 4./3., 0.1, 1000 );
```

Здесь первым аргументом опять же выступает целевая матрица, а далее задаются:

- Горизонтальный угол обзора (fov), в радианах,
- Отношение длины экрана к ширине (aspect ratio),
- Расположение ближней и дальней отсекающих плоскостей.

Можно отметить, что функции построения матриц наблюдателя и проектирования обладают постфиксом LH. Он означает использование левосторонней (Left Handed) координатной системы. Ко всем таким функциям существует парная с постфиксом RH, соответствующая правосторонней системе.

Существую и другие функции для задания приведенных выше матриц, рассмотрении которых оставляются учащемуся на самостоятельное изучение.

Для того, чтобы передать устройству соответствующие матрицы, используется метод SetTransform. Так, чтобы указать все три преобразования, необходимо осуществить следующие вызовы:

```
device->SetTransform ( D3DTS_VIEW      , &mtView );
device->SetTransform ( D3DTS_WORLD     , &mtWorld );
device->SetTransform ( D3DTS_PROJECTION, &mtProj );
```

Отметим, что порядок и место вызова этих функций не принципиально, важно лишь то, с какими матрицами осуществляется запуск конвейера отрисовки.

3.3.6. Отсечения

Для отсечения по глубине в Direct3D применяется технология Z-буфера. Для этого создается дополнительный буфер, разрешение которого совпадает с разрешением экрана. Отсечение производится на этапе растеризации,

Для корректного выполнения отсечения на каждом кадре буфер глубины необходимо очищать. Ранее было описано, как это можно сделать.

Управляется отсечение при помощи стейтов. Так, чтобы сбросить для текущего вывода отсечение по глубине, необходимо осуществить вызов

```
device -> SetRenderState ( D3DRS_ZENABLE, FALSE );
```

Обратное включение производится тем же методом. Также можно управлять отсечениями по пирамиде обзора стейтом D3DRS_CLIPPING.

Стейт D3DRS_CULLMODE служит для управления отсечениями нелицевых граней. Для него может быть указано три значения:

- D3DCULL_NONE — отсечение не производится
- D3DCULL_CW — для отсечение строится нормаль по правилу правой руки при обходе индексов грани по часовой стрелке (ClockWise).
- D3DCULL_CCW — для отсечение строится нормаль по правилу правой руки при обходе индексов грани против часовой стрелки (CounterClockWise).

Все три типа отсечений могут быть установлены и сброшены независимо.

3.3.7. Освещение

Освещение в Direct3D управляется при помощи стейта D3DRS_LIGHTING. В выключенном состоянии этого стейта для закрашивания треугольников используется их вершинный цвет.

Если стейт включен, для закрашивания используется один или несколько (не более 8) источников света. Каждый источник включается отдельно при помощи метода LightEnable

```
device->LightEnable( 0, TRUE );
```

Параметры источника света могут быть заданы при помощи методов GetLight и SetLight. Источники могут быть трех типов:

- Точечные (Point)
- Прожекторы (Spot)
- Направленные (Directional)

По-умолчанию все источники света являются направленными. Кроме этого, для источников можно указать их цвет, параметры затухания, положение, ориентацию, а для прожекторов еще и размеры конусов освещения.

Кроме того, при помощи метода SetMaterial для текущей модели могут быть указаны параметры материала — цвета и коэффициенты фонового, собственного и отраженного света.

На фиксированном конвейере освещенность всегда рассчитывается для вершин. Интерполяция освещения задается при помощи стейта D3DRS_SHADEMODE, принимающем значения D3DSHADE_FLAT (плоское закрашивание) или D3DSHADE_GOURAUD (интерполяция по Гуро).

Также заявлена интерполяция по Фонгу (D3DSHADE_PHONG), однако она не поддерживается ни одной из версий DirectX, так как гораздо проще реализуется при помощи программируемого конвейера.

3.3.8. Текстурирование

Система Direct3D поддерживает три основных типа текстур:

- Двумерные
- Пространственные (трехмерные)
- Кубические

Последние два типа являются довольно сложными и служат для реализации сложных методов закрашивания, описание которых выходит за рамки настоящего пособия. В подавляющем большинстве случаев используются обычные двумерные текстуры.

Текстура описывается интерфейсом ID3DTexture9. Для загрузки текстуры можно воспользоваться вспомогательным методом библиотеки D3DX

```
D3DXCreateTextureFromFile
```

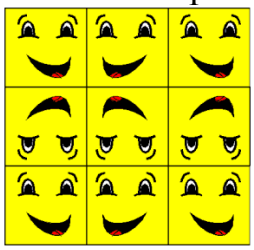
Для загрузки текстуры из файла указывается устройство, адрес файла, а также адрес указателя на текстуру.

Поскольку при вызове этой и аналогичных функций создается объект, после его использования необходимо освободить занимаемую им память, вызвав соответствующий метод Release.

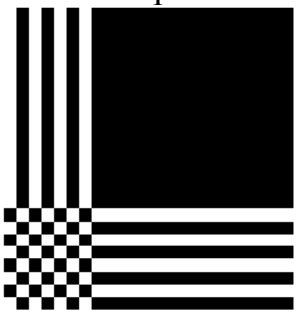
Для двумерных текстур вводится понятие адресации. Адресация указывает, каким образом производится выборка из текстуры при превышении текстурными координатами единичного квадрата. Доступными методам адресации являются:

- Повтор (wrap) — целая часть координаты отбрасывается, и используется только дробная.

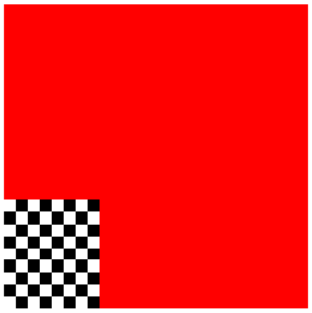
- Отражение (mirror) — при выходе за диапазон координаты отражаются от краев



- Обрезка (clamp) — при выходе за диапазон координаты приравниваются к краевым значениям



- Краевой цвет (border color) — при выходе за диапазон вместо значения текстуры возвращается указанный цвет



Выборка точки из текстуры по указанным координатам называется сэмплированием. Сэмплирование бывает следующих видов:

- По ближайшей точке (Nearest) — из текстуры выбирается ближайший к указанным координатам тексель, и возвращается его цвет. Подобная фильтрация может привести к значительной «каще» при сильном удалении от объекта, а также появлению жестких пикселизованных перепадов цвета.
- Билинейная интерполяция (Linear) — из текстуры выбираются четыре ближайших текселя, и результирующий цвет выбирается усреднением между ними в зависимости от расстояний. Такой метод ведет к более размытому текстурированию, однако предотвращает появление «каши» на экране и способствует появлению более плавных линий.
- Анизотропная интерполяция (Anisotropic) — схожа с билинейной, однако при усреднении учитывается ориентация текстурируемой грани. Тем самым эффект «каши» еще больше минимизируется на

наклонных поверхностях. Однако этот метод является самым ресурсоемким и может серьезно замедлить вывод.

Для указания текстуры при выводе геометрии необходимо воспользоваться методом SetTexture. Для него нужно указать текстурный «этап» (stage) — идентификатор текстуры (обычно он равен нулю), и саму текстуру, загруженную ранее тем или иным образом.

3.3.9. Наложение

После того, как фрагмент растеризован и закрашен, он должен быть наложен на уже заполненный буфер изображения. Это осуществляется при помощи операции наложения, или блендинга (blending).

При осуществлении наложения основную роль играет альфа-канал. По его содержимому могут производиться так называемы альфа-отсечение и альфа наложение.

Альфа-отсечение позволяет использовать для наложения только те пиксели фрагмента, альфа-составляющая которых удовлетворяет некоторому условию. Такое условие обычно задается при помощи опорного значения (alpharef) и функции сравнения (alphafunc). Так, например, чтобы визуализировать только те точки фрагмента, альфа которых превышает половину (то есть значение 128), можно указать следующий код:

```
device->SetRenderState( D3DRS_ALPHATESTENABLE, TRUE );
device->SetRenderState( D3DRS_ALPHAREF, 128 );
device->SetRenderState( D3DRS_ALPHAFUNC, D3DCMP_GREATER );
```

Использование альфа-отсечений позволяет получить однобитную прозрачность, то есть для каждого пикселя фрагмента ответить на вопрос, виден он или нет.

Если же требуется наложение с полупрозрачностью, необходимо использовать альфа-наложение. Для включения такого наложения необходимо воспользоваться стейтом ALPHABLENDENABLE

```
device->SetRenderState( D3DRS_ALPHABLENDENABLE, TRUE );
```

При альфа-наложении результирующий цвет формируется комбинацией из цвета пикселя фрагмента и цвета соответствующего пикселя буфера. Наложение происходит по формуле

```
ColorDest*FactorDest BlendOp ColorSrc*FactorSrc
```

Оператор наложения BlendOp может быть одним из следующих:

- Сложение (Source + Destination)
- Вычитание (Source - Destination)
- Обратное вычитание (Destination - Source)
- Максимум
- Минимум

Факторы наложения могут принимать одно из следующих значений:

- `D3DBLEND_ZERO`

- D3DBLEND_ONE
- D3DBLEND_SRCCOLOR
- D3DBLEND_INVSRCCOLOR
- D3DBLEND_SRCALPHA
- D3DBLEND_INVSRCALPHA
- D3DBLEND_DESTALPHA
- D3DBLEND_INVDESTALPHA
- D3DBLEND_DESTCOLOR,
- D3DBLEND_INVDESTCOLOR,
- И др.

Например, при обычном наложении без учета прозрачности SrcFactor берется равным 1, а DstFactor – 0. Для осуществления самого обычного альфа-наложения используются коэффициенты A и $(1 - A)$ соответственно.

```
device->SetRenderState( D3DRS_BLENDOP  , D3DBLENDOP_ADD );
device->SetRenderState( D3DRS_SRCBLEND , D3DBLEND_SRCALPHA );
device->SetRenderState( D3DRS_DESTBLEND, D3DBLEND_SRCINVALPHA);
```

При установке значений обоих факторов в единицу можно получить аддитивное наложение, соответствующее световому наложению — таковое часто применяется, например, при имитации световых эффектов.

3.3.10. Заключение

В результате освоения раздела учащиеся ознакомились с основными методами визуализации реального времени на примере системы визуализации DirectX. Дальнейшее изучение этой системы может быть осуществлено самостоятельно по материалам, прилагаемым в [17] и [18].

Список литературы

1. Никулин Е.А. Компьютерная геометрия и алгоритмы машинной графики. — СПб.: БХВ-Петербург, 2005 — 560 с.
2. Кэмпбелл М. Компьютерная графика. М. АСТ, 2006 – 392 с.
3. Торн А. Графика в формате DirectX 9. М.: НТ Пресс 2007 — 288 с.
4. Гонсалес Р., Вудс Р. Цифровая обработка изображений. М.: Техносфера, 2006 — 1072 с.
5. Foley J.D., van Dam A., Feiner S. K., Hughes J. F. Computer Graphics, Second Addison-Wesley, Reading, MA, 1996 – 496 с.
6. Херн Д., Бейкер М.П. Компьютерная графика и стандарт OpenGL. М.: Вильямс, 2005 — 1168 с.
7. Инженерная и компьютерная графика. М.: Высшая школа, 2006 г. — 334 с.
8. Порев В.Н., Блинова Т.А. Компьютерная графика. М.: Юниор, 2006 — 520 с.
9. Shirley P, Marschner S. Fundamentals to Computer Graphics. AKPeters, 2009 – 804 с.
10. Vince J. Mathematics for Computer Graphics. Springer, 2005 — 247 с.
11. Шикин Е.В., Боресков А.В. Компьютерная графика. Полигональные модели. — М.: ДИАЛОГ-МИФИ, 2001. 464 с
12. Шикин Е.В., Боресков А.В. Компьютерная графика. Динамика, реалистичные изображения. — М.: ДИАЛОГ-МИФИ, 1995 – 288 с.
13. Ласло М. Вычислительная геометрия и компьютерная графика на C++ — М.: издательство БИНОМ», 1997. — 304 с: ил.
14. Порев В.Н. Компьютерная графика. — СПб.: БХВ-Петербург, 2002. — 432 с: ил.
15. Эйнджел, Эдвард. Интерактивная компьютерная графика. Вводный курс на базе OpenGL, 2 изд.: — М.: Издательский дом "Вильяме", 2001. — 592 с
16. http://lib.ifmo.ru — библиотека СПбГУ ИТМО
17. http://msdn.com — библиотека и база знаний по разработке компании Microsoft
18. http://gamedev.ru — русскоязычный ресурс, посвященный разработке трехмерных приложений
19. http://www.twirpx.com/files/informatics/cgraph/ — сборники статей, лекций, презентаций и книг по компьютерной графике