В.В. СЕЛЯНКИН

КОМПЬЮТЕРНАЯ ГРАФИКА

ТАГАНРОГ
2009

Рецензенты:

Заведующий кафедрой САУ ТТИ ЮФУ, д-р техн. наук

Финаев В.И.

Зам. директора ЮРКЦ «Земля» канд. техн. наук

Калачев Д.П.

Селянкин В.В.

Компьютерная графика: Учебное пособие. – Ростов-на-Дону, 2009. – с.

В учебном пособии рассматриваются алгоритмы растровой графики, служащие основой для построения изображений на растровом экране, математические преобразования точки на плоскости и в пространстве, задача двумерного отсечения, а также методы программирования видеоадаптеров. Пособие рассчитано на студентов и аспирантов, специализирующихся в области разработки программного обеспечения.

ВВЕДЕНИЕ

Настоящее учебное пособие предназначено для изучения методов и алгоритмов построения изображений, их обработки и анализа. В основе работы с изображением лежат операции с точками на растровом экране. Построение изображения связано с изучением алгоритмических, математических и аппаратных средств компьютерной графики.

В первом разделе приводятся алгоритмы растровой развертки отрезков, окружностей, эллипсов, заполнения сплошных областей, многоугольников, круга, а также алгоритмы обработки фрагментов изображений и создания растровых шрифтов. По некоторым алгоритмам приводятся фрагменты программ их реализации. Особое внимание уделяется построению эффективных алгоритмов.

Во втором разделе дается математический аппарат преобразований точки на плоскости и основные сведения по элементарным преобразованиям – переносу, масштабированию, повороту изображений и их композициям. Приводится понятие однородных координат, и даются матричные операции для выполнения указанных преобразований в однородных координатах. Излагается методика определения взаимного положения прямой и точки, с помощью которой решается задача определения положения точки относительно многоугольников.

В третьем разделе рассматривается задача двумерного отсечения для отрезков и областей в виде многоугольников как выпуклых, так и невыпуклых. Отсечение может выполняться ортогональными окнами или произвольными многоугольниками. Даются методы вычисления нормалей к ребрам многоугольников и определения выпуклости многоугольников.

В четвертом разделе рассматриваются вопросы архитектуры видеоадаптеров, методы их программирования, возможности прямого обращения к видеопамяти, а также графические функции базовой системы ввода-вывода. Дается краткое описание стандарта VESA и перечень функций для видеоадаптеров SVGA.

В пятом разделе предлагается математический аппарат работы с точкой в пространстве, приводятся уравнения прямой и плоскости в виде, удобном для работы с графическими изображениями. Даются

понятия построения трехмерных изображений на плоском экране, параллельных и центральных проекций. Все преобразования излагаются в однородных координатах.

ОБЩИЕ ПОЛОЖЕНИЯ

Предметом изучения данного учебного пособия является графика, создаваемая программными средствами на растровом экране компьютера. При этом могут рассматриваться три различных **задачи**: синтеза, анализа и обработки изображений. Первая задача предполагает создание изображения по заданному его описанию. Вторая задача обратная по отношению к первой: нужно сделать описание изображения по заданному изображению. Наконец, третья задача предполагает построение нового изображения по заданному исходному изображению. В последнем случае используются различные методы переработки изображения, которые применяются по некоторому заданию, определяющему конечный вид изображения.

Методы дисциплины заключаются в использовании алгоритмов построения изображений на растровом экране, учета аппаратных особенностей видеосистемы компьютера и математического аппарата преобразования точки на плоскости и пространстве в однородных координатах.

Освоение методов построения изображений предполагает определенные знания и навыки в области программирования на машинно-ориентированных языках, языках высокого уровня под операционными системами DOS и Windows. Кроме того, требуются знания некоторых разделов математики и информатики.

Модуль № 1
Алгоритмы растровой графики и преобразования точки на плоскости

Комплексная цель

Определить основные алгоритмы построения различных примитивов на растровом экране: отрезка, окружности, эллипса, заполнения сплошных областей, многоугольников, круга, обработки фрагментов изображений и построения растровых и векторных шрифтов. Дать навыки особенностей программирования растровых изображений и построения эффективных методов решения этих задач.

Содержание модуля

1. АЛГОРИТМЫ РАСТРОВОЙ ГРАФИКИ

1.1. Особенности растровой графики

Среди многообразия возможностей, предоставляемых современными вычислительными средствами, компьютерная графика, ориентированная на пространственно-образное мышление человека, занимает особое место. Ее методы и средства компьютерной графики представляют собой эффективный инструмент при выполнении проектно-конструкторских, научно-исследовательских, оформительских работ, а также всех случаев визуализации различных объектов. При этом наблюдается взаимное повышение возможностей, как человека, владеющего эффективным инструментом, так и компьютера, обогащенного новыми возможностями. В этом смысле значительную роль играет разработка соответствующих алгоритмов, отличающихся качественными параметрами по быстродействию, объемам используемой памяти и спектру предоставляемых возможностей.

Основой компьютерной графики являются методы и алгоритмы растровой графики, которые позволяют строить любые изображения на растровых дисплеях, использующих прямоугольную матрицу точек (пикселей). Каждый пиксель может изображаться некоторым цветом,

выбранным из имеющейся палитры цветов. Для реализации любого изображения в компьютере имеется видеоадаптер, хранящий в своей видеопамяти изображение и обеспечивающий регенерацию изображения на экране со скоростью порядка 50 раз в секунду. Размер палитры определяется объемом видеопамяти, отводимой под один пиксель, и зависит от типа видеоадаптера. Для ПЭВМ типа IBM существуют несколько видеоадаптеров, отличающихся возможностями, своим устройством и принципами работы с ними: Hercules, CGA, EGA, VGA, SVGA. Почти все указанные адаптеры поддерживают несколько режимов работы, которые отличаются друг от друга размерами матрицы пикселей (разрешающей способностью изображений) и размером палитры. Как правило, развитие видеоадаптеров строится по принципу совместимости с предыдущими моделями, однако при этом могут иметь место некоторые особенности, нарушающие это положение.

Формирование изображения базируется на работе с отдельными пикселями, однако пользователю может предоставляться достаточно широкий набор библиотечных примитивов, которые повышают эффективность работы за счет операций с графическими объектами. Такими объектами могут выступать линии, дуги, окружности, сложные кривые, сплошные объекты, шрифты, картинки и т.д. Как правило, каждый язык программирования имеет свою графическую библиотеку, обеспечивающую работу с элементами и группами графических объектов и поддерживающую работу с основными типами видеоадаптеров.

Особенности растровой графики связаны с тем, что обычные изображения, с которыми сталкивается человек в своей деятельности (чертежи, графики, карты, художественные картины и т.п.), реализованы на плоскости, состоящей из бесконечного набора точек. Экран же растрового дисплея представляется матрицей дискретных элементов, имеющих конкретные физические размеры. При этом число дискретных элементов ограничено. В связи с этим нельзя провести точную линию из одной точки в другую, а можно выполнить только аппроксимацию этой линии с отображением её на дискретной матрице.

Для большей определенности следует ввести понятие дискретной плоскости, имеющей целочисленные координаты. Такую плоскость также называют целочисленной решеткой, растровой плоскостью или растром. Эта решетка представляется квадратной сеткой с шагом 1. Узлы целочисленной решетки являются центрами соответствующих квадратных ячеек сетки. Таким образом, узлы растра окружены "единичными" квадратными окрестностями "радиуса" 1/2. При обращении к точке растра с координатами (i,j) выполняется инициализация единичного квадрата с закрашиванием его соответствующим цветом.

Отображение любого объекта на целочисленную решетку называется разложением его в растр или просто растровым представлением. Естественно, это разложение лишь приблизительно представляет изображаемый объект. Отображение является неоднозначным ввиду зависимости его от алгоритмов аппроксимации, а они, в свою очередь, - от критериев определения точек растра. В связи с этим возникает задача выбора оптимального алгоритма по соответствующему критерию.

1.2. Растровая развертка отрезков

При анализе конкретных алгоритмов рисования отрезков необходимо рассмотреть общие требования к таким алгоритмам и определить критерии их оценок. Естественно, ставится задача рисования отрезков таким образом, чтобы они выглядели прямыми и начинались и заканчивались в заданных точках. Важное значение имеет яркость изображения отрезка, которая должна быть постоянной вдоль всего отрезка и не зависеть от его наклона. Кроме того, вывод изображения должен быть выполнен с высокой скоростью. Каждый из перечисленных параметров выполним в определенных границах, определяемых параметрами экрана, быстродействием компьютера, его объемами памяти и выбранным алгоритмом. Улучшение перечисленных характеристик приводит к более полному удовлетворению указанных требований.

Постоянная яркость вдоль всего отрезка достигается лишь при проведении горизонтальных и вертикальных линий или под углом 45 градусов. Кроме того, эти отрезки, имея постоянную яркость, различаются между собой степенью яркости: два первых имеют

наибольшую, а третий - наименьшую. Для всех других вариантов расположения отрезки будут иметь неравномерную и различную яркость. Обеспечение одинаковой яркости вдоль отрезков разных длин и ориентаций требует извлечения квадратного корня или использования адаптеров, дающих набор полутонов, что существенно замедляет процесс вычисления. Обычно реализация разложения отрезка в растр выполняется с выбором некоторого компромисса рассмотренных вариантов, при котором длина отрезка определяется приближенно, вычисления сводятся к некоторому минимуму за счет использования целочисленной арифметики с исключением операций деления и умножения, реализации алгоритмов на аппаратном или микропрограммном уровне.

Для некоторого отрезка M_1M_2 с координатами концов (x_1,y_1) и (x_2,y_2) можно построить **простой пошаговый алгоритм**, использующий уравнение отрезка прямой следующего вида:

$$y = y_1 + k\,(x - x_1),$$

где $k = (y_2 - y_1)/(x_2 - x_1)$, $x_1 \le x \le x_2$.

В простейшем случае предполагается, что коэффициент **k** неотрицателен и не превосходит единицы. Перебирая значения **x** в пределах от $\mathbf{x_1}$ до $\mathbf{x_2}$ с определенным шагом, можно вычислить соответствующие значения **y**, которые будут отвечать условию $y_1 \le y \le y_2$ и прорисовать полученные точки (x,y). Для простоты шаг прорисовки Δx можно принять равным единице.

Алгоритм генерации растровой развертки отрезка при указанном алгоритме будет иметь следующий вид:

```
Dx:= 1; Dy:= abs((y2 - y1)/(x2 - x1));
x:= x1; y:= y1;
for i := 0 to L-1 do
begin
    PutPixel (x,round(y));
    x := x + Dx; y := y + Dy;
end.
```

Реализация этой процедуры сопряжена со следующими проблемами. Если угол наклона отрезка прямой к оси абсцисс больше 45^0, то приращение отрезка за один шаг по оси ординат превышает соответствующее приращение по оси абсцисс. Это

приводит к тому, что при $\Delta x = 1$ приращение $\Delta y > 1$, и линия отрезка будет иметь разрывы. Кроме того, реализация операций деления и округления требует обработки вещественных чисел, что приводит к уменьшению быстродействия. Первую проблему можно решить симметричной заменой координат x и y, вторую - соответствующей корректировкой алгоритма.

Более совершенным, с точки зрения эффективности реализации, является алгоритм **цифрового дифференциального анализатора**. Он основывается на том, что для прямой можно записать соотношение

$$\frac{dy}{dx} = const \text{ или } \frac{\Delta y}{\Delta x} = const .$$

Константа в правой части уравнения представляет собой коэффициент наклона прямой к оси абсцисс. Поэтому можно записать

$$\frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}.$$

Отсюда

$$\Delta y = \frac{y_2 - y_1}{x_2 - x_1}\Delta x.$$

Алгоритм строится с использованием итерационного процесса вычисления текущих значений координат точки по предыдущим

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

Если принять приращение по оси x, равным единице, что вполне обоснованно для целочисленной матрицы пикселей, то последнее выражение можно записать как

$$x_{i+1} = x_i + 1;$$
$$y_{i+1} = y_i + \Delta y;$$
$$\Delta y = \frac{y_2 - y_1}{x_2 - x_1}.$$

Как уже говорилось, результат построения прямой зависит от соотношения приращений координат текущей точки по осям **x** и **y**. При соотношении $\Delta y > \Delta x$ и выборе шага $\Delta x=1$ приращение $\Delta y > 1$, что ведет к появлению разрывов в построении линий. В этом случае

следует изменить стратегию приращений таким образом, чтобы большее приращение получало значение, равное единице, а меньшее выбиралось равным соответствующему коэффициенту наклона прямой. Тогда все октанты координатной плоскости разбиваются на две группы. К первой группе относятся 1, 4, 5 и 8 октанты. Для них расчет текущих значений координат точек отрезка вычисляется по выражениям, приведенным выше. А для второй группы (2, 3, 6 и 7 октанты) расчет идет по следующим выражениям:

$$y_{i+1} = y_i + 1;$$

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

$$\Delta x = \frac{x_2 - x_1}{y_2 - y_1}.$$

В соответствии с приведенными вычислениями реализация алгоритма может быть представлена следующим образом.

```
var
x1,y1,x2,y2: integer;
i,l:          integer;
x,y,dx,dy:    real;
{вычисляем большую из длин l отрезка по оси x и y}
if    abs (x2 - x1) ≥ abs (y2 - y1)
then  l:= abs (x2 - x1)
else   l:= abs (y2 - y1);
{определяем приращения по оси x и y, полагая большее из них равным единице}
dx:= (x2 - x1)/l;
dy:= (y2 - y1)/l;
{начинаем с точки (x,y)}
x:= x1;
y:= y1;
{цикл по большей длине отрезка}
i:= 1;
while (i ≤ l)
PutPixel (Round(x),Round(y));
x:= x + dx;
y:= y + dy;
 i:= i + 1;
```

end while

Данный алгоритм, как и предыдущий, содержит операции деления и округления, что, кроме снижения скорости реализации, может приводить к накоплению погрешности, в результате которой последняя точка может отличаться от заданной.

Более эффективный алгоритм был предложен Брезенхемом (Bresenham) для графопостроителей, который используется теперь для растровых дисплеев. Алгоритм позволяет выбор оптимальных растровых координат для представления отрезка. Идея алгоритма заключается в том, что одна координата изменяется на единицу, а другая - либо не изменяется, либо изменяется на единицу в зависимости от расположения соответствующей точки от ближайшего узла координатной сетки.

Расстояние от точки отрезка до ближайшего узла по соответствующей ортогональной координате называется ошибкой. Алгоритм организован таким образом, что для вычисления второй координаты требуется только определять знак этой ошибки. Для первого октанта это показано на рис. 1.1.

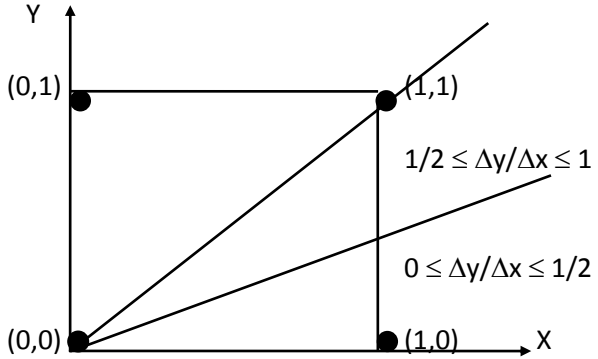


Рис. 1.1

Величину ошибки Δ можно определить в соответствии со следующим выражением:

$$\Delta = y_{уз} - y_{от},$$

где $y_{уз}$ - y-координата ближайшего узла для x = 1,

$y_{от}$ - y-координата точки отрезка для x = 1.

Если при x = 1 y-координата точки отрезка равна 1/2, то узлы координатной сетки (1,0) и (1,1) находятся на одинаковом расстоянии от отрезка и в качестве "ближайшего" выбирается узел (1,1). Таким образом, если $y_{от} \geq 1/2$, то $\Delta \geq 0$, в противном случае, при $y_{от} < 1/2$, получаем $\Delta < 0$. Для организации вычислений удобнее пользоваться смещенной величиной ошибки

$$d = \Delta - 1/2,$$

которая позволяет пользоваться не значением ошибки, а ее знаком.

При изменении координаты x на единицу величина d меняется на значение углового коэффициента:

$$d = d + \Delta y/\Delta x.$$

В соответствии с этим можно построить график изменения величины d, как показано на рис. 1.2. На каждом шаге алгоритма производится вычисление координаты x, величины d и выполняется анализ знака d. При этом, если окажется, что $d \geq 0$, то производится увеличение y на единицу, а значение d корректируется уменьшением на единицу. Тогда алгоритм аппроксимации отрезка в первом октанте будет иметь следующий вид:

```
x:= x1;
y:= y1;
dx:= x2 - x1;
dy:= y2 - y1;
d:= -1/2;
while x ≤ x2 do
PutPixel (x,y);
x:= x + 1;
d:= d + dy/dx;
if d ≥ 0 then
begin
y:= y + 1;
d:= d - 1
end
end while.
```

Представленный алгоритм использует вещественные числа и операцию деления. Оба недостатка можно исключить заменой величины d на другую, равную D = 2 dx d.

В данном случае меняется значение смещенной ошибки. Однако для алгоритма важно не само абсолютное значение d, а ее знак, который меняется синхронно с изменением знака D. Эффективность алгоритма можно повысить за счет замены операции умножения на 2 сдвигом на один разряд и вынесения сдвиговых операций из оператора цикла.

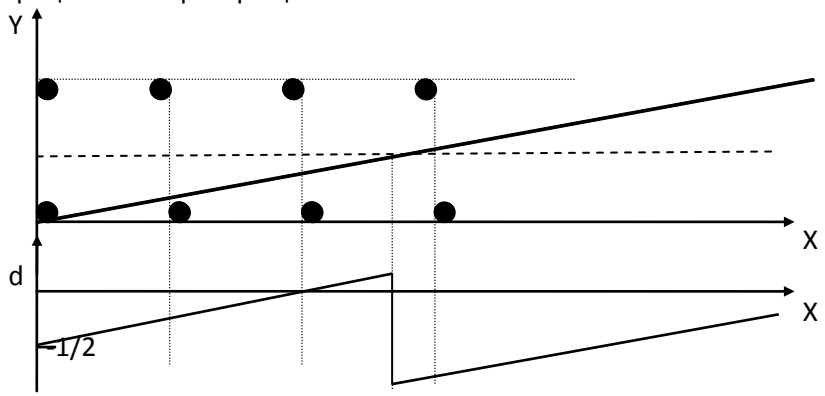


Рис. 1.2

Тогда алгоритм принимает окончательный вид:

```
x:= x1;
y:= y1;
dx:= x2 - x1;
dy:= y2 - y1;
D:= -dx;
DX:= dx shl 1;
DY:= dy shl 1;
while x ≤ x2 do
PutPixel (x,y);
x:= x + 1;
D:= D + DY;
if D ≥ 0 then
begin
y:= y + 1;
D:= D - DX
end
```

end while.

Оценивая достоинства полученного алгоритма, можно отметить, что он выполняет оптимальную аппроксимацию отрезка, используя при этом целочисленную арифметику и минимальное количество операций сложения и вычитания. Алгоритм можно использовать и для других октантов плоскости по схеме показанной на рис. 1.3. где отмечены координаты, подлежащие изменению на единицу в соответствующих октантах.

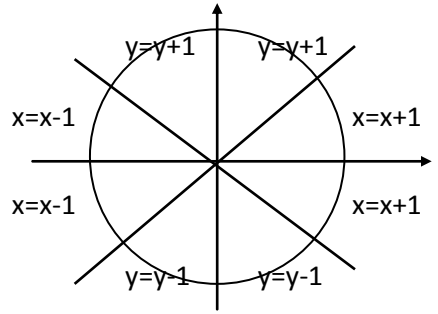


Рис. 1.3

При реализации общего алгоритма растровой развертки отрезков следует учитывать отличие экранной системы координат от декартовой. Считается, что координаты отрезка заданы в декартовой системе, начало которой располагается в центре экрана. Для правильной работы алгоритма необходимо предусмотреть программный перевод этих координат в экранную систему.

1.3. Растровая развертка окружности

Наиболее простой способ определения точек окружности можно получить решением уравнения окружности

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

относительно одной координаты. Например, изменяя x от -R до +R, можно вычислить y из соотношения

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

При этом, однако, требуется операция вычисления корня квадратного, что приводит к значительному объему вычислений, а

также к появлению разрывов окружности по мере приближения к точкам x = -R и x = +R.

Несколько лучший вариант получается при использовании полярных координат

$$x = R * \cos\varphi,\ y = R * \sin\varphi.$$

Здесь изменение угла в пределах от 0^0 до 360^0 позволяет вычислять значения координат x и y. Основная трудность использования этого метода заключается в вычислении тригонометрических функций угла. Кроме того, необходимы определенные усилия по выбору шага изменения угла, который должен быть переменным для того, чтобы избежать разрывов. Либо шаг изменения сделать постоянным, но со значением равным самому минимальному шагу. В этом случае возрастает объём вычислений.

Возможен еще один способ получения окружности путем аппроксимации ее вписанным многоугольником, при котором соседние вычисляемые точки соединяются отрезками прямых. В зависимости от требуемой точности построения окружности можно устанавливать соответствующее число сторон многоугольника, выбирая компромиссное решение между точностью и сложностью вычислений.

При построении окружности любым способом можно использовать методы повышения эффективности алгоритма. Например, достаточно вычислять точки только одного (например, первого) октанта, по которым остальные точки окружности получают за счет симметрии их расположения относительно четырех осей, как показано на рис. 1.4.

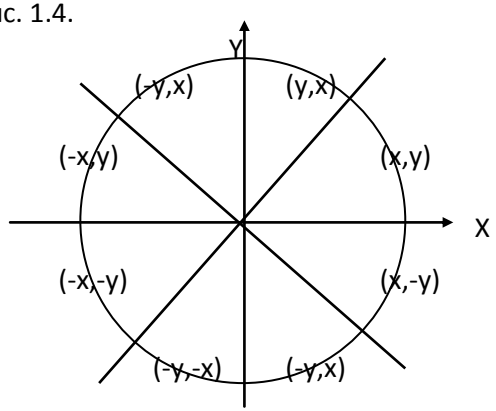


Рис. 1.4

При этом каждая вычисленная точка (x,y) позволяет получить еще 7 других за счет простановки соответствующих знаков перед координатами: (-x,y), (x,-y), (-x,-y), (y,x), (-y,x), (y,-x), (-y,-x).

Наиболее эффективным по скорости алгоритмом построения окружности является алгоритм Брезенхема. Он же дает и высокую точность построения точек окружности. Основан он на пошаговом методе их вычисления. Рассмотрим алгоритм построения окружности в первом октанте. В этом случае координаты окружности имеют следующие пределы изменения:

$$R/\sqrt{2} \le x \le R;$$

$$0 \le y \le R/\sqrt{2}.$$

На рис. 1.5 показана часть окружности, расположенная в первом октанте. В этом октанте начальной точкой является точка с координатами (R,0), конечной – точка ($R/\sqrt{2}$, $R/\sqrt{2}$).

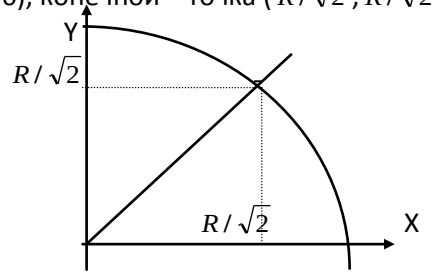


Рис. 1.5

При таком порядке вычисления координат точек окружности в первом октанте координата x убывает, а координата y возрастает. Перемещение от предыдущего узла растра к последующему возможно в двух вариантах, которые можно обозначить как "движение a" и "движение b" (рис. 1.6).

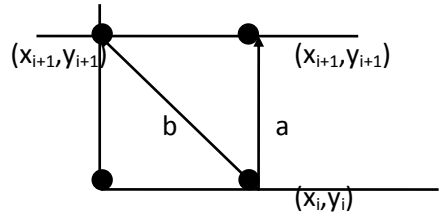


Рис. 1.6

В соответствие с этим можно описать указанные движения по изменению их координат:

движение a: $x_{i+1} = x_i$;

$y_{i+1} = y_i + 1$;

движение b: $x_{i+1} = x_i - 1$;

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

При выборе того или иного движения руководствуются соображениями получения наименьшей ошибки, которая определяется как разность между координатами точек растра и точек окружности:

$$\Delta_i = x_i + y_i - R^2.$$

Выполним вычисления этой ошибки для обоих движений. Сначала выполним вычисление для первого движения:

$$\Delta^a_{i+1} = x^2_{i+1} + y^2_{i+1} - R^2 = x_i^2 + (y_i + 1)^2 - R^2 = \Delta_i + 2 * y_i + 1.$$

Аналогичное вычисление можно выполнить и для другого движения:

$$\Delta^b_{i+1} = x^2_{i+1} + y^2_{i+1} - R^2 = (x_i - 1)^2 + (y_i + 1)^2 - R^2 =$$
$$= \Delta_i - 2 * x_i + 2 * y_i + 2.$$

Для анализа погрешности Δ_{i+1} вычислим разность погрешностей, получающихся при движении “a" и при движении “b":

$$d_{i+1} = \left|\Delta^a_{i+1}\right| - \left|\Delta^b_{i+1}\right|.$$

Выбор оптимального движения теперь можно выполнить по знаку величины d_{i+1}. Так, если $d_{i+1} < 0$, то следует выполнять движение “a", в противном случае - движение “b". Вычисление d_{i+1} можно заменить более простым выражением

$$d_{i+1} = \Delta^a_{i+1} + \Delta^b_{i+1},$$

сохранив логику выбора движения. Для подтверждения справедливости данного утверждения рассмотрим три случая взаимного расположения точек растра и точек окружности.

Первый случай. Узлы растра для движениях «a» и «b» лежат по разные стороны окружности: узел растра при движении "a" – вне окружности, а при движении "b" - внутри. Тогда

$$\Delta^a_{i+1} \geq 0; \Delta^b_{i+1} < 0.$$

Если при этом узел растра при движении «a» лежит ближе к окружности, тогда

$$\Delta^a_{i+1} + \Delta^b_{i+1} < 0,$$

так как

$$\left|\Delta^a_{i+1}\right| < \left|\Delta^b_{i+1}\right|$$

и меньшая погрешность положительна, а большая - отрицательна. В случае, когда ближе к окружности лежит узел при движении «b», то указанная сумма будет больше нуля по аналогичным соображениям. Таким образом, при $d_{i+1} < 0$, выбирается движение «a» с выбором узла, наиболее близко расположенного к окружности. В противоположном случае – движение «b».

Второй случай. Обе точки растра расположены внутри окружности. Следовательно,

$$\Delta^a_{i+1} < 0; \Delta^b_{i+1} < 0.$$

Тогда

$$\Delta^a_{i+1} + \Delta^b_{i+1} < 0,$$

и точка «a» всегда ближе к окружности по определению. Значит, следует выбирать движение «a».

Третий случай. Обе точки растра лежат снаружи окружности. Поэтому

$$\Delta^a_{i+1} \geq 0; \Delta^b_{i+1} \geq 0.$$

Этот случай симметрично противоположен второму - сумма отрицательна, точка "b" ближе к окружности и выбирается движение “b".

С учетом сказанного и ранее полученных выражений можно записать

$$d_{i+1} = \Delta_i + 2 * y_i + 1 + \Delta_i - 2 * x_i + 2 * y_i + 2 =$$
$$= 2 * \Delta_i - 2 * x_i + 4 * y_i + 3.$$

Последнее выражение позволяет построить процедуру выбора оптимальных точек растра (x_{i+1},y_{i+1}), дающих минимальную погрешность, по предыдущим значениям координат точки (x_i,y_i), значению Δ_i и знаку d_{i+1}. Эффективность процедуры можно повысить за счет организации вычисления погрешности d_{i+1} по предыдущему значению d_i. Для этого выберем начальную точку построения окружности, например, с координатами (R,0). Тогда

$d_{i+1} = 2*0 - 2*R + 4*0 + 3 = 3 - 2*R.$

Имея значение d_{i+1} , можно вычислить d_{i+2} для движения “a" и движения “b".

$$d^a_{i+2} = 2*\Delta^a_{i+1} - 2*x^a_{i+1} + 4*y^a_{i+1} + 3 =$$
$$= 2*(\Delta_i + 2*y_i + 1) - 2*x_i + 4*(y_i + 1) + 3 =$$
$$= d_{i+1} + 4*y_i + 6.$$
$$d^b_{i+2} = 2*\Delta^b_{i+1} - 2*x^b_{i+1} + 4*y^b_{i+1} + 3 =$$
$$= 2*(\Delta_i - 2*x_i + 2*y_i + 2) - 2*(x_i - 1) + 4*(y_i + 1) + 3 =$$
$$= d_{i+1} - 4*x_i + 4*y_i + 10.$$

Таким образом, находясь в точке i, можно вычислить d_{i+1} , определить его знак и выбрать соответствующее движение. Затем вычисляется d_{i+2}, а по его знаку выполняется выбор очередного движения. Вычисление d_{i+2} и выбор движения производятся в цикле до конечной точки окружности первого октанта.

В соответствии с изложенным можно записать алгоритм.

```
y:= 0;
x:= R;
d:= 3 - 2*R;
while x > y do
begin {вывести точки (x,y); (-x,y); (x,-y); (-x,-y);
                    (y,x); (-y,x); (y,-x); (-y,-x);}
if d < 0 then {движение a}
  d = d + 4*y + 6
else        {движение b}
begin
d = d + 4*(y - x) + 10;
x = x - 1
```

end
y = y + 1
end
if x = y then {вывести точки (x,y); (-x,y); (x,-y); (-x,-y);}.

Оценивая достоинства данного алгоритма, можно отметить, что здесь используется целочисленная арифметика, операции сложения, вычитания и сдвига. Недостатком является невозможность работы непосредственно с видеопамятью, т.к. приходится выводить сразу 8 точек, расположенных в разных местах экрана. Однако можно построить процедуру пошаговой развертки всей окружности, если внести соответствующие коррективы в вычислении координат и ошибки в различных октантах.

По такой же стратегии можно организовать алгоритм растровой развертки эллипса, при которой отклонение выбранной точки узла будет определяться по отношению к точке, принадлежащей эллипсу, вычисляемой по формуле

$$a^2x^2 + b^2y^2 = a^2b^2.$$

Растровая развертка дуги может быть выполнена по алгоритму Брезенхема. При этом параметры ее могут задаваться координатами центра окружности, которой принадлежит данная дуга, и координатами ее концов или начальным и конечным углами концов дуги. При построении растровой развертки дуги возможен вариант того, что начальная точка не совпадает с точкой растра, поэтому начальная погрешность отлична от нуля, что следует учитывать в процедуре реализации. Кроме того, необходимо точно определить момент окончания вывода точек, чтобы не оказалось лишних или недостающих элементов.

1.4. Заполнение сплошных областей

Сплошные области могут ограничиваться отрезками прямых (многоугольники) или произвольным контуром. В связи с этим область может задаваться своими вершинами или ребрами для многоугольников или контуром, представленным его растровой разверткой. Генерация сплошных областей носит название растровой развертки сплошных областей и многоугольников или заполнением сплошных областей и многоугольников. Заполнение областей может

быть выполнено двумя способами - сканированием строк и затравочным заполнением.

Метод сканирования строк решает задачи определения порядка сканирования, принадлежности точки внутренней области многоугольника или контура. При затравочном заполнении предполагается наличие некоторой точки, принадлежащей внутренней области (затравочная точка), для которой определяют ее соседние точки и включают ее в список других затравочных точек. Список этих точек анализируется с точки зрения необходимости их закрашивания.

Методы сканирования пригодны как для растровых дисплеев, так и для векторных, а методы затравочного заполнения - только для растровых. Области могут задаваться внутренне определенным и гранично-определенным способом. Внутренне определенная область состоит из точек только одного цвета или интенсивности. Пиксели, внешние по отношению к точкам внутренне определенной области, имеют отличную от них интенсивность или цвет (рис. 1.7).

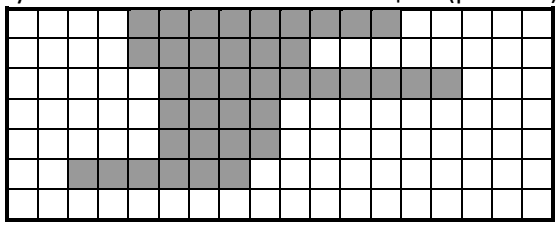


Рис. 1.7

Гранично-определенные области имеют контур, состоящий из пикселей с выделенным значением цвета или интенсивности. Внутренние пиксели области отличны от них по цвету или интенсивности, а внешние могут с ними совпадать (рис. 1.8).

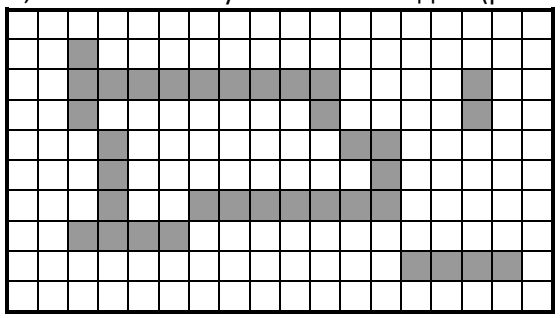


Рис. 1.8

Внутренне и гранично-определенные области могут быть четырех- и восьмисвязными. В четырехсвязных областях любой пиксель достигается движением по четырем взаимно перпендикулярным направлениям, а восьмисвязные - по восьми: горизонтальным, вертикальным и диагональным. На рис. 1.9 показаны внутренне и гранично-определенные четырехсвязные области.

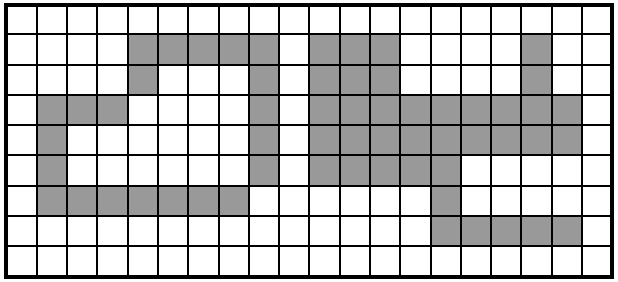


Рис. 1.9

На рис. 1.10 представлены внутренне и гранично-определенные восьмисвязные области.

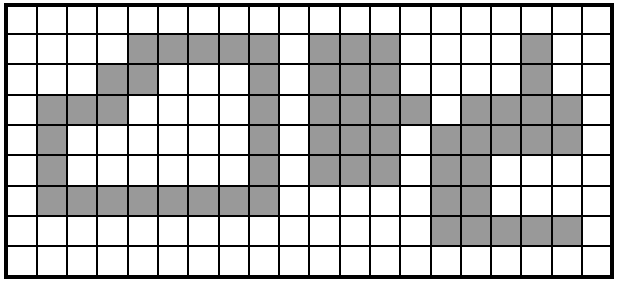


Рис. 1.10

Для восьмисвязных гранично-определенных областей необходимо, чтобы граница отвечала требованию четырехсвязности. Здесь граница рассматривается как внутренне определенная область. При невыполнении этого условия граница не выполняет своей функции разделения плоскости на внешнюю и внутреннюю области. Так как через восьмисвязную область можно переходить к любой точки плоскости. Следует также отметить, что алгоритмы рисования восьмисвязных областей применимы и для четырехсвязных.

Обратное несправедливо, т.е. алгоритмы четырехсвязных областей неприменимы для восьмисвязных. Допустимо, чтобы границы четырехсвязных гранично-определенных областей были восьмисвязны.

Алгоритмы заполнения сплошных областей в основном зависят от вида области. Можно сформировать простейший алгоритм затравочного заполнения **четырехсвязной гранично-определенной области** в следующем виде.

```
var
x,y:    integer; {координаты затравочной точки}
oldcolor:  word; {исходное значение цвета}
newcolor:  word; {новое значение цвета}
framcolor:  word; {цвет границы}
begin
помещаем пиксель в Stack;
while (Stack не пуст) do
begin
извлекаем пиксель из Stack;
PutPixel(x,y,newcolor);
для точек (xт,yт), соседних с (x,y):
(x+1,y); (x,y+1); (x-1,y); (x;y-1), проверяем условие:
if GetPixel(xт,yт) ≠ newcolor and
GetPixel(xт,yт) ≠ framcolor
then помещаем (xт,yт) в Stack;
end
end.
```

Аналогично строится алгоритм и для **внутренне определенной области**. В этом случае меняется только условный оператор, который должен проверять принадлежность соседних точек и наличие неизмененного цвета.

Тогда внешний блок описанной процедуры будет иметь вид

```
begin
помещаем пиксель в Stack;
while (Stack не пуст) do
begin
извлекаем пиксель из Stack;
```

PutPixel(x,y,newcolor);
для точек ($x_т$,$y_т$), соседних с (x,y):
(x+1,y); (x,y+1); (x-1,y); (x;y-1), проверяем условие:
if GetPixel($x_т$,$y_т$) = oldcolor
then помещаем ($x_т$,$y_т$) в Stack;
end
end.

Предложенные алгоритмы легко обобщаются на случай **восьмисвязных областей**, для которых необходимо выполнять анализ восьми соседних точек.

Процедура затравочного заполнения областей, описанная выше, обладает двумя недостатками: стек заполняется большим количеством повторяющихся точек, что приводит к существенным временным затратам при их анализе и, как следствие, требуется стек большого объема. Однако эти проблемы легко разрешить изящным методом перестановки местами операций помещения точек в стек и их закраски, получив таким образом **более эффективный алгоритм**. Тогда процедура будет записана следующим образом:

begin
PutPixel(x,y,newcolor);
помещаем пиксель в Stack;
while (Stack не пуст) do
begin
извлекаем пиксель (x,y) из Stack;
для точек ($x_т$,$y_т$), соседних с (x,y):
(x+1,y); (x,y+1); (x-1,y); (x;y-1), проверяем условие:
if GetPixel($x_т$,$y_т$) ≠ newcolor and
GetPixel($x_т$,$y_т$) ≠ framcolor
then
begin
PutPixel($x_т$,$y_т$,newcolor);
помещаем ($x_т$,$y_т$) в Stack
end
end
end.

Другой метод повышения эффективности затравочного заполнения областей заключается в организации **построчного заполнения непрерывного интервала** на сканирующей строке с одной затравочной точкой. Непрерывный интервал точек образуется примыкающими друг к другу соседними пикселями и ограничивается либо уже заполненными точками, либо границами области. Алгоритм применим для произвольных четырехсвязных гранично-определенных областей сплошных или содержащими дыры. При этом должно соблюдаться условие отсутствия во внешней области, примыкающей к границе, пикселей с цветом заполнения.

Идею алгоритма можно сформулировать следующим образом:

1. Затравочная точка (x,y) помещается в стек.

2. Если стек не пуст, то извлекается точка из стека, и влево и вправо от нее формируется непрерывный интервал до границы области, в противном случае - переход к п. 5.

3. Запоминается крайняя левая точка интервала xl и крайняя правая – xr

4. В диапазоне интервала $xl \le x \le xr$ в строках (y+1) и (y– 1) отыскиваются незаполненные пиксели, образующие непрерывные интервалы. В каждом интервале крайний правый пиксель считается затравочным и помещается в стек. Переход к п. 2.

5. Конец работы алгоритма

Алгоритм растровой развертки многоугольников основан на построчном сканировании многоугольника. При этом производится заполнение многоугольника по сканируемым строкам. Рассмотрим пример растровой развертки многоугольника (рис. 1.11).

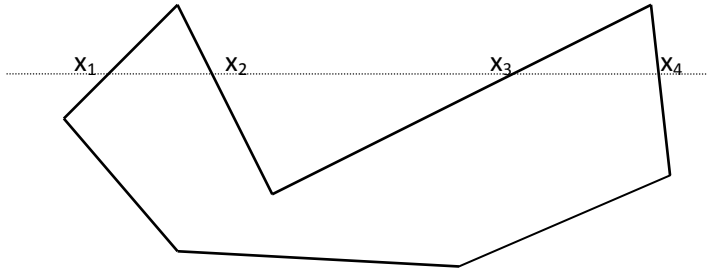


Рис. 1.11

Сканирующая строка разбивается многоугольником на пять интервалов в точках ее пересечения со сторонами многоугольника. Необходимо выделить на сканирующей строке точки, принадлежащие внутренней области многоугольника. Из рисунка видно, что внутри многоугольника лежат точки интервалов (x_1-x_2) и (x_3-x_4). Если вычислить x-координаты точек пересечения сканирующей строки с ребрами многоугольника и упорядочить их по возрастанию, то нечетные интервалы будут лежать во внутренней области многоугольника. Тогда процедура сканирования и поиска точек закрашивания может иметь следующий вид:

1. Находятся x-координаты всех точек пересечения сканирующей строки с ребрами многоугольника.

2. X-координаты упорядочиваются по возрастанию.

3. Закрашиваются точки в нечетных интервалах.

Однако алгоритм требует уточнения для двух частных случаев.

Первый случай. Пересечение сканирующей строки с горизонтальным ребром (рис. 1.12).

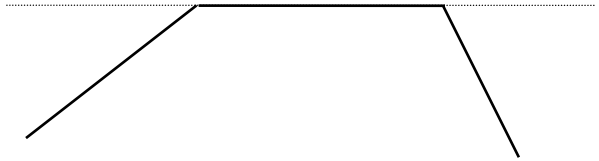


Рис. 1.12

В этом случае имеется бесконечное множество точек пересечения ребра со сканирующей строкой. В связи с этим понятие четных и нечетных интервалов теряет смысл. Для решения задачи следует удалить горизонтальное ребро, а пересечения сканирующей строки искать с ребрами смежными ему. В соответствии с идеей алгоритма точки пересечения со смежными ребрами дадут интервал заполнения, совпадающий с удаленным ребром.

Второй случай. Пересечение сканирующей строки с вершиной многоугольника. При этом возможны два варианта - пересечение вершины, являющейся локальным экстремумом, и вершины, не являющейся таковой.

<u>В первом варианте</u> (рис. 1.13) сканирующая строка пересекает два ребра, пересечения с которыми дают две совпадающие точки -

саму вершину многоугольника. Требуется заполнить интервал длиной в одну точку. Этот случай

Рис. 1.13

является просто экстремальным. Здесь интервал заполнения вырожден в одну точку.

Во втором варианте (рис. 1.14) сканирующая строка дает три точки пересечения с ребрами многоугольника. Причем две из них также совпадают и являются вершиной многоугольника. После упорядочивания x-координат по возрастанию они образуют нечетный интервал, который подлежит заполнению. А интервал между вершиной и точкой пересечения с ребром оказывается четным, не подлежащим заполнению. Для устранения этой проблемы необходимо укоротить одно из смежных ребер, сходящихся в указанной вершине, на одну точку.

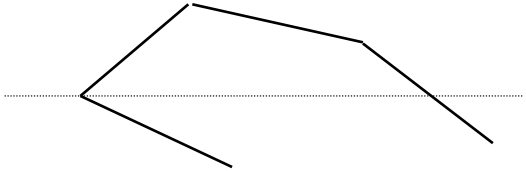


Рис. 1.14

Нахождение x-координат точек пересечения сканирующей строки с ребрами многоугольника можно выполнить по уравнению прямой. Процедуру нахождения этих точек можно организовать итерационным методом, для чего необходимо задать конечные точки ребер. Тогда вычисления будут выполняться в соответствии со следующими выражениями:

$$x_{i+1} = x_i + m;$$

$$m = \frac{x_{\text{ђ}} - x_{\text{,}}}{y_{\text{ђ}} - y_{\text{,}}},$$

где $x_к$,$y_н$,$x_к$,$y_н$ - координаты концевых точек ребра.

Для реализации алгоритма вводится понятие активных ребер. Это такие ребра многоугольника, которые пересекаются сканирующей строкой. При последовательном сканировании экрана сверху вниз список активных ребер меняется за счет добавления новых или удаления некоторых ребер из этого списка. Изменения списка происходят в моменты прохождения сканирующей строки через вершины ребер, причем прохождение через максимальную y-координату ребра делает его активным, а через минимальную y-координату - исключает это ребро из списка активных ребер.

Активные ребра содержатся в таблице активных ребер (ТАР), которая имеет следующую структуру записей для каждого ребра:

l - номер ребра,

y_{max} - максимальное значение y-координаты ребра,

x - текущее значение x-координаты пересечения сканирующей строки с ребром,

m - приращение x-координаты для данного ребра.

С учетом изложенного можно сформировать алгоритм обработки очередной сканирующей строки i.

1. Включить в ТАР ребра, для которых $i = y_{min}$.

2. Упорядочить записи ТАР по возрастанию значений x-координат.

3. Закрасить нечетные интервалы между точками пересечений.

4. Исключить из ТАР ребра, для которых $i = y_{max}$.

5. Вычислить $x = x + m$.

6. Получить новый номер сканирующей строки $i = i + 1$.

Кроме ТАР, существует еще таблица ребер (ТР), которая состоит из всех ребер данного многоугольника. Она имеет другую структуру записей:

l - номер ребра,

y_{min} - минимальная y-координата ребра,

y_{max} - максимальная y-координата ребра,

x_{min} - x-координата ребра, соответствующая координате y_{min} ,

m - приращение x-координаты для данного ребра.

ТР должна быть упорядочена по возрастанию y_{min}, а при наличии ребер с одинаковыми y_{min} они упорядочиваются по возрастанию x_{min} . Это упорядочивание позволяет при поиске

активных ребер сравнить номер i сканирующей строки со значением y_{min} первой записи ТР. Если при этом i < y_{min}, то добавляем новую запись в ТАР, а из ТР удаляем первую запись. Затем выполняется очередное сравнение с первой записью ТР. Окончательно алгоритм растровой развертки многоугольников будет следующим.

1. Перебор всех вершин многоугольника с укорочением одного из смежных ребер для вершин, не являющихся локальными экстремумами.
2. Поиск и удаление горизонтальных ребер.
3. Построение упорядоченной ТР.
4. Принимают первоначальное значение номера сканирующей строки i равной y_{min} первой записи ТР.
5. Создается пустая ТАР.
6. Выполняется обработка сканирующей строки.
7. Пока ТАР или ТР не пусты, выполняется переход к п. 6.
8. Окончание процедуры.

Эффективность полученного алгоритма можно повысить за счет закраски интервалов многоугольника через обращение к видеопамяти.

Растровую развертку круга можно построить с помощью алгоритма Брезенхема для первого октанта. При этом использование симметричности точек окружности позволяет организовать заполнение четырех интервалов со следующими координатами:

- первый интервал - (-y,x) - (y,x);
- второй интервал - (-x,y) - (x,y);
- третий интервал - (-x,-y) - (x,-y);
- четвертый интервал - (-y,-x) - (y,-x) (рис. 1.15).

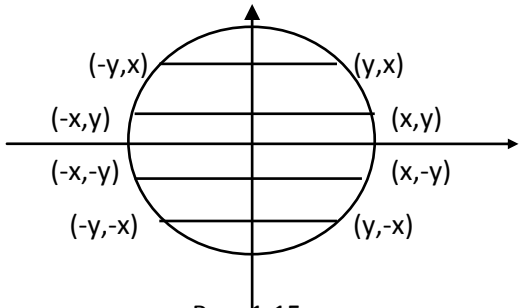


Рис. 1.15

Следует выделить граничные ситуации, когда

(x,y) = (R,0) и (x,y) =($R/\sqrt{2}$, $R/\sqrt{2}$).

В первом случае вместо четырех интервалов получают один интервал, совпадающий с горизонтальным диаметром - (0,R) -(0,-R), и две точки - (R,0) и (-R,0). Во втором случае получается только два интервала:

первый - (- $R/\sqrt{2}$, $R/\sqrt{2}$) - ($R/\sqrt{2}$, $R/\sqrt{2}$);

второй - (- $R/\sqrt{2}$,- $R/\sqrt{2}$) - ($R/\sqrt{2}$,- $R/\sqrt{2}$).

Кроме того, при реализации алгоритма необходимо учитывать выбор узла растра при движении “a" и движении “b", так как в первом случае меняется только координата y, а координата x остается неизменной. Во втором случае меняются обе координаты. Это приводит к тому, что интервалы, расположенные во 2, 3, 6 и 7 октантах - ((-y,x) - (y,x)) и ((-y,-x) - (-y,x)), при движении “a" совпадают с предыдущими интервалами по координате x, а по координате y больше на два пикселя - по одной точке с каждого конца интервала. Это приводит к тому, что приходится повторно заполнять интервал, а при выводе линии в инверсных режимах (стили XOR или NOT) ранее заполненный интервал будет стерт. С учетом сказанного алгоритм растровой развертки круга можно сформулировать в следующем виде:

```
y := 0;
x := R;
d := 3 - 2*R;
while x > y do
begin
заполняем интервалы  1, 4, 5, 8 октантах:
((-x,y) - (x,y)), ((-x,-y) - (x,-y));
if d < 0 then  {движение a}
d := d + 4*y + 6;
else
begin    {движение b}
заполняем интервалы в 2, 3, 6, 7 октантах:
((-y,x) - (y,x)), ((-y,-x) - (y,-x));
d := d + 4*(y - x) + 10;
x := x - 1
```

```
end
y := y + 1;
end
```

1.5. Обработка фрагментов изображений. Растровые шрифты

Обработка изображений может сопровождаться с необходимостью выполнения операций с отдельными их частями - фрагментами, которые могут сохраняться в памяти, а затем воспроизводиться на экране. Такие задачи возникают при построении динамических изображений, в игровых программах, мультипликации и в других случаях. Эти операции возможны и с текстовой информацией. Выполнить их можно процедурами GetImage и PutImage. Обычно, для простоты операций, фрагменты выбирают прямоугольной формы и задают их координатами вершин, расположенных по диагонали.

Сохранение прямоугольного фрагмента в памяти выполняется путем запоминания его размеров и значения яркости каждой точки фрагмента. При этом структура данных имеет следующий вид:

```
type
Image = record
W: integer;          {ширина фрагмента}
H: integer;          {высота фрагмента}
Buf: array[0..H-1,0..W-1] of 0..MaxColor;
end
```

Такой фрагмент можно запоминать, последовательно считывая значения яркости всех точек с помощью процедуры GetImage, в массиве Buf:

```
W := xmax - xmin + 1;
H := ymax - ymin + 1;
for i := 0 to H-1 do
for j := 0 to W-1 do
Buf[i,j] := GetPixel(xmin + j,ymin + i);
```

Воспроизведение записанного фрагмента выполняется аналогично через процедуру PutPixel. При этом следует учитывать размеры фрагмента и область вывода так, чтобы он не выходил за границы экрана. Описанный метод обработки фрагментов

изображения является недостаточно эффективным из-за того, что каждая точка экрана для сохранения требует одного байта памяти, а также из-за выполнения большого числа обращений к процедурам GetPixel и PutPixel. Каждое такое обращение к названным процедурам содержит операции вычисления адресов видеопамяти и является достаточно сложным по числу вычислений.

Более эффективным методом является способ сохранения фрагмента изображения в виде массива строк, каждая из которых имеет структуру строк видеопамяти, т.е. каждой точке соответствует один бит памяти, а строки записываются в последовательность байт. Для сохранения одной строки черно-белого изображения требуется k байт памяти:

k = ((xmax - xmin + 1) + 7) div 8 = (xmax - xmin) div 8 + 1.

В этом случае фрагмент изображения можно описать структурой

```
type
Image = record
W: integer;
H: integer;
Buf: array[0..H-1,0..K-1] of byte
end
```

Рассмотрим операцию копирования упакованных строк фрагмента. Вводятся упакованные строки источника и приемника.

```
LS: array[    ] of byte;   {строка-источник}
LD: array[    ] of byte;   {строка-приемник}
```

В общем случае требуется скопировать биты строки LS, начиная с номера X_{S1} и кончая номером X_{S2}, в биты строки LD, начиная с номера X_{D1} и кончая номером X_{D2} (рис. 1.16).

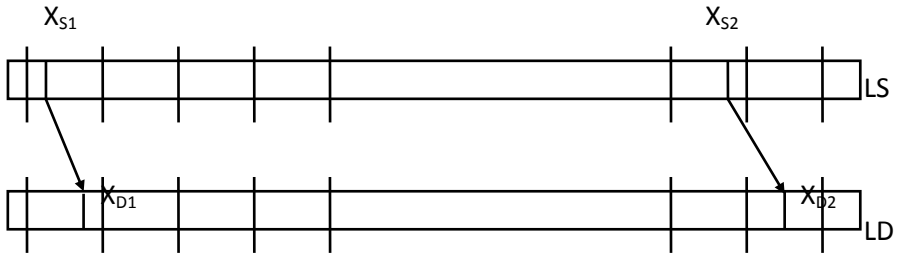


Рис. 1.16

Задача решается эффективно при копировании группами бит, упакованных в байты. Процедура выполняется с использованием операций маскирования и сдвига. При этом требуется две маски для копирования одного байта и две маски – для записи.

Хранение и отображение на экране литер может быть выполнено двумя методами - растровым и векторным. При растровом методе литеры представляются в виде фрагментов образов фиксированного размера. Вывод этих образов на экран возможно процедурой аналогичной процедуре PutImage. Полный набор символов хранится обычно в одном файле, в котором литеры расположены последовательно одна за другой. Для выбора необходимого символа по его коду определяется местоположение образа, который выводится в требуемую точку экрана. Размеры литер могут быть стандартными (стандарт IBM) 8 x 8, 8 x 14, 9 x 16 или произвольного формата. На рис. 1.17 показаны примеры символов, представленных на растре. Каждый символ в данном примере, включая межстрочные и межбуквенные интервалы, изображен на матрице 8 x 8.

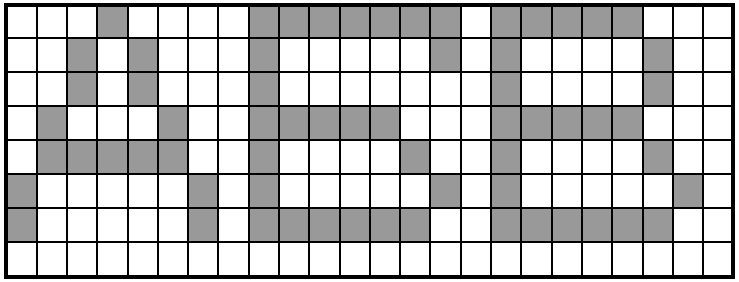


Рис. 1.17

Каждый символ кодируется 8 байтами (восемь строк по восемь бит). Например, для символа “A” двоичные строки выглядят следующим образом:
00010000; 00101000; 00101000; 01000100; 01111100; 10000010; 10000010; 00000000.

Эти биты, упакованные в шестнадцатеричный код, имеют вид: 10; 28; 28; 44; 7C; 82; 82; 00. Таким образом, можно сформировать файл, хранящий все необходимые символы в шестнадцатеричном виде. По номеру символа можно находить требуемый символ в файле.

Векторный метод формирования символов заключается в построении процедуры вычерчивания отрезков (векторов), составляющих изображение символа. При этом используются процедуры MoveTo, LineTo и другие.

Растровые шрифты легче формировать и редактировать. Однако они хуже поддаются масштабированию, так как при этом возникает “лестничный эффект”. Поэтому для использования больших размеров символов, требуется хранение специально изготовленных шрифтов. Растровые шрифты масштабируются без заметных дефектов.

2. ГЕОМЕТРИЧЕСКИЕ ПРЕОБРАЗОВАНИЯ

2.1. Элементарные преобразования точки

Любые преобразования изображений на экране дисплея сводятся к преобразованиям отдельных точек, составляющих растровое изображение. Элементарные преобразования точки базируются на преобразованиях переноса, масштабирования (сжатия, растяжения) и вращения. Из аналитической геометрии известно, что любые сложные преобразования точки можно получить последовательностью (суперпозицией, композицией) элементарных преобразований. В декартовой системе координат точка на плоскости представляется парой чисел (x, y) или в векторном виде **P** = [x y].

Перенос точки (x, y) относительно начала координат в точку (x*, y*) математически может быть записан следующим образом:

$$x^* = x + Dx;$$

$$y^* = y + Dy.$$

В векторной форме это преобразование имеет вид **P*** = **P** + **T**, где **P** = [x y], **P*** = [x* y*], **T =** [Dx Dy]. Результаты переноса точки проиллюстрированы на рис. 2.1.

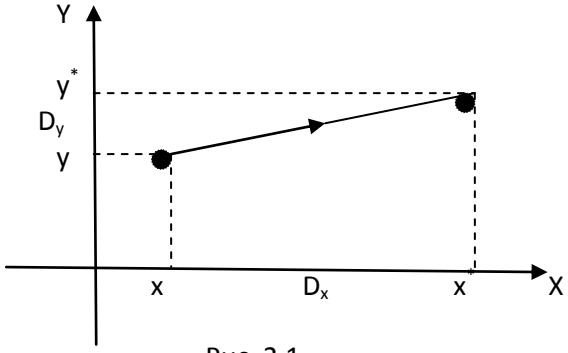


Рис. 2.1

Масштабирование точки может выполняться относительно начала координат по обеим осям - X и Y через коэффициенты масштабирования S_x и S_y. Координаты точки после масштабирования определяются в соответствии с выражениями

$$x^* = x \cdot Sx,$$

$$y^* = y \cdot Sy$$

или в векторной форме

$$P^* = P \cdot S,$$

где **S** = $\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$.

Поворот точки на угол φ относительно начала координат (рис. 2.2) выполняется следующими соотношениями:

$$x^* = x \cdot \cos\varphi - y \cdot \sin\varphi,$$
$$y^* = x \cdot \sin\varphi + y \cdot \cos\varphi.$$

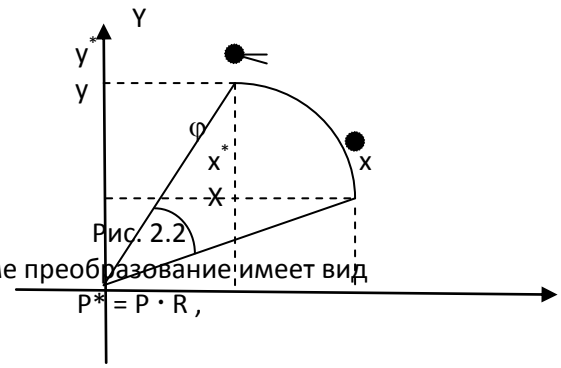


Рис. 2.2

В векторной форме преобразование имеет вид

$$P^* = P \cdot R,$$

где **R** = $\begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix}$.

Рассмотренные элементарные преобразования и их композиции в компьютерной графике обычно выполняются в **однородных координатах**, обладающих тем свойством, что определяемый ими объект не меняется при умножении всех координат на одно и то же число. Введение однородных координат позволяет унифицировать все элементарные преобразования, которые получаются с использованием только операции умножения.

В однородных координатах точка P(x, y) записывается как трехкомпонентный вектор с некоторым произвольным масштабным множителем $W \neq 0$: P(X, Y, W). Связь декартовых и однородных координат устанавливается следующим образом:

$$x = X/W,\ y = Y/W.$$

Обычно удобно использовать множитель W = 1. Тогда соотношение декартовых и однородных координат выглядит совсем просто: x = X, y = Y, а запись точки в однородных координатах имеет вид P(X, Y, 1). Элементарные преобразования в однородных координатах математически будут выглядеть следующим образом.

Обозначим исходную точку P вектором **P =** [x y 1], тогда преобразованную точку запишем как **P* =** [x* y* 1]. Преобразование **переноса точки** в однородных координатах можно выразить как

$$P^* = P \cdot T,$$

где

$$\mathbf{T}(D_x, D_y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix}.$$

Преобразование **масштабирования точки** в однородных координатах имеет вид

$$P^* = P \cdot S,$$

где

$$\mathbf{S}(S_x, S_y) = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Аналогично можно записать преобразование **поворота точки** в однородных координатах.

$$P^* = P \cdot R,$$

где

$$\mathbf{R}(\varphi) = \begin{bmatrix} Cos\varphi & Sin\varphi & 0 \\ -Sin\varphi & Cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

2.2. Композиции элементарных преобразований

В предыдущем подразделе рассматривались преобразования точки относительно начала координат. В качестве примера несложных композиций преобразований можно рассмотреть преобразования масштабирования и поворота относительно произвольной точки плоскости.

Пусть требуется выполнить масштабирование изображения относительно точки P(x,y). Это преобразование можно выполнить последовательностью следующих элементарных преобразований:

Перенос точки P(x,y) и изображения таким образом, чтобы точка P(x,y) совпала с началом системы координат;

Масштабирование изображения относительно начала координат;

Перенос точки P(x,y) с изображением в исходное положение.

Каждое из указанных элементарных преобразований изображения можно получить путем умножения на соответствующую матрицу преобразования. Результирующая матрица преобразования может быть получена перемножением всех матриц элементарных преобразований:

$$MS = T(-x, -y) \cdot S(Sx, Sy) \cdot T(x, y) =$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x & -y & 1 \end{bmatrix} * \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ x(1-S_x) & y(1-S_y) & 1 \end{bmatrix}.$$

Для преобразования всего изображения необходимо преобразовать каждую точку с помощью результирующей матрицы $\mathbf{M_S}$. Возможно, что преобразования можно упростить за счет вычисления только некоторых точек изображения, а остальные точки получить построением отрезков прямых или иным способом. Это возможно, так как здесь рассматриваются только аффинные преобразования, при которых прямые преобразуются в прямые, параллельные – в параллельные, пересекающиеся – в пересекающиеся.

Аналогично можно выполнить преобразование поворота относительно произвольной точки P(x,y). Результирующая матрица получается следующим образом:

$$MR= T(-x, -y) \cdot R(\varphi) \cdot T(x, y) =$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x & -y & 1 \end{bmatrix} * \begin{bmatrix} Cos\varphi & Sin\varphi & 0 \\ -Sin\varphi & Cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} Cos\varphi & Sin\varphi & 0 \\ -Sin\varphi & Cos\varphi & 0 \\ x(1-Cos\varphi)+ySin\varphi & y(1-Cos\varphi)-xSin\varphi & 1 \end{bmatrix}.$$

В более сложных случаях композиция преобразований может состоять из любого числа последовательных преобразований переноса, поворота и масштабирования. В этом случае результирующая матрица преобразования имеет характерные вид

$$M = \begin{bmatrix} r_{11} & r_{12} & 0 \\ r_{21} & r_{22} & 0 \\ t_x & t_y & 1 \end{bmatrix}.$$

Подматрица размером 2x2 из элементов r_{ij} является результирующей матрицей поворотов и масштабирования, а элементы t_x и t_y представляют суммарный перенос по осям x и y. Характерность матрицы заключается и в том, что последний столбец содержит нули и одну единицу. Последнее обстоятельство позволяет упростить вычисления. Прямое умножение на матрицу третьего порядка требует выполнения 9 операций умножения и 6 операций сложения. Однако, тот же результат можно получить, выполнив следующие вычисления:

$$x^* = x \cdot r_{11} + y \cdot r_{21} + t_x;$$
$$y^* = x \cdot r_{12} + y \cdot r_{22} + t_y.$$

Здесь требуется четыре операции умножения и четыре - сложения.

2.3. Операции с точками и прямыми

При выполнении синтеза и преобразования изображений применяются операции, связанные с определением взаимного расположения точки и отрезка прямой. Эти преобразования требуют матричных операций и, как правило, проводятся в однородных координатах. Рассмотрим некоторые из них.

Уравнение прямой, проходящей через две точки. Если заданы две точки в однородных координатах $\mathbf{P_1}=(x_1\ y_1\ w_1)$ и $\mathbf{P_2} = (x_2\ y_2\ w_2)$, то

для любой текущей точки **P**=(x y w), принадлежащей прямой, проходящей через эти точки, справедливо выражение

$$\begin{vmatrix} x & x_1 & x_2 \\ y & y_1 & y_2 \\ w & w_1 & w_2 \end{vmatrix} = 0.$$

Раскрывая определитель, получаем уранение прямой

$x*(y_1 w_2 - y_2 w_1) - y*(x_1 w_2 - x_2 w_1) + w*(x_1 y_2 - x_2 y_1) = 0.$

Иначе это можно записать следующим образом

$$a*x + b*y + c*w = 0,$$

где

$$a = y_1 w_2 - y_2 w_1,\ b = x_2 w_1 - x_1 w_2,\ c = x_1 y_2 - x_2 y_1.$$

Координаты пересечения двух прямых. Запишем уравнение прямой в следущем виде:

$$a*x + b*y + c = 0.$$

Тогда для двух прямых можно записать

$$a_1*x + b_1*y + c_1 = 0,$$
$$a_2*x + b_2*y + c_2 = 0.$$

Для третьей прямой $a*x + b*y + c = 0$, проходящей через точку пересечения двух прямых, выполняется равенство

$$\begin{vmatrix} a & a_1 & a_2 \\ b & b_1 & b_2 \\ c & c_1 & c_2 \end{vmatrix} = 0.$$

После раскрытия определителя имеем

$a*(b_1 c_2 - b_2 c_1) - b*(a_1 c_2 - a_2 c_1) + c*(a_1 b_2 - a_2 b_1) = 0.$

Выражения в скобках дают значения однородных координат точки пересечения трех прямых:

$$\mathbf{P} = (b_1c_2 - b_2c_1\ a_2c_1 - a_1c_2\ a_1b_2 - a_2b_1).$$

Если прямые не пересекаются, то $a_1b_2 - a_2b_1 = 0$.

Положение точки относительно прямой. Пусть дана точка **P**(x,y,w) и прямая, проходящая через точки $\mathbf{P_1}(x_1,y_1,w_1)$ и $\mathbf{P_2}(x_2,y_2,w_2)$. Точка **P**(x,y,w) расположена справа от прямой (рис. 2.3), если при движении наблюдателя от точки $\mathbf{P_1}$ к точке $\mathbf{P_2}$ она находится справа.

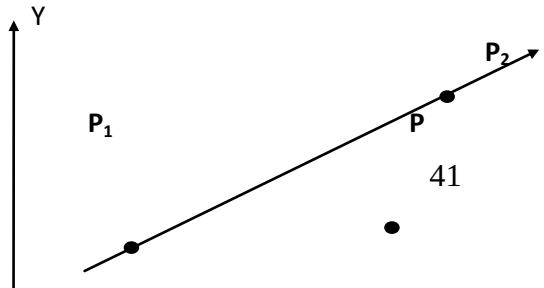


Рис. 2.3

В этом случае определитель отвечает условию

$$\begin{bmatrix} x & x_1 & x_2 \\ y & y_1 & y_2 \\ w & w_1 & w_2 \end{bmatrix} < 0$$

.

Это условие в векторной форме записывается как

$$\left|P \quad P_1 \quad P_2\right| < 0.$$

При изменении направления движения наблюдателя меняется и знак неравенства. Расположение точки слева от прямой определяется изменением знака неравенства на противоположный.

Пересечение двух отрезков прямых. Метод определения взаимного положения прямой и точки позволяет выявить наличие пересечения и точку пересечения двух отрезков прямых. Для того чтобы отрезки прямых P_1P_2 и P_3P_4 пересекались, необходимо выполнение одновременно двух условий:

1) точки P_3 и P_4 должны лежать по разные стороны от прямой P_1P_2;

2) точки P_1 и P_2 должны лежать по разные стороны от прямой P_3P_4.

Если обозначить определители третьего порядка для точек и прямых следующим образом:

S1 = $\left|P_1 \quad P_3 \quad P_4\right|$, **S2 =** $\left|P_2 \quad P_3 \quad P_4\right|$,

S3 = $\left|P_3 \quad P_1 \quad P_2\right|$, **S4 =** $\left|P_4 \quad P_1 \quad P_2\right|$,

то условие пересечения двух отрезков прямых записывается как

(S1 * S2 < 0) & (S3 * S3 < 0).

Координаты точки пересечения **P**(x,y,w) можно определить, решив систему уравнений

$$\begin{cases} \left|P \quad P_1 \quad P_2\right| = 0 \\ \left|P \quad P_3 \quad P_4\right| = 0 \end{cases},$$

которую следует дополнить уравнением w = c, где **c** является любой константой больше нуля. В простейшем случае можно положить w =

1. Решение системы дает координаты точки пересечения двух отрезков прямых в однородных координатах. Возможны граничные случаи, когда одна из концевых точек отрезка P_i (i = 1, 2, 3, 4) попадает на другой отрезок. Тогда соотвествующий определитель S_i будет равен нулю. Возможен и случай равенства нулю двух определителей, если они относятся к точкам разных отрезков. Это возможно, когда точка пересечения совпадает с концевыми точками разных отрезков. Эти случаи следует учитывать, тем более, что они не выявляются с помощью предложенного условия пересечения отрезков прямых.

Положение точки относительно многоугольника. Задачу определения взаимного положения точки и выпуклого многоугольника можно решить последовательным обходом сторон многоугольника по или против часовой стрелки. При обходе по часовой стрелке необходимо определять взаимное положение точки и сторон многоугольника. Если точка находится справа относительно всех сторон многоугольника, то она расположена во внутренней области. В противном случае она находится снаружи по отношению к многоугольнику.

Для невыпуклого многоугольника задача решается проведением произвольной прямой (например, горизонтальной) через интересующую нас точку. Определяются точки пересечения прямой со сторонами многоугольника и упорядочиваются по возрастанию x-координат. Если точка расположена между нечетным и четным пересечением, то она находится во внутренней области.

При решении задачи взаимного положения точки и многоугольника возможны частные случаи попадания точки на одну из сторон многоугольника. Обычно считается, что стороны многоугольника принадлежат внутренней области.

3. ДВУМЕРНЫЕ ОТСЕЧЕНИЯ

3.1. Отсечение отрезков регулярным окном

Задача отсечения решает вопрос выделения части некоторого изображения. При построении сложных изображений эта задача выступает инструментом при удалении невидимых линий и поверхностей. Кроме этого, задача отсечения решает вопросы определения пересечения фигур, объемных тел, многогранников. В

общем случае вопросы отсечения решают задачи как плоских, так и объемных изображений. Основное требование к алгоритмам - обеспечение приемлемой скорости обработки изображений, желательно в реальном масштабе времени. С этой целью возможны аппаратная и микропрограммная реализация алгоритмов. Решение задачи отсечения предполагает использование регулярных отсекающих окон в виде прямоугольника, квадрата, куба, параллелепипеда и нерегулярных произвольной формы.

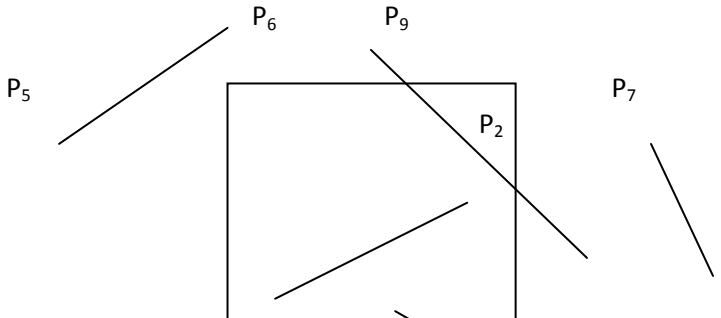
Рассмотрим задачу отсечения для **регулярного окна** в виде прямоугольника со сторонами (ребрами), обозначаемыми символами Л, П, В, Н для левой, правой, верхней и нижней сторон соответственно. Целью отсечения является определение тех точек, отрезков или их частей, которые лежат внутри отсекающего окна. Эти элементы изображения являются видимыми в окне, и они остаются для визуализации, а остальные элементы отсекаются.

Обычно при решении задачи отсечения приходится иметь дело с большим числом элементов изображения. В связи с этим важное значение приобретает возможность отбрасывания большого числа элементов изображения с минимальными затратами машинного времени за счет выявления у них некоторых характерных особенностей.

Рассмотрим возможные варианты расположения элементов изображения по отношению к отсекающему окну (рис. 3.1). Точки, лежащие внутри окна, являются видимыми и удовлетворяют условию

$$x_л \leq x \leq x_п,\ y_н \leq y \leq y_в.$$

Знак равенства указывает на то, что граничные точки также входят в видимую область. Если оба конца отрезка лежат внутри отсекающего окна, то отрезок целиком находится внутри его (например, отрезок P_1P_2). Обратное утверждение в общем случае неверно (например, отрезок P_9P_{10}). Если оба конца отрезка лежат снаружи, то отрезок может быть полностью невидимым (отрезки P_5P_6, P_7P_8) или видимым частично (P_9P_{10}).



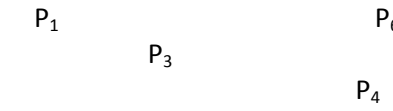


Рис. 3.1

Если один конец отрезка находится внутри, а второй снаружи, то отрезок будет частично видимым (отрезок P_3P_4).

Прямые, проходящие через ребра отсекающего окна, делят плоскость на две полуплоскости. Такие прямые называются гранями отсекающего окна. Полуплоскость, содержащая отсекающее окно, называется внутренней, а другая соответственно – внешней. Если отрезок полностью лежит во внешней полуплоскости для какого-то ребра, то отрезок будет полностью невидимым.

В соответствие с изложенным можно сформулировать простые условия **полной видимости-невидимости отрезка** P_1P_2, заданного координатами (x_1, y_1) и (x_2, y_2).

Условие **полной видимости**:

$$(x_л \le x_1 \le x_п) \,\&\, (x_л \le x_2 \le x_п) \,\&\, (y_н \le y_1 \le y_в) \,\&\, (y_н \le y_2 \le y_в).$$

Аналогичное условие для **полной невидимости** (например, отрезка – P_7P_8):

$$((x_1 < x_л) \,\&\, (x_2 < x_л)) \vee ((x_1 > x_п) \,\&\, (x_2 > x_п)) \vee$$
$$\vee ((y_1 < y_н) \,\&\, (y_2 < y_н)) \vee ((y_1 > y_в) \,\&\, (y_2 > y_в)).$$

Если условие полной видимости не выполняется, то отрезок является, либо полностью невидимым, либо невидимым частично. Если при этом не выполняется еще и условие полной невидимости, то отрезок, либо полностью невидим, либо частично. Это связано с тем, что условие полной невидимости выделяет только часть невидимых отрезков тех, что располагаются целиком по одну сторону грани, являющейся продолжением некоторого ребра отсекающего окна. Таким образом, два указанных условия позволяют относительно простой процедурой выявить все видимые отрезки и часть невидимых. Другая часть полностью невидимых отрезков, не выявленных применением второго условия, определяется **общим алгоритмом задачи отсечения**.

Данный алгоритм применяется к отрезкам, для которых не выполняются условия полной видимости-невидимости. На рис. 3.2 показан пример расположения такого отрезка. Для выделения видимой части отрезка необходимо найти точки пересечения его со

сторонами отсекающего окна. В общем случае неизвестно, с какими сторонами окна пересекается заданный отрезок, поэтому следует определить его пересечения со всеми сторонами окна, а затем выделить интересующие нас точки. Для решения этой задачи следует определить пересечения прямой, продолжающей отрезок P_1P_2, с прямыми, продолжающими ребра отсекающего окна (гранями отсекающего окна).

Уравнение прямой для отрезка P_1P_2 имеет вид

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1},$$

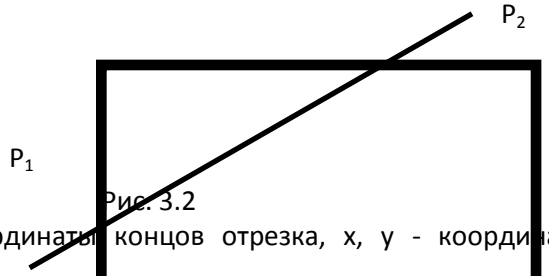


Рис. 3.2

где x_i, y_i - координаты концов отрезка, x, y - координаты текущей точки отрезка.

Пересечение этой прямой с гранями отсекающего окна имеет следующие координаты (рис. 3.3):

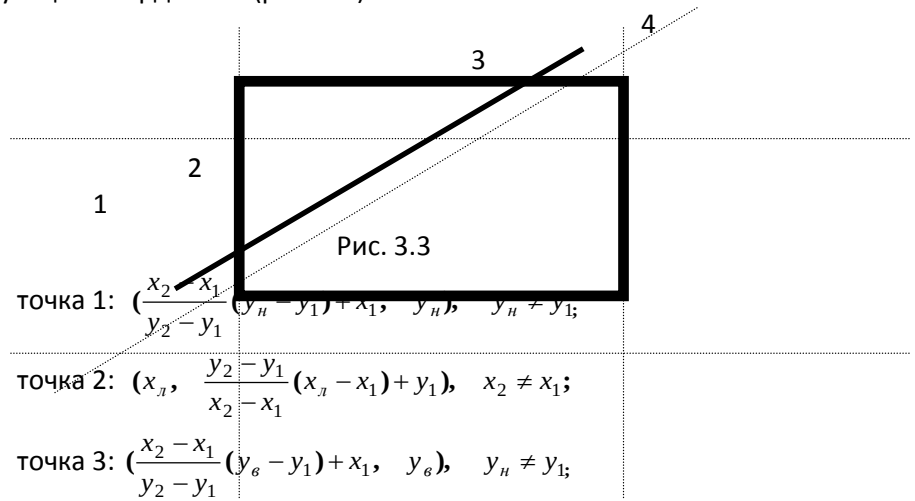


Рис. 3.3

точка 1: $\left(\frac{x_2 - x_1}{y_2 - y_1}(y_н - y_1) + x_1,\ y_н\right),\quad y_н \neq y_1;$

точка 2: $\left(x_л,\ \frac{y_2 - y_1}{x_2 - x_1}(x_л - x_1) + y_1\right),\quad x_2 \neq x_1;$

точка 3: $\left(\frac{x_2 - x_1}{y_2 - y_1}(y_в - y_1) + x_1,\ y_в\right),\quad y_н \neq y_1;$

точка 4: $(x_н,\ \frac{y_2 - y_1}{x_2 - x_1}(x_н - x_1) + y_1),\quad x_2 \neq x_1;$

После нахождения четырех точек пересечения необходимо выделить среди них те, которые принадлежат отсекающему окну и отрезку. Проверка принадлежности окну точек пересечения (x_i, y_i) выполняется с помощью условий $x_л \le x_i \le x_п$, $y_н \le y_i \le y_в$, а принадлежность отрезку аналогичными условиями: $x_{P_1} \le x_i \le x_{P_2}$, $y_{P_1} \le y_i \le y_{P_2}$.

В случае расположения отрезка таким образом, что одна его концевая точка располагается внутри окна, а вторая - снаружи, необходимо проверить координаты точек пересечения 1, 2, 3, 4 на принадлежность их отсекающему окну и прямоугольнику, построенному на отрезке P_1P_2, как на его диагонали (рис. 3.4), т.е. выполнить проверку условия для точек, попадающих в отсекающее окно:

$$(x_{P_1} \le x \le x_{P_2}) \,\&\, (y_{P_1} \le y \le y_{P_2}).$$

Рис. 3.4

Кроме случаев, рассмотренных выше, следует учитывать возможность особого (граничного) варианта расположения отрезка, при котором он располагается по горизонтали, вертикали или проходит через вершину отсекающего окна. Для анализа этих случаев требуются несложные дополнительные проверки.

Приведенный выше подход может быть положен в основу разработки алгоритмов, детализирующих его реализацию в

различных вариантах. В качестве примера рассмотрим одну из таких модификаций, носящею название **алгоритма Сазерленда-Коэна**.

Идея алгоритма заключается в том, что каждая грань отсекающего окна делит плоскость на две полуплоскости - внутреннюю, содержащую отсекающее окно, и внешнюю. В свою очередь, отрезок в точке пересечения с гранью окна также делится на две части, которые располагаются во внешней и внутренней полуплоскостях. Часть отрезка, расположенная во внешней полуплоскости, в соответствии с алгоритмом отбрасывается, как заведомо невидимая. Алгоритм можно сформулировать в следующем виде.

1. Для отрезка P_1P_2 проверяется выполнение условий полной видимости и полной невидимости. Если выполняется одно из условий, то отрезок является либо полностью видимым, либо невидимым и работа алгоритма заканчивается.
2. Если точка P_1 находится во внутренней полуплоскости, то обозначения концевых точек P_1P_2 меняются местами.
3. Точка P_1 заменяется на пересечение отрезка P_1P_2 с очередной границей окна. Переход к п. 1.

Реализация алгоритма требует детализации двух операций - определение внутренней и внешней полуплоскостей и нахождение точек пересечения с границами отсекающего окна. Задачу с полуплоскостями можно решить на основе учета наименования ребра, для которого решается эта задача. Так, для левого ребра внутренняя полуплоскость располагается правее и отбрасывать следует ту часть отрезка, для концевой точки Р которой выполняется условие $X_P < X_Л$**.** Аналогично решается задача и для других ребер окна. Точки пересечения определяются так, как это было показано в подразделе 1.2.

Для повышения эффективности алгоритма можно использовать специальный четырехбитовый код признака М(Р) точки Р, который имеет следующий структуру:

Л	П	В	Н

Соответствующий бит равен единице, если по отношению к этой грани точка находится во внешней полуплоскости. При таком подходе плоскость разбивается на 9 областей. На рис. 3.5 показаны эти области с соответствующими кодами признаков.

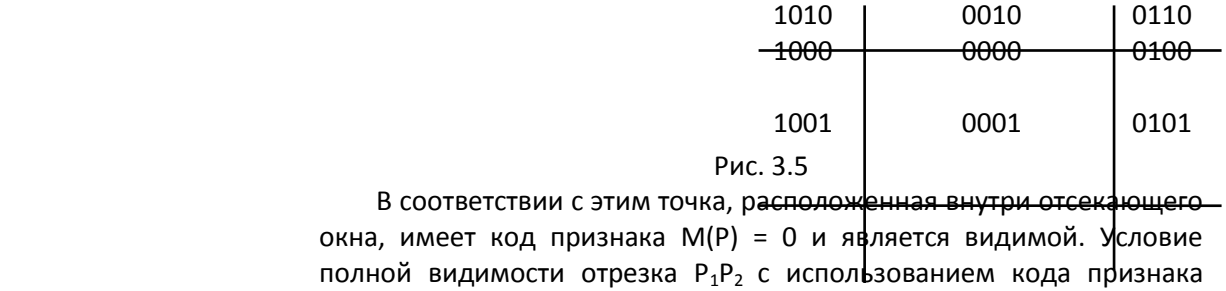


Рис. 3.5

В соответствии с этим точка, расположенная внутри отсекающего окна, имеет код признака M(P) = 0 и является видимой. Условие полной видимости отрезка P_1P_2 с использованием кода признака примет вид

$$(M(P1) = 0) \ \& \ (M(P2) = 0).$$

Аналогично строится условие полной невидимости этого отрезка.

$$(M(P1) \ \& \ M(P2)) \neq 0.$$

Первое условие представляет собой логическое выражение , а второе - логическую операцию над кодами.

С целью повышения эффективности алгоритма можно также использовать параметрическую форму представления отрезков, которая облегчает проверку их расположения. Координаты текущей точки P(x,y) отрезка P1P2 в параметрической форме записываются следующим образом:

$$x = x1 + (x2 - x1) * t,$$

$$y = y1 + (y2 - y1) * t,$$

где $t \in [0,1]$, т.е. параметр t принимает значения в интервале от 0 до 1. Параметрическая запись в векторной форме имеет вид:

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

Тогда пересечение прямой, продолжающей отрезок P_1P_2, с левой гранью отсекающего окна дает значение

$$t = \frac{x - x_1}{x_2 - x_1},$$

где $t = t_л$ при $x = x_л$ для левой грани,

$t = tп$ при $x = xп$ для правой грани.

Аналогично можно получить значения параметра t для верхней и нижней граней

$$t = \frac{y - y_1}{y_2 - y_1},$$

где $t = t_в$ при $y = y_в$ для верхней грани,

$t = tн$ при $y = yн$ для нижней грани.

Если выполняется условие $0 \leq t \leq 1$, то точка пересечения принадлежит отрезку P_1P_2, в противном случае - нет.

3.2. Задача отсечения отрезков многоугольником

В качестве отсекающего окна при решении задачи отсечения может использоваться произвольный многоугольник. Простейшим вариантом такого многоугольника может служить прямоугольное окно с неортогональными сторонами по отношению к осям координат.

Алгоритмы отсечения для выпуклых и невыпуклых многоугольников в общем случае различаются. Рассмотрим общие подходы к решению задачи отсечения для выпуклых многоугольников.

Пусть выпуклый многоугольник задан последовательностью вершин, указанных в порядке обхода его по часовой стрелке - A_1, A_2, ..., A_n. Для выпуклого многоугольника также можно сформулировать условия полной видимости и полной невидимости отрезков. Как и для регулярного окна, условием полной видимости отрезка является расположение обеих его концевых точек внутри многоугольника. Принадлежность точки внутренней области многоугольника определяется обходом его сторон по часовой стрелке. Если при этом обходе точка расположена справа по отношению всех ребер многоугольника, то точка принадлежит внутренней его области. Формально процедура обхода и выяснения расположения точки относительно ребер многоугольника выполняется вычислением определителей, составленных для всех ребер многоугольника, для которых должно выполняться следующее условие

$$\begin{vmatrix} x & x_{Ai} & x_{Ai+1} \\ y & y_{Ai} & y_{Ai+1} \\ W & W_{Ai} & W_{Ai+1} \end{vmatrix} \leq 0,$$

где x,y - координаты анализируемой точки, x_{Ai}, y_{Ai}, x_{Ai+1}, y_{Ai+1} - координаты концов ребра A_i A_{i+1} многоугольника, W, W_{Ai}, W_{Ai+1} - множители однородных координат.

Более компактно это условие записывается в векторной форме

$$\left|PA_iA_{i+1}\right| \le 0 \text{ , } \quad i = 1,2, \dots , n\text{-}1; \qquad \left|PA_nA_1\right| \le 0,$$

где **P, A -** вектор-столбцы однородных координат соответствующих точек.

Таким образом, для определения принадлежности точки области многоугольника необходимо вычислить n определителей третьего порядка, а для отрезка - 2n определителей. В связи с тем, что даже для небольших значений n это может представлять достаточно трудоемкую операцию, то целесообразно отказаться от проверки условия полной видимости отрезка, перенеся решение задачи видимости отрезка на процедуру анализа расположения отрезка в общем случае.

Случай полной невидимости отрезка проверяется также как и для ортогонального окна. В соответствии с этим отрезок должен располагаться полностью по одну сторону (внешнюю) от прямой, продолжающей сторону многоугольника (рис. 3.6).

Здесь показаны случаи расположения двух отрезков P_1P_2 и P_3P_4. Они располагаются за пределами окна отсечения и поэтому являются полностью невидимыми.

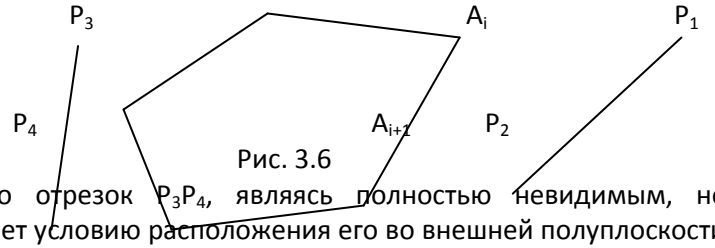


Рис. 3.6

Однако отрезок P_3P_4, являясь полностью невидимым, не удовлетворяет условию расположения его во внешней полуплоскости ни для одной грани многоугольника, и, следовательно, не может быть выявлен как невидимый с помощью проверки условий, аналогичных тем, что были рассмотрены в подразделе 1.1. В отличие от этого отрезок P_1P_2 находится во внешней полуплоскости по отношению к грани ребра A_iA_{i+1}, и определители для концевых точек отрезка имеют значения

$$|P_1A_iA_{i+1}| > 0 \text{ и } |P_2A_iA_{i+1}| > 0.$$

В общем случае выявление невидимости отрезка приводит к вычислению не более 2n определителей третьего порядка. При удачном выборе обхода ребер многоугольника может оказаться достаточным вычисление только двух определителей. Во многих случаях можно уменьшить объем вычисления использованием минимаксной оболочки в виде ортогонального окна, имеющего координаты

$$x_л = x_{min};\ x_п = x_{max};\ y_в = y_{max};\ y_н = y_{min}.$$

Здесь min и max относятся к минимальным и максимальным координатам вершин многоугольника (рис. 3.7).

Естественно, при этом упрощении часть полностью невидимых отрезков, которые могли бы быть выявленными проверкой условия невидимости для многоугольника, будет отброшено. В общем случае эта часть отрезков может быть невелика и выигрыш в объеме вычислений компенсирует эту потерю.

Возможна и ситуация (по объему вычислений), при которой имеет смысл отказаться от выявления невидимых отрезков с помощью определителей, переложив эту задачу на общий алгоритм задачи отсечения.

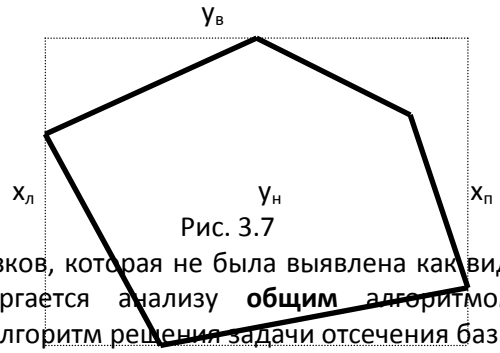


Рис. 3.7

Та часть отрезков, которая не была выявлена как видимая или невидимая, подвергается анализу **общим** алгоритмом задачи отсечения. Общий алгоритм решения задачи отсечения базируется на определении точки пересечения отрезка P_1P_2, для которого решается эта задача, с ребром многоугольника. С этой целью определим положение всех вершин многоугольника относительно прямой, продолжающей отрезок P_1P_2, путем вычисления следующих определителей третьего порядка.

$$S_i = | \mathbf{A_i\ P_1\ P_2} |, \quad I = 1, 2, \ldots, n.$$

Значения n определителей S_i могут быть больше, меньше нуля или равны ему. Все возможные сочетания этих значений разобьем на два случая.

Первый случай: все значения определителей S_i имеют один знак или равны нулю, т.е. вершины расположены по одну сторону отрезка P_1P_2 или лежат на нем. При этом возможны следующие варианты.

Вариант 1. Все значения S_i не равны нулю ($S_i \neq 0$) - отрезок полностью невидим (все вершины расположены по одну сторону отрезка и не соприкасаются с ним).

Вариант 2. Одно значение определителя S_i равно нулю ($S_i = 0$) - прямая, продолжающая отрезок P_1P_2, проходит через вершину A_i. В этом случае надо решить задачу принадлежности этой вершины отрезку P_1P_2 путем определения принадлежности вершины A_i внутренней области регулярного окна в виде прямоугольника, построенного на диагонали P_1P_2. Если она принадлежит отрезку, то у этого отрезка видимой является только точка A_i.

Вариант 3. Два значения определителя равны нулю ($S_i = 0$ и $S_j = 0$) - прямая, продолжающая отрезок P_1P_2, совпадает с гранью ребра A_iA_j. Здесь необходимо исследовать взаимное расположение отрезка P_1P_2 и ребра A_iA_j. При этом возможны варианты полной видимости отрезка, когда

$$x_{min}(P_1P_2) > x_{min}(A_iA_j) \quad \text{и} \quad x_{max}(P_1P_2) < x_{max}(A_iA_j),$$

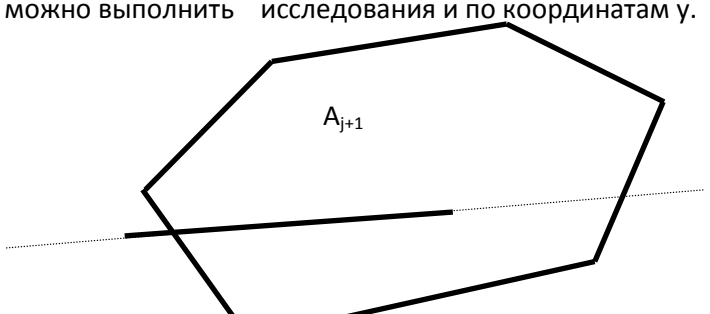
а также полной невидимости, когда либо

$$x_{max}(P_1P_2) < x_{min}(A_iA_j),$$

либо

$$x_{min}(P_1P_2) > x_{max}(A_iA_j).$$

В случае выполнения равенства, в приведенных выше условиях, будет видна только одна точка отрезка P_1P_2 - точка $P_2 = A_i$ или точка $P_1 = A_j$ соответственно. Кроме этого, возможны и частичные перекрытия отрезков, что можно выяснить аналогичными сравнениями координат. Причем здесь приведен анализ точек по координатам x. Но можно выполнить исследования и по координатам y.



P_1 P_2

A_{k+1}

A_j

Рис. 3.8

Второй случай: определители S_i имеют разные знаки, т.е. прямая, продолжающая отрезок P_1P_2, пересекает многоугольник. При этом пересечение возможно только в двух точках. Так как многоугольник выпуклый, то при последовательном обходе его вершин знаки определителей S_i могут менять свое значение на противоположное только два раза. Пусть эта смена знака происходит для смежных пар вершин A_j и A_{j+1}, A_k и A_{k+1} (рис. 3.8).

Для определения видимой части отрезка P_1P_2 необходимо определить положение точек P_1 и P_2 относительно отрезков A_jA_{j+1}, A_kA_{k+1}. Для этого требуется выполнить анализ четырех определителей при обходе ребер многоугольника по часовой стрелке:

$$S_{1j} = |\mathbf{P_1 A_j A_{j+1}}|, \quad S_{2j} = |\mathbf{P_2 A_j A_{j+1}}|,$$
$$S_{1k} = |\mathbf{P_1 A_k A_{k+1}}|, \quad S_{2k} = |\mathbf{P_2 A_k A_{k+1}}|.$$

Если определитель меньше нуля, то соответствующая точка отрезка лежит справа от ребра, т.е. внутри многоугольника. При значении определителя больше нуля точка находится слева во внешней по отношению к многоугольнику области. Равенство определителя нулю говорит о нахождении точки на соответствующем ребре многоугольника.

Рассмотрение всех возможных сочетаний значений этих определителей дает 64 варианта (4^3). Исключение случая нахождения концевой точки отрезка на одном из ребер многоугольника существенно уменьшает число вариантов до 16. Однако практически достаточно рассмотреть только четыре из них. Рассмотрим эти варианты.

Вариант 1. *Отрезок лежит вне многоугольника.* В этом случае он полностью невидим. Данное расположение отрезка соответствует следующим сочетаниям знаков определителей:

	S_{1j}	S_{1k}	S_{2j}	S_{2k}
Вариа	+	-	+	-

нт 1.1.				
Вариант 1.2.	-	+	-	+

Эти варианты показаны на рис. 3.9. Вариант 1.1 соответствует положению отрезка слева от многоугольника, а 1.2 – справа. Следует отметить, что ориентация отрезка в обратном направлении не влияет на результат его невидимости.

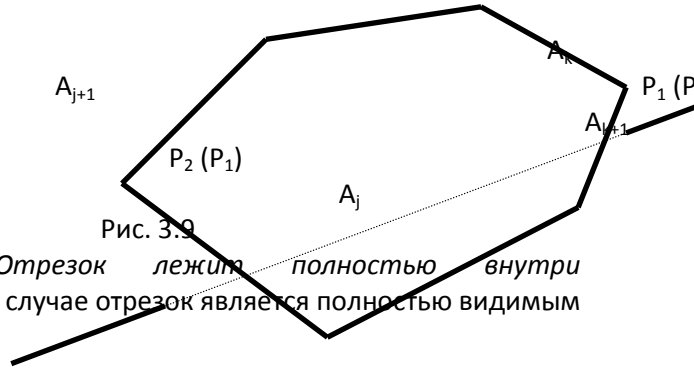


Рис. 3.9

Вариант 2. *Отрезок лежит полностью внутри многоугольника.* В этом случае отрезок является полностью видимым (рис. 3.10.)

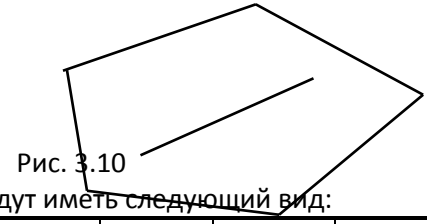
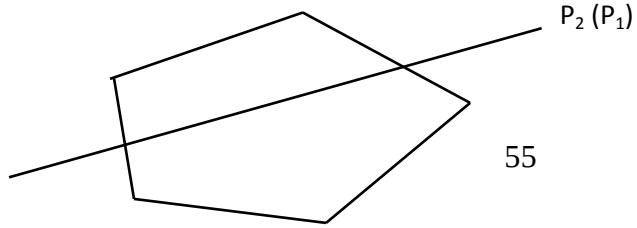


Рис. 3.10

Знаки определителей будут иметь следующий вид:

	S_{1j}	S_{1k}	S_{2j}	S_{2k}
Вариант 2.1.	-	-	-	-

Вариант 3. *Точки P_1, P_2 лежат по разные стороны многоугольника*. В этом случае отрезок является частично видимым. Поэтому следует определить две точки пересечения отрезка с ребрами многоугольника. Видимая часть отрезка будет находиться между ними (рис. 3.11).



P_1 (P_2)

Рис. 3.11

Знаки определителей для этого случая приводятся ниже. При этом изменение ориентации отрезка на противоположную дает такое же изменение знаков.

	S_{1j}	S_{1k}	S_{2j}	S_{2k}
Вариант 3.1.	+	-	-	+
Вариант 3.2.	-	+	+	-

Вариант 4. *Один конец отрезка находится внутри многоугольника, другой - снаружи.* При этом отрезок также является частично видимым - между точкой пересечения отрезка с ребром многоугольника и концевой точкой отрезка, расположенной внутри отсекающего многоугольника (рис. 3.12).

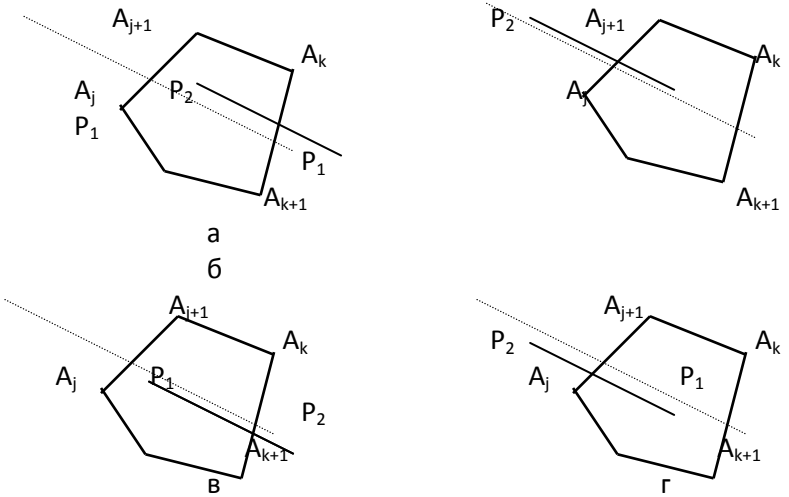


Рис. 3.12

Знаки определителей для этих случаев имеют следующий вид:

	S_{1j}	S_{1k}	S_{2j}	S_{2k}
Вариант 4.1.	-	+	-	-

Вариант 4.2.	+	-	-	-
Вариант 4.3.	-	-	-	+
Вариант 4.4.	-	-	+	-

В соответствие с рассмотренными вариантами можно построить общий алгоритм определения видимой части отрезка.

Повышение быстродействия алгоритмов отсечения возможно за счет использования более эффективного метода определения местоположения точки отрезка относительно отсекающего окна. Такой метод заложен в **алгоритме Кируса-Бека**. В нем используется вектор внутренней нормали к ребрам многоугольника. При этом выпуклый многоугольник представляется как область пересечений полуплоскостей, ограниченных прямыми, проходящими через ребра многоугольника (рис. 3.13).

Внутренней нормалью к ребру многоугольника называется единичный вектор, перпендикулярный к этому ребру и направленный во внутреннюю полуплоскость, определяемой гранью данного ребра многоугольника. Обозначим внутреннюю нормаль ребра A_iA_{i+1} через $\mathbf{n}_i$ (I = 1,2,...,n). Если f - некоторая точка, лежащая на ребре многоугольника, **n** - внутренняя нормаль к этому ребру, а P - некоторая точка, полуплоскости, то знак скалярного произведения нормали и вектора из точки f в точку P определяет положение точки относительно грани, содержащей данное ребро (рис. 3.14).

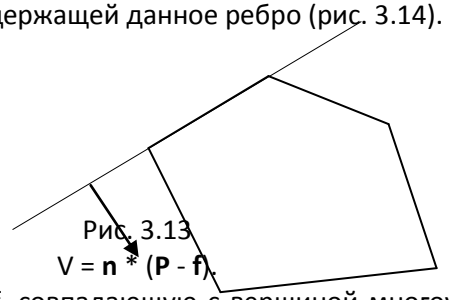


Рис. 3.13

V = **n** * (**P** - **f**).

Если взять точку f, совпадающую с вершиной многоугольника, то можно рассмотреть следующие три характерных случая расположения точки P, принадлежащей отрезку P_1P_2 (рис. 3.15).

Первый случай - угол между векторами **n** и **(P-f)** больше 90^0:

V = **n** * (**P** - **f**) < 0, $\alpha > \pi/2$.

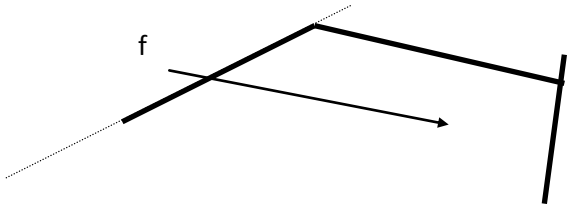


Рис. 3.14

Второй случай - угол между векторами **n** и **(P-f)** равен 90^0:

$$V = \mathbf{n} * (\mathbf{P} - \mathbf{f}) = 0, \ \alpha = \pi/2.$$

Третий случай - угол между векторами **n** и **(P-f)** меньше 90^0:

$$V = \mathbf{n} * (\mathbf{P} - \mathbf{f}) > 0, \ \alpha < \pi/2.$$

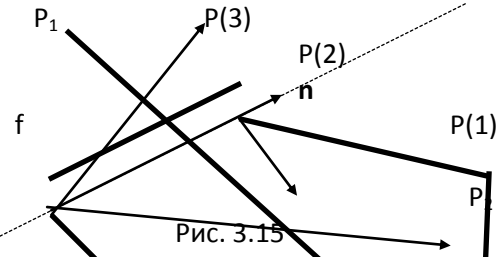


Рис. 3.15

Для построения алгоритма здесь используется параметрическая запись прямой, продолжающей отрезок P_1P_2.

$$\mathbf{P}(t) = \mathbf{P_1} + (\mathbf{P_2} - \mathbf{P_1}) * t,$$

где t - параметр, определяющий положение точки на прямой.

При значениях $0 \le t \le 1$ точка принадлежит отрезку P_1P_2. Если $t < 0$, то точка лежит левее точки P_1, а при $t > 1$ - правее P_2.

В соответствии с этим для точки пересечения прямой с ребром многоугольника (второй случай расположения точки P) можно вычислить параметр t, заменяя точку f на вершину многоугольника A_i:

$$\mathbf{n_i} * (\mathbf{P_1} + (\mathbf{P_2} - \mathbf{P_1}) * t - \mathbf{A_i}) = 0.$$

Из этого выражения после преобразований получаем значение параметра:

$$t = -\frac{n_i * (P_1 - A_i)}{n_i * (P_2 - P_1)}.$$

Выполним анализ полученного выражения. Для этого рассмотрим *случай равенства нулю знаменателя.* Это возможно, когда $P_2 = P_1$ или же в случае расположения отрезка P_1P_2 параллельно ребру многоугольника (угол α между векторами **n** и $\mathbf{P_1P_2}$ равен $\pi/2$). Совпадение точек концов отрезка дает вырожденный случай, который не входит в рассмотрение задачи отсечения отрезка. При

параллельном расположении отрезка возможны три варианта его видимости, определяемые из анализа числителя:

1) $\mathbf{n_i} * (\mathbf{P_1} - \mathbf{A_i}) < 0$. В этом случае точка P_1 находится вне отсекающего многоугольника ($\alpha > \pi/2$), а следовательно, и точка P_2, поэтому отрезок будет полностью невидимым;
2) $\mathbf{n_i} * (\mathbf{P_1} - \mathbf{A_i}) = 0$. Точка P_1 находится на границе отсекающего окна, значит отрезок лежит на грани данного ребра многоугольника, и для выделения видимой части отрезка следует найти общую часть отрезка и ребра;
3) $\mathbf{n_i} * (\mathbf{P_1} - \mathbf{A_i}) > 0$. Точка P_1 находится внутри отсекающего окна, в данном случае рассматриваемое ребро не влияет на видимость отрезка.

В *случае неравенства нулю знаменателя* значение t дает точку пересечения прямой, продолжающей отрезок, с гранью ребра. При значениях $t < 0$ или $t > 1$ точка пересечения находится за пределами отрезка P_1P_2. Тогда при $t < 0$ данное ребро не влияет на видимость отрезка (рис. 3.16).

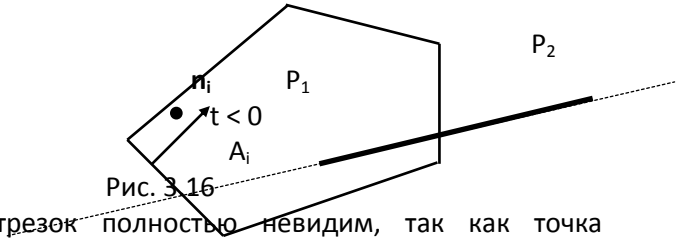


Рис. 3.16

При $t > 1$ отрезок полностью невидим, так как точка пересечения находится правее отрезка (рис. 3.17).

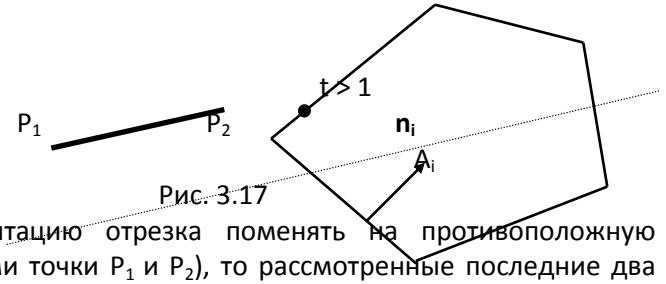


Рис. 3.17

Если ориентацию отрезка поменять на противоположную (поменять местами точки P_1 и P_2), то рассмотренные последние два случая также поменяются местами.

Рассмотрим теперь случай $0 \le t \le 1$. В этом случае отрезок P_1P_2 пересекает ребро многоугольника в точке, определяемой

параметром t. Здесь различаются два случая: во внешней полуплоскости для данного ребра располагается точка P_1 отрезка – имеет место ограничение отрезка снизу, во внешней полуплоскости располагается точка P_2 отрезка – имеет место ограничение отрезка сверху. Эти два случая выявляются вычислением скалярного произведения двух векторов **n** и **$(P_2 - P_1)$**. Если произведение положительно, то имеется ограничесе снизу, при отрицательном значении – сверху.

Прямая, продолжающая отрезок P_1P_2, может пересечь многоугольник только в двух местах. Однако решение уравнения для t относительно всех ребер даст множество решений в интервале $0 \leq t \leq 1$, из которых следует выбрать необходимые. Для этого все решения в интервале $0 \leq t \leq 1$ разбиваются на две группы, которые называются нижней и верхней в зависимости от того, как ограничивается отрезок при пересечении его с ребром многоугольника – снизу или сверху. После разбиения решений выделяют наибольшее значение из нижней группы $t_н$ и наименьшее - из верхней $t_в$, которые рассматриваются, как возможные ограничения снизу и сверху значения точки пересечения. Если окажется, что $t_н > t_в$, то отрезок является полностью невидимым.

На основании описанного алгоритма можно составить программу.

Отсечение отрезка невыпуклым многоугольником во многом совпадает с алгоритмом для выпуклого многоугольника. Сначала вычисляются определители

$$S_i = | \mathbf{A_i P_1 P_2} |, \quad i = 1,2, \ldots , n,$$

которые выявляют расположение вершин многоугольника A_i относительно прямой, продолжающей отрезок P_1P_2. Первый случай, когда все определители имеют один знак или равны нулю, полностью совпадает с предыдущим алгоритмом, а во втором случае, при наличии определителей разного знака, следует выявить ребра, которые пересекает прямая, продолжающая отрезок P_1P_2. Именно эти ребра имеют в своих вершинах определители S_i разного знака (рис. 3.18).

Для определения видимых частей отрезка необходимо:

- найти точки пересечения прямой, продолжающей отрезок, с ребрами многоугольника;

- выделить нечетные интервалы, принадлежащие области многоугольника;

определить принадлежность этих интервалов отрезку.

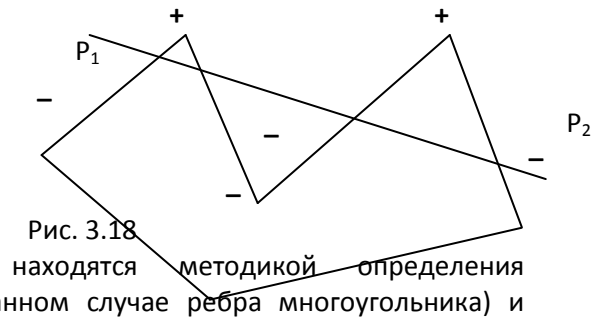


Рис. 3.18

Точки пересечения находятся методикой определения пересечения отрезка (в данном случае ребра многоугольника) и прямой, продолжающей отрезок P_1P_2. Нечетные интервалы находятся упорядочиванием точек пересечений по возрастанию. За первым, третьим и т.д. пересечениями лежат нечетные интервалы. После этого необходимо выяснить, какие части этих интервалов совпадают с отрезком, которые и будут являться видимой частью данного отрезка.

3.3. Отсечение многоугольников

Задача отсечения многоугольников заключается в отсечении области (областей) попадающей (попадающих) в окно отсечения. При этом отсеченная область представляет собой многоугольник, ребрами которого могут быть ребра отсекаемого и отсекающего многоугольников, или их части

Рассмотрим решение задачи отсечения многоугольников выпуклым многоугольником по **алгоритму Сазерленда-Ходжмена**. Идея алгоритма основывается на последовательном отсечении многоугольника полуплоскостями, образованными гранями выпуклой отсекающей области. Каждая полуплоскость может разбивать многоугольник на две части. Та часть, которая попадает во внешнюю полуплоскость, отбрасывается.

Реализация алгоритма заключается в последовательном формировании контура отсеченной области. Процесс начинается с анализа первой вершины отсекаемого многоугольника. Если она находится в отсекающем окне, то ее заносят в контур отсеченной

области. Далее выполняется анализ положения ребер отсекаемого многоугольника по отношению к грани отсечения. При этом может иметь место четыре различных варианта взаимного расположения ребра многоугольника и отсекающей полуплоскости. Рассмотрим эти варианты.

Обе вершины A_i и A_{i+1} ребра находятся вне полуплоскости (рис. 3.19). В этом случае при обходе этого ребра в результирующую последовательность не передается ни одна вершина.

Рис. 3.19

Обе вершины A_i и A_{i+1} находятся внутри полуплоскости (рис. 3.20).

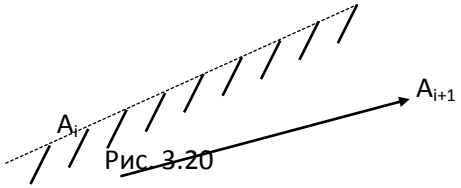


Рис. 3.20

В результирующую последовательность передается вершина A_{i+1}.

Одна вершина A_i находится снаружи, а другая A_{i+1} - внутри полуплоскости (рис. 3.21).

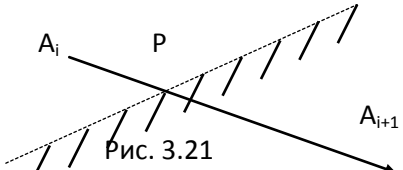


Рис. 3.21

В этом случае в результирующую последовательность передаются точки P и A_{i+1}. Этот случай называется вхождением во внутреннюю полуплоскость.

Одна вершина A_i находится внутри, а другая A_{i+1} - снаружи полуплоскости (рис. 3.22). В результирующую последовательность передается точка P.

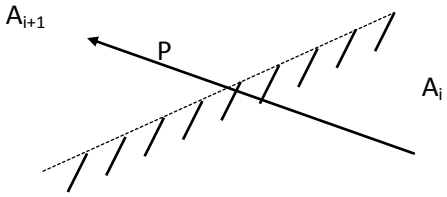


Рис. 3.22

При обработке последнего ребра A_nA_1 в случае входа или выхода из полуплоскости в результирующую последовательность передается точка P пересечения ребра A_nA_1 с гранью полуплоскости. В соответствии с изложенным выше можно построить алгоритм решения задачи отсечения многоугольников выпуклым окном.

Для отсечения невыпуклых многоугольников выпуклым многоугольником можно применять аналогичную процедуру, однако при этом следует учитывать возможность появления лишних ребер в результирующей последовательности, если отсекаемая область состоит из несвязных частей.

Другой а**лгоритм Вейлера-Азертона** позволяет отсекать один невыпуклый многоугольник другим, в общем случае тоже невыпуклым. При этом каждый многоугольник может содержать дыры (отверстия). Идея алгоритма заключается в последовательном обходе вершин многоугольника. При входе в отсекающее окно обход выполняется по ребрам отсекаемого (обрабатываемого) многоугольника, при выходе - по ребрам отсекающего окна до прихода в следующую точку пересечения многоугольников. Алгоритм предусматривает формирование контуров обхода обрабатываемого и отсекающего многоугольников с указанием точек их пересечения. Рассмотрим пример работы алгоритма отсечения многоугольника. На рис. 3.23 жирной линией обозначен обрабатываемый многоугольник $A_1A_2A_3A_4A_5A_{16}$, а отсекающее окно - многоугольник $C_1C_2C_3C_4$ - тонкой линией. Для организации работы алгоритма необходимо отметить точки пересечения ребер многоугольников.

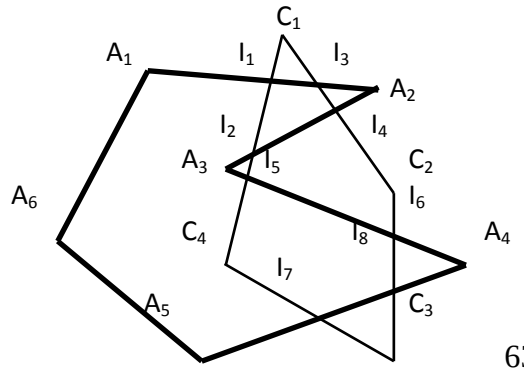


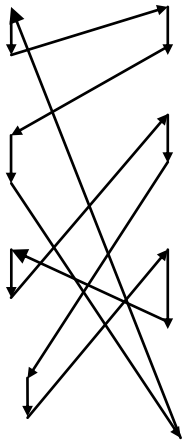
Рис. 3.23

На рис. 3.23 они обозначены как I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8. Эти точки пересечения следует разделить на две группы - на точки входа в отсекающую область (I_1, I_4, I_5, I_8) и на точки выхода из нее (I_2, I_3, I_6, I_7). Поиск отсекаемых областей выполняется по контурам многоугольников, составленных в виде цепочек последовательностей из вершин многоугольников и точек пересечения, полученных обходом многоугольников по часовой стрелке. Движение начинается из любой точки входа по часовой стрелке, расположенной на обрабатываемом многоугольнике до прихода в точку выхода. После этого выполняется переход на отсекающий многоугольник, по которому движение производится до прихода во входную точку. Затем выполняется аналогичный переход на обрабатываемый многоугольник. Эта процедура выполняется до возврата в точку, с которой начиналось движение. Если при этом будут пройдены все точки пересечения, то процесс заканчивается. В противном случае снова выбирается одна из оставшихся входных точек, и процедура повторяется по описанному алгоритму. Покажем это на примере многоугольников, представленных на рис. 23. Контурные цепочки расположены в виде двух столбцов, а последовательность их обхода обозначена стрелками. В результате обхода получены два контура отсеченных областей: I_1 I_3 I_4 I_2 I_1 и I_5 I_6 I_8 I_7 C_4 I_5 (рис. 3.24).

Полученные области можно проверить по рис. 3.23.

Данный метод пригоден и для выделения **внешних отсекаемых областей.** В этом случае следует внести следующие коррективы в процедуру выделения.

A_1	C_1
I_1	I_3
I_3	I_4
A_2	C_2
I_4	I_6
I_2	I_8
A_3	C_3
I_5	I_7
I_6	C_4
A_4	I_5
I_8	I_2



I_7 I_1
A_5 C_1
A_6
A_1

Рис. 3.24

Начинать движение по ребрам многоугольников необходимо с точки выхода, а движение по ребрам отсекающего многоугольника выполнять против часовой стрелки. Покажем это на предыдущем примере (рис. 3.23). В результате обхода (рис. 3.25) ребер многоугольников, начиная с точек пересечения, получены три внешних отсекаемых области - $I_3A_2I_4I_3$, $I_6A_4I_8I_6$ и $I_2A_3I_5C_4I_7A_5A_6A_1I_1I_2$.

A_1 C_1
I_1 I_3
I_3 I_4
A_2 C_2
I_4 I_6
I_2 I_8
A_3 C_3
I_5 I_7
I_6 C_4
A_4 I_5
I_8 I_2
I_7 I_1
A_5 C_1

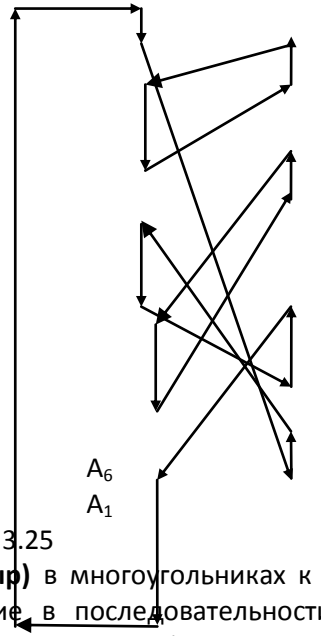
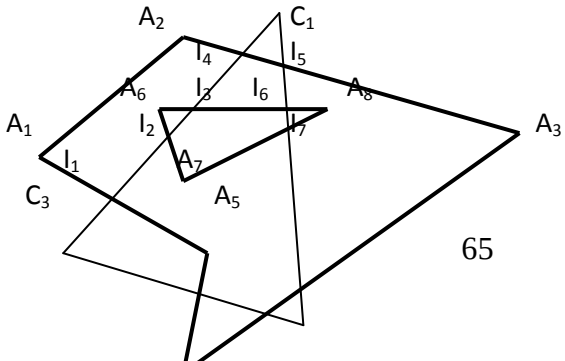


Рис. 3.25

При наличии отверстий (дыр) в многоугольниках к внешним контурам добавляются внутренние в последовательности обхода против часовой стрелки. Рассмотрим работу алгоритма на следующем примере (рис. 3.26).



I_{10} I_8

I_9 C_2

A_4

Рис. 3.26

Точки пересечения многоугольников, как и прежде, разделим на два множества - точки входа I_2, I_4, I_6, I_8, I_{10} и точки выхода I_1, I_3, I_5, I_7, I_9. Расположим последовательности вершин и точек пересечения отсекаемого и отсекающего многоугольников в виде двух столбцов (рис. 3.27). В результате обхода контуров получаем область отсечения I_4 I_5 I_6 I_3 I_4. При этом часть точек пересечения многоугольников не попала в отсеченную область, поэтому процесс поиска областей отсечения необходимо продолжить.

Аналогичный обход последовательностей вершин и точек пересечений обоих многоугольников, например, с точки входа I_8, даст следующую область - I_8 I_9 I_{10} A_5 I_1 I_2 A_7 I_7 I_8. В результате выполненных действий выделены две отсекаемые области, и, т.к. исчерпаны все точки пересечений, алгоритм на этом закончен.

A_1 A_2 I_4 I_5 A_3 I_8 I_9 A_4 I_{10} A_5 I_1 A_1 A_6 I_2 A_7 I_7 A_8 I_6 I_3 A_6

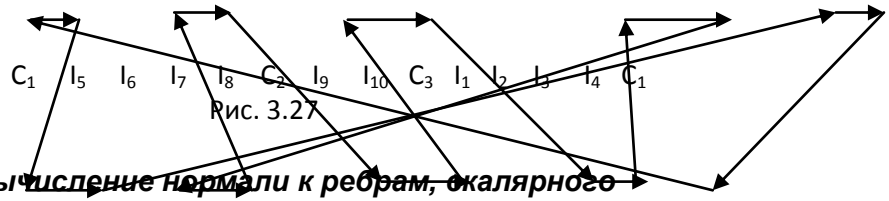


Рис. 3.27

3.4. Вычисление нормали к ребрам, скалярного произведения двух векторов и определение выпуклости многоугольников

Для организации работы по алгоритму Кируса-Бека требуется вычислять внутреннюю нормаль к ребрам многоугольника, которая представляет собой единичный вектор, перпендикулярный к ребру и направленный во внутреннюю область многоугольника. Вычисление нормали $\mathbf{n_i}$ для ребра A_iA_{i+1} можно выполнить операцией поворота вектора данного ребра на угол (-90^0):

$$\mathbf{n_i} = \mathbf{(A_{i+1} - A_i)} * R(-90^0),$$

где $R(-90^0)$ - матрица поворота,

$\mathbf{n(A_i)}$ - вектор нормали ребра A_iA_{i+1}.

Матрицу поворота в однородных координатах можно вычислить на основании матрицы преобразований R(φ):

$$R(\varphi) = \begin{bmatrix} Cos\varphi & Sin\varphi & 0 \\ -Sin\varphi & Cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

После подстановки значения угла φ получаем матрицу

$$R(\varphi) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

При этом вектор ребра представляет собой трехкомпонентный вектор-строку

$$\mathbf{(A_{i+1} - A_i)} = [(x_{Ai+1} - x_{Ai}) \quad (y_{Ai+1} - y_{Ai}) \quad 0].$$

Перемножение матрицы поворота на вектор ребра дает следующий результат

$$\mathbf{n_i} = [(y_{Ai+1} - y_{Ai}) \quad -(x_{Ai+1} - x_{Ai}) \quad 0].$$

Таким образом, компоненты вектора нормали можно определять по координатам ребра. Для этого надо взять разность y-координат ребра в качестве первой компоненты нормали, а разность x-координат с обратным знаком – в качестве второй компоненты.

Вычисление V скалярного произведения двух векторов **a** и **b** в алгоритме Кируса-Бека удобнее производить по координатам векторов, т.к. они всегда известны. Для этого вычисляется сумма x- и y-координат этих векторов:

$$V = \mathbf{a} \times \mathbf{b} = \mathbf{x_a x_b + y_a y_b}.$$

Проверка выпуклости многоугольника выполняется путем вычисления определителей третьего порядка вида

$$S_i = |\mathbf{A_{i+2}\ A_i\ \ A_{i+1}}|,$$

позволяющих определить взаимное положение вершины A_{i+2} относительно ребра $A_i\ A_{i+1}$. Если $S_i < 0$, то вершина лежит справа от ребра при обходе многоугольника по часовой стрелке. Знак этого определителя относится к вершине A_{i+1}. Вычислив определители для всех ребер многоугольника по их знакам, определяют выпуклость данного многоугольника: если знаки имеют значение “минус”, то многоугольник является выпуклым. При невыполнении этого условия многоугольник является невыпуклым, а число положительных знаков

определяет число вершин, нарушающих выпуклость многоугольника. При обходе многоугольника против часовой стрелки знаки определителей и условия выпуклости принимают противоположный смысл.

Если ставится задача разбиения невыпуклого многоугольника на выпуклые части, то это можно сделать продлением одного из смежных ребер вершины, нарушающей выпуклость многоугольника, до пересечения с его ребром. Каждая такая операция уменьшает число вершин, нарушающих выпуклость многоугольника, по крайней мере, на одну. Для полного разбиения многоугольника требуется число операций не превосходящее числа вершин, нарушающих его выпуклость.

4. ВИДЕОСИСТЕМЫ КОМПЬЮТЕРОВ

4.1.Типы графических устройств

Для ввода графической информации в настоящее время широкое распространение получили сканеры, которые позволяют выполнить этот процесс быстро и качественно. Визуализация изображений выполняется видеосистемами, включающие в себя видеоадаптеры и дисплеи.

Графическими устройствами (ГУ) являются различные графопостроители, печатающие устройства (точечно-матричные, электростатические, лазерные), фильмирующие устройства и дисплеи на электронно-лучевых трубках (ЭЛТ). В машинной графике наибольшее применение имеют графические устройства на ЭЛТ, которые бывают трех типов:

- запоминающие дисплеи с прямым копированием изображения (рисование отрезками);
- векторные дисплеи с регенерацией изображения (рисование отрезками);
- растровые сканирующие дисплеи с регенерацией (точечное рисование).

Рассмотрим подробнее характеристики графических дисплеев на ЭЛТ.

4.2. Графические дисплеи на запоминающей трубке

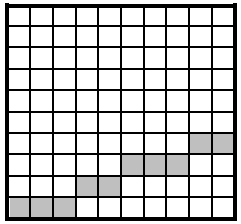
Эти ГУ представляют наиболее простой тип по своему устройству. Дисплеи имеют экран, покрытый люминофором с длительным временем послесвечения до 1 часа. При выводе изображения интенсивность луча увеличивается до величины, способной вызвать свечение следа луча на люминофоре. Стирание изображения выполняется подачей специального напряжения, снимающего свечение люминофора, в результате которого экран вспыхивает и принимает исходное темное состояние. Процедура стирания занимает время около 0.5 с. Такой способ вывода изображения и его стирания делает невозможным удаления части изображения или текста, и поэтому нельзя создавать динамические изображения. Вывод отрезков на запоминающих дисплеях выполняется непосредственно из одной адресуемой точки в другую

вектором. Достоинствами этих ГУ являются простота программирования из-за векторного построения отрезков и точности их прорисовки, простота получения твердой копии изображения, а главным недостатком, как уже говорилось, невозможность обработки части изображения и создания движущихся изображений.

4.3. Векторные графические дисплеи с регенерацией изображений

Векторные дисплеи используют трубки с люминофором, имеющим малое время послесвечения. Они так же, как и запоминающие, рисуют отрезки векторами по адресуемым точкам. Для сохранения изображения при коротком свечении люминофора возникает необходимость регенерации. Минимальная скорость регенерации составляет 30 1/с, а оптимальная 4050 1/с. При скорости меньшей 30 1/с изображение будет мерцающим.

Векторные дисплеи, кроме ЭЛТ, имеют дисплейный буфер и дисплейный контроллер. Дисплейный буфер представляет собой непрерывный участок памяти, содержащий дисплейный файл, а дисплейный контроллер устройство циклической обработки информации для вывода изображения со скоростью регенерации. Сложность изображения ограничивается размерами дисплейного буфера и скоростью контроллера. Для повышения скорости обработки изображений могут использоваться различные методы, например, использование двойного буфера, деление изображения на две составляющие статическую и динамическую. Канал связи центрального процессора и графического устройства должен обладать достаточной шириной полосы пропускания для требуемой скорости регенерации. Ее можно оценить следующим образом. Например, для вывода изображения, содержащего 250 отрезков или точек, в трехмерном пространстве с представлением чисел в форме с плавающей запятой (шесть значащих цифр) по байту на цифру и скоростью регенерации 30 1/с, ширина полосы пропускания составит величину 250 * 3 * 6 * 8 * 30 = 1080000 бит/с, т.е. скорость передачи данных должна превышать 1 Мбит/с. Для более сложных изображений она может составлять значение около 10 Мбит/с. Такие скорости достижимы при наличии прямого доступа в память или параллельного интерфейса. Векторные дисплеи позволяют строить



динамические изображения, однако не могут генерировать сплошные области.

4.4. Растровые графические дисплеи

В отличие от запоминающих и векторных ГУ, рисующих элементы изображений из одной адресуемой точки в другую, растровые дисплеи выполняют аппроксимацию изображения. Растровые дисплеи представляют собой устройство с дискретной матрицей ячеек (точек), каждая из которых может менять свою яркость, т.е. быть видимой невидимой, или цвет. Таким образом, непрерывная линия на растровом дисплее представляется аппроксимированной последовательностью точек растра, близлежащих к реальной линии. При этом изображение становится дискретным. Точки, составляющие изображение, называются элементами изображения, пикселами, или пэлами, от английского picture element. Отрезок на растровом дисплее выглядит прямым только при угле наклона 0, 45 или 90 градусов (рис.5.1).

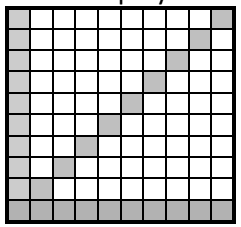


Рис. 5.1

Для растровых дисплеев используется буфер кадра, представляющий собой непрерывный участок памяти и называемый битовой плоскостью. Отдельный разряд битовой плоскости соответствует одному пикселу. Видеопамять может состоять из нескольких битовых плоскостей, каждая из которых будет соответствовать одному полутону или цвету. Объем одной битовой составляет 262144 (2^{18} бита).

Одна битовая плоскость дает черно-белое изображение. Для расширения цветовой гаммы используют необходимое число битовых плоскостей и регистров, которые позволяют выбирать необходимую палитру из некоторой цветовой гаммы. Например, при наличии буфера кадра с 24 битовыми плоскостями и тремя 10-

разрядными регистрами для таблицы цветов можно получить 16777216 (2^{24}) цветовых оттенка из палитры 1073741824 (2^{30}) цветов (около 17 млн. из более 1 млрд.).

Основная проблема для растровых дисплеев состоит в достижении требуемой производительности при работе в реальном масштабе времени и приемлемой скорости регенерации. Например, при средней скорости обращения к одному пикселу 200нс для квадратного растра 512 x 512 требуется время 0.0524 с, что соответствует 19 кадрам в секунду. А для кадра 1024 x 1024 уже 0.21 с (около 5 кадров в секунду). Поэтому для получения приемлемых параметров требуется скорость порядка 2 нс/пиксел. При работе в реальном масштабе времени для улучшения скоростных характеристик используют метод доступа сразу к группе пикселов (16, 32 и более).

Основным достоинством растровых дисплеев является возможность получения плавных переходов цветов и генерации сплошных областей.

4.5. Видеосистемы компьютера

Видеосистема любого компьютера состоит из двух основных частей - видеоадаптера и монитора (рис. 4.1). Видеоадаптер, в свою очередь, включает в себя видеопамять (ВП), в которой хранится изображение, постоянное запоминающее устройство (ПЗУ), предназначенное для хранения наборов шрифтов, а также программы функций базовой системы ввода-вывода (BIOS) для работы с видеоадаптером. Взаимодействие видеопамяти, постоянного запоминающего устройства и системы BIOS с монитором по формированию и выводу изображения осуществляется устройством управления (УУ) адаптера (рис. 4.2).

Для видеоадаптеров существуют различные режимы работы, различающиеся между собой видом формируемого изображения (графическое, текстовое), разрешающей способностью экрана (числом элементов изображения по горизонтали и вертикали), допустимым количеством отображаемых цветов и другими характеристиками.

Сформированное видеоадаптером изображение выводится на экран монитора. Способы его визуализации монитором зависят от

конструкции самого монитора. Наиболее распространенными являются мониторы на электронно-лучевых трубках (ЭЛТ). Блокнотные и переносные варианты компьютеров используют мониторы с жидкокристаллическими и газоразрядными экранами, обладающими малой потребляемой мощностью. Кроме этого, мониторы могут быть монохромными и цветными.

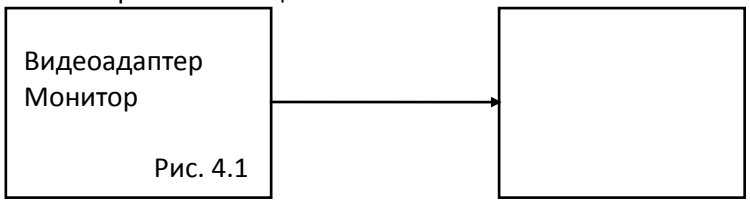


Рис. 4.1

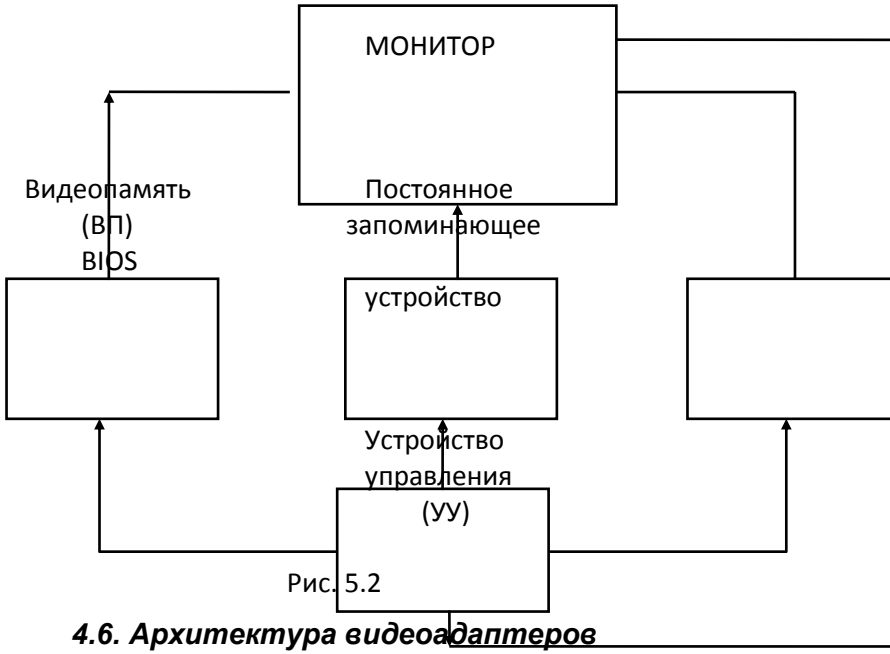


Рис. 5.2

4.6. Архитектура видеоадаптеров

4.6.1.Монохромный адаптер Hercules

Имеет разрешающую способность экрана 720 х 348 точек и две страницы видеопамяти по 32 Кбайта (2^{18} бит) каждая. Пикселю экрана соответствует один бит видеопамяти, что позволяет получать только черно-белые изображения. Каждая страница имеет 4 сегмента со следующими шестнадцатеричными адресами:

страница 0 - $B000 **страница 1** - $B800

$B200 $BA00

$B400 $BC00

$B600 $BE00

Для повышения скорости воспроизведения (частоты кадров) в данном адаптере применяется четырехкратный вывод одного кадра, разбитого на фрагменты изображения (сегменты) через четыре строки. Каждый сегмент содержит такую четверть кадра. Таким образом, один кадр содержится в четырех сегментах страницы. При этом строки кадра распределяются по сегментам следующим образом:

- сегмент B000 содержит строки с номерами 0, 4, 8, ... , 344;
- сегмент B200 содержит строки с номерами 1, 5, 9, ... , 345;
- сегмент B400 содержит строки с номерами 2, 6,10, ... , 346;
- сегмент B600 содержит строки с номерами 3, 7,11, ... , 347.

Другая страница организована аналогично. Как видно из сказанного, сегменты содержат последовательно строки кадра, взятые через четыре строки.

Каждая строка экрана занимает 90 байт (90 x 8 = 720), а каждый байт видеопамяти соответствует 8 пикселям экрана (рис. 4.3).

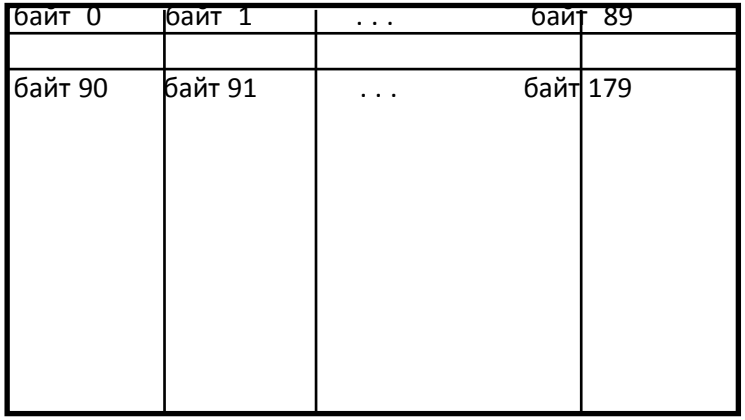


Рис. 4.3

На экране байты отображаются как обычно старшими разрядами вперед, а номера пикселей (координата X) растут слева направо. Соответствие номеров пикселей в пределах одного байта и номеров бит показано на рис. 4.4, а структура координат X и Y пикселя - на рис. 4.5

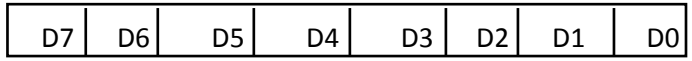
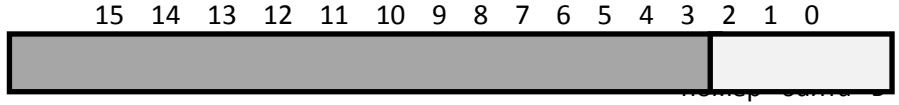


Рис. 4.4

Координата X:



Координата Y:

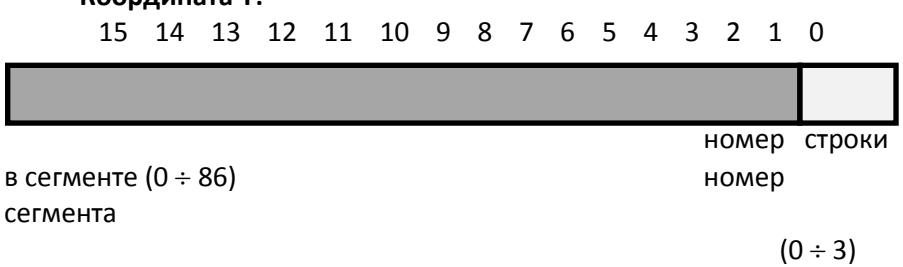


Рис. 4.5

Например, для нулевого байта первой строки пиксели экрана соответствуют следующим разрядам:

пиксель с координатами (0,0) - 7 разряду нулевого байта,
пиксель с координатами (0,1) - 6 разряду нулевого байта,
пиксель с координатами (0,2) - 5 разряду нулевого байта,
пиксель с координатами (0,3) - 4 разряду нулевого байта,
пиксель с координатами (0,4) - 3 разряду нулевого байта,
пиксель с координатами (0,5) - 2 разряду нулевого байта,
пиксель с координатами (0,6) - 1 разряду нулевого байта,

пиксель с координатами (0,7) - 0 разряду нулевого байта.

По такой схеме можно определить местоположение любого пикселя в видеопамяти.

Графический адаптер Hercules представляет собой одну из наиболее удачных моделей видеоадаптеров, которая имеет простую архитектуру и высокое качество черно-белого изображения. Говоря о режимах работы адаптера, которые определяют тип отображаемой информации (текстовый, графический), количество используемых цветов и разрешающую способность экрана и символов, следует подчеркнуть, что адаптер HGC имеет только два режима - текстовый и графический, в отличие от других адаптеров, имеющих более широкий спектр режимов. Ниже приводятся описания архитектуры видеоадаптеров фирмы IBM и их режимов работы.

4.6.2.Видеоадаптер CGA

Представляет собой устаревшую модель. Поэтому здесь приводится лишь краткое описание его архитектуры. Он построен на основе микросхемы Motorola 6845 или ее аналога и включает в себя контроллер ЭЛТ, который устанавливает формат экрана и управляет курсором, световым пером и цветовыми характеристиками изображения. Кроме контроллера, микросхема содержит видеопамять объемом 16 Кбайт. Аппаратное обеспечение воспроизводит изображение на экране, хранящееся в видеопамяти, со скоростью 50 раз в секунду. Процессор может непосредственно обращаться в видеопамять, которая расположена в его адресном пространстве начиная с адреса B800:0000. В некоторых режимах видеопамять разделена на несколько страниц. При этом в каждый момент времени только одна страница может быть либо активной (для нее формируется изображение), либо видимой (ее изображение выводится на экран). Переключение страниц в активное, пассивное или видимое состояние можно выполнять с помощью функций BIOS или путем непосредственного программирования регистров видеоадаптера. Можно предварительно подготовить несколько изображений, записанных в различные страницы видеопамяти, а затем с высокой скоростью вывести их на экран.

Текстовые режимы. В текстовых режимах (0, 1, 2, 3) на экране отображаются текстовые символы и символы псевдографики. Стандартные текстовые режимы позволяют выводить на экран 25 строк по 40 (режимы 0, 1) или 80 символов (режимы 2, 3). Каждый символ на экране кодируется двумя байтами. Первый из них содержит ASCII-код отображаемого символа, а второй - атрибуты символа. Коды символов имеют четные адреса, а их атрибуты - нечетные. Атрибуты определяют цвет символа и цвет фона. Такой способ кодирования текстовой информации дает значительную экономию по сравнению с представлением текстов в графическом режиме.

Три младших разряда 0 - 2 байта атрибутов содержат информацию о цвете символа, а три разряда - 4, 5 и 6 - информацию о цвете фона. Разряды 3 и 7 определяют интенсивность символа и фона соответственно. Кроме этого, разряд 7 может вместо интенсивности фона определять мигание фона символа. ASCII-код преобразуется в двумерный массив пикселей с помощью знакогенератора (таблица трансляции символов).

Режимы 0 и 2 отличаются от режимов 1 и 3 тем, что при использовании композитных мониторов в них подавляется цвет, который заменяется на оттенки серого. При работе с другими мониторами режимы 0 и 1, а также режимы 2 и 3 не различаются.

Знакогенератор видеоадаптера CGA находится в постоянном запоминающем устройстве (ПЗУ), которое расположено вне адресного пространства процессора. В связи с этим нет возможности ни изменить содержимое знакогенератора, ни даже прочитать его. Аналогичное положение и у адаптеров MDA и Hercules. Для “русификации” или изменения шрифтов в текстовых режимах этих видеоадаптеров необходимо перепрограммировать микросхему ПЗУ знакогенератора, расположенную на плате видеоадаптера. Это положение не относится к использованию шрифтов в графических режимах видеоадаптера CGA.

Графические режимы. Распределение видеопамяти в графических и текстовых режимах видеоадаптера CGA различается. Это вызвано необходимостью хранения в графических режимах информации о каждом пикселе изображения. Видеоадаптер поддерживает три графических режима - 4,5 и 6.

Режимы 4 и 5 имеют низкое разрешение 320x200 и используют четыре цвета. Режим 5 отличается от режима 4 подавлением цвета при использовании композитного дисплея и заменой его оттенками серого. Вывод кадра на экран выполняется двумя фрагментами (блоками) - сначала фрагмент, содержащий четные строки, а затем - нечетные. Таким образом, получается выдача изображения на экран чересстрочной разверткой. Оба фрагмента содержатся в одном сегменте памяти с адресом B800h. При этом фрагмент с четными строками кадра располагается по адресу B800:0000h, а фрагмент с нечетными строками - по адресу B800:2000h. Каждому пикселю изображения соответствуют два бита видеопамяти, и один байт содержит информацию о четырех пикселях. В режимах 4 и 5 имеются два набора цветов - стандартный и альтернативный, которые приведены в табл. 4.2.

Таблица 4.2

Значение битов пикселя	Стандартный цвет	Альтернативный цвет
00	Черный	Черный
01	Светло-синий	Зеленый
10	Малиновый	Красный
11	Ярко-белый	Коричневый

Для выбора используемого набора цветов можно воспользоваться функцией 0Bh прерывания INT 10h.

Режим 6 имеет разрешение 640x200 и два цвета изображения. Видеопамять имеет структуру, аналогичную режимам 4 и 5. Четные и нечетные строки изображения хранятся по адресам B800:0000h и B800:2000h. Каждому пикселю соответствует один бит видеопамяти и в каждом байте содержится информация о восьми пикселях изображения.

4.6.3. Видеоадаптеры EGA и VGA

Имеют более сложную архитектуру по сравнению с рассмотренными выше. Условно их можно разбить на шесть логических блоков: видеопамять, графический контроллер, последовательный преобразователь, контроллер ЭЛТ, контроллер атрибутов, синхронизатор (рис. 4.6).

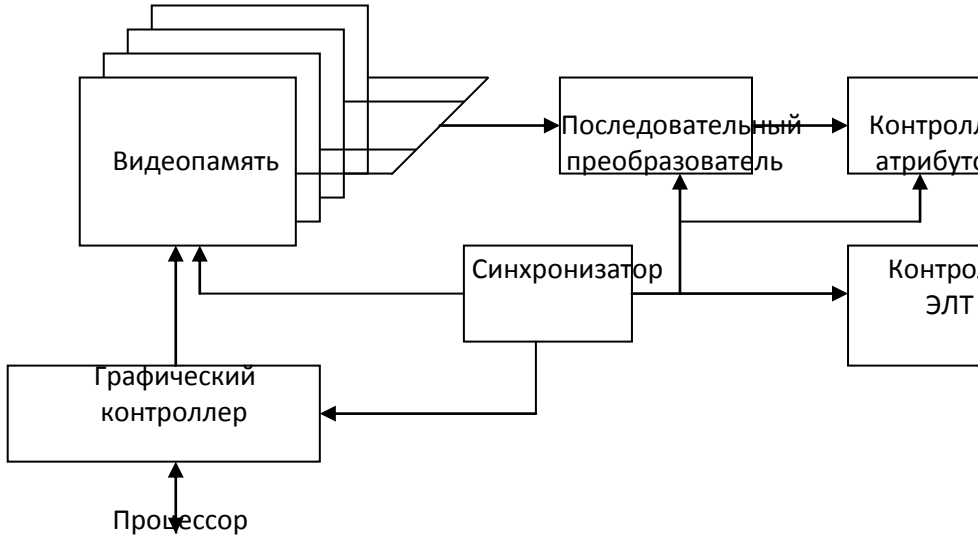


Рис. 4.6

Видеопамять содержит данные, отображаемые адаптером на экране монитора. Объем памяти обычно 256 Кбайт. Видеопамять находится в адресном пространстве процессора. Программы могут непосредственно выполнять с ней обмен данными.

Графический контроллер используется при обмене данными между центральным процессором компьютера и видеопамятью. Аппаратура графического контроллера позволяет выполнять простейшие логические операции (И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ, циклический сдвиг) над данными, поступающими в видеопамять.

Последовательный преобразователь выбирает из видеопамяти один или несколько байт, преобразует их в последовательный поток бит и передает контроллеру атрибутов.

Контроллер ЭЛТ генерирует временные синхросигналы, управляющие ЭЛТ.

Контроллер атрибутов преобразует информацию о цвете из формата, в котором она хранится в видеопамяти, в формат, необходимый для ЭЛТ. Преобразование цветов производится в соответствии с таблицей цветовой палитры (Color Look-up Table). Модифицируя таблицу цветовой палитры, можно выбрать 16 цветов, поддерживаемых видеоадаптером EGA, из 64 цветов, которые может отображать цветной улучшенный монитор. Видеоадаптер VGA также

содержит набор регистров, называемый таблицей цветов. Она позволяет видеоадаптеру VGA отображать на экране монитора пиксели 262 144 различных цветов.

Синхронизатор определяет все временные параметры видеоадаптера и управляет доступом процессора к цветовым слоям видеопамяти.

Видеопамять. Видеоадаптеры EGA и VGA содержат на своей плате до 256 Кбайт оперативной памяти, разделенной на четыре цветовых слоя (банка). Эти слои размещаются в одном адресном пространстве таким образом, что по каждому адресу располагается 4 байта - по одному байту в каждом цветовом слое. Доступ к отдельным цветовым слоям организуется с помощью установки соответствующих регистров адаптера.

Такая организация видеопамяти позволяет производить запись данных сразу во все четыре слоя, получая при этом высокую скорость обработки изображения. При необходимости записи данных только в отдельные слои памяти используют программирование регистра разрешения записи цветового слоя.

Для операции чтения в каждый отдельный момент возможен доступ только к одному цветовому слою, который определяется регистром выбора читаемого слоя.

В большинстве режимов видеопамять состоит из нескольких страниц, с которыми можно работать как и с видеопамятью адаптера CGA.

Текстовые режимы. Стандартные текстовые режимы (табл. 4.9) позволяют выводить на экран 25 строк по 40 символов (25x40) - режимы 0, 1 или 25x80 символов - режимы 2, 3, 7. Перепрограммирование некоторых регистров видеоадаптера позволяет установить для EGA режимы 43x40 и 43x80 символов, а для VGA - 50x40 и 50x80 символов. При наличии видеоадаптера SVGA возможна установка других текстовых режимов - 25x132, 43x132, 50x132, 60x132, 60x80.

Кодирование каждого символа выполняется, как и для адаптера CGA, двумя байтами - один байт содержит ASCII-код, другой - атрибуты символа. При этом память имеет следующую структуру - нулевой цветовой слой видеопамяти содержит ASCII-коды, первый слой - атрибуты, а второй слой - таблицы знакогенератора (рис. 4.7).

Отображение символа на экран сопровождается преобразованием ASCII-кода с помощью таблиц знакогенератора в двумерный массив пикселей. При непосредственном доступе к видеопамяти нулевой и первый цветовые слои располагаются в общем адресном пространстве с чередованием байтов этих слоев - коды символов имеют четные адреса, атрибуты - нечетные.

Таблица 4.9

№ п/п	Режимы	Тип режима	Кол-во цветов	Разрешение, пиксель	Размер символов, пиксель
1.	0, 1	Текстовый цветной	16	40x25	8x8
2.	0*, 1*	Текстовый цветной	16	40x25	8x14
3.	0^+, 1^+	Текстовый цветной	16	40x25	9x16
4.	2, 3	Текстовый цветной	16	80x25	8x8
5.	2*, 3*	Текстовый цветной	16	80x25	8x14
6.	2^+, 3^+	Текстовый цветной	16	80x25	9x16
7.	4, 5	Графический цветной	4	320x200	
8.	6	Графический цветной	2	640x200	
9.	7	Текстовый монохромный	2	80x25	9x14
10.	7^+	Текстовый монохромный	2	80x25	9x16
11.	0Dh	Графический цветной	16	320x200	
12.	0Eh	Графический цветной	16	640x200	
13.	0Fh	Графический монохромный	2	640x350	
14.	10h	Графический цветной	16	640x350	
15.	11h	Графический цветной	2	640x480	
16.	12h	Графический цветной	16	640x480	
17.	13h	Графический цветной	256	320x200	

Режимы 0 и 1. Режимы поддерживают разрешение 25x40. Из-за низкой разрешающей способности практически не используются. Современные видеоадаптеры поддерживают эти режимы только для обеспечения совместимости с видео-

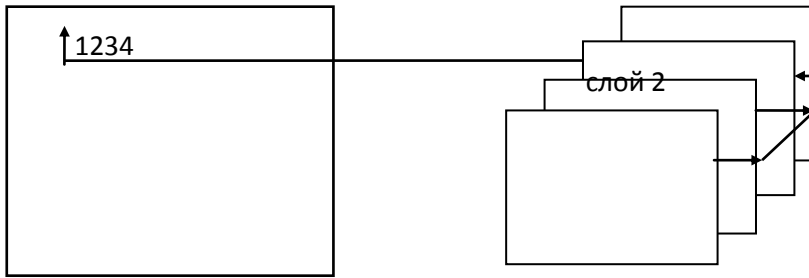


Рис. 7. Структура видеопамяти в текстовых режимах

адаптерами CGA и MDA. Каждый символ отображается матрицей пикселей размером 8x8. Символы могут отображаться в восьми основных и в восьми дополнительных цветах. Дополнительные цвета отличаются от основных большей интенсивностью (табл. 4.3).

Таблица 4.3

Стандартный цвет	**Дополнительный цвет**
Черный	Серый
Синий	Светло-синий
Зеленый	Светло-зеленый
Морской волны	Голубой
Красный	Светло-красный
Фиолетовый	Малиновый
Коричневый	Желтый
Белый	Ярко-белый

Кроме замены набора цветов за счет использования стандартного и дополнительного наборов, имеется возможность изменения этих наборов, выбирая их из палитры в 64 цвета, а для адаптеров VGA и SVGA - из палитры в 262 144 цвета.

Режимы поддерживают 8 страниц видеопамяти, каждая из которых определяет содержимое одного экрана монитора. Одна из страниц является активной и видимой на экране. Изменение активной страницы можно выполнить соответствующей функцией BIOS или изменением содержимого регистра начального адреса, расположенного в контроллере ЭЛТ. Страницы видеопамяти располагаются по следующим адресам:

Страница 0 - B800:0000h Страница 1 - B800:0800h
Страница 2 - B800:1000h Страница 3 - B800:1800h
Страница 4 - B800:2000h Страница 5 - B800:2800h
Страница 6 - B800:3000h Страница 7 - B800:3800h
Страница 0

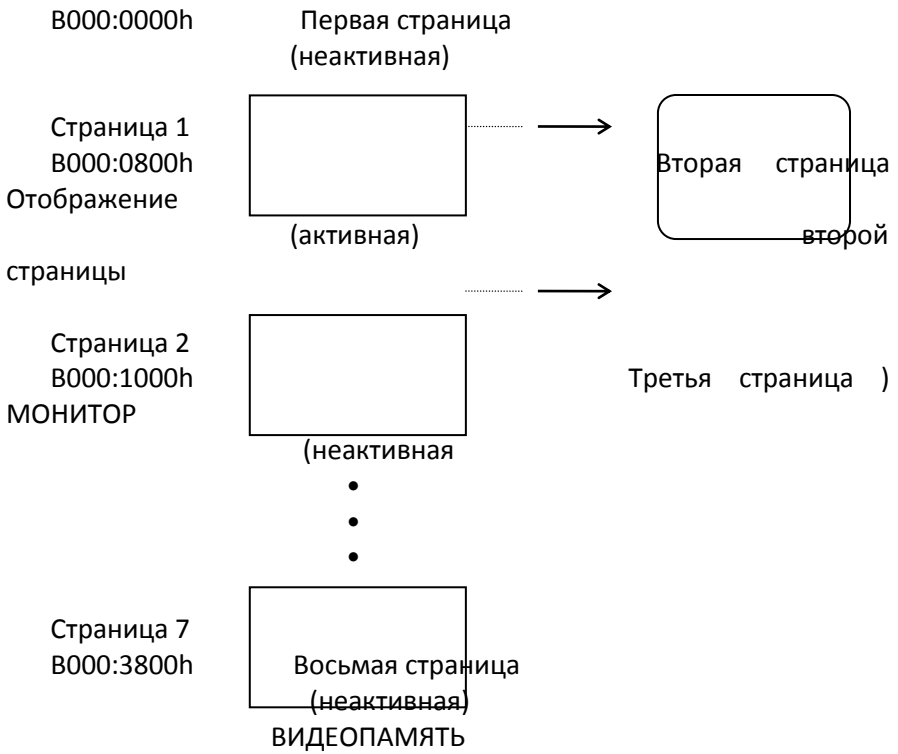


Рис. 4.8

Страничная организация видеопамяти показана на рис.4.8. Видеоадаптеры VGA и SVGA в режимах с низким разрешением используют двойное сканирование, заключающееся в том, что при работе видеоадаптера в режимах с разрешением 200 строк каждая из строк отображается дважды, увеличивая разрешение по вертикали до 400 строк. В результате улучшается восприятие текста, так как фактически увеличивается разрешающая способность. Этот прием используется в режимах 0 - 6, 0Dh и 0Eh.

Для видеоадаптера EGA существует расширенная версия режимов 0 и 1 адаптера CGA (0^* и 1^*), отличающаяся размером матрицы пикселей в изображении символов - 8x14. Аналогичные режимы предусмотрены и для видеоадаптеров VGA и SVGA (0^+ и 1^+), которые имеют матрицу пикселей 9x16. Эти режимы создают некоторые трудности для перенесения программ, написанных для

адаптеров CGA, EGA, VGA и SVGA, связанные с установкой формы курсора и положения линии подчеркивания символов.

Режимы 2 и 3 отличаются от предыдущих разрешением по горизонтали 25x80 символов. Символы текста можно отображать в восьми основных и восьми дополнительных цветах, которые совпадают с цветами режимов 0 и 1. Режимы также поддерживают восемь страниц экрана. Только для видеоадаптера EGA с памятью в 64 Кбайта поддерживаются 4 первые страницы. Страницы видеопамяти имеют следующие адреса.

Страница 0 - B800:0000h	Страница 1 - B800:1000h
Страница 2 - B800:2000h	Страница 3 - B800:3000h
Страница 4 - B800:3000h	Страница 5 - B800:5000h
Страница 6 - B800:4000h	Страница 7 - B800:7000h

Для видеоадаптеров EGA и VGA /SVGA предусмотрены расширенные варианты режимов с разрешением матрицы пикселей 8x14 (2^* и 3^*) и 9x16 (2^+ и 3^+), соответственно, с наличием тех же проблем, как и в режимах 0 и 1. Для видеоадаптеров VGA и SVGA используют двойное сканирование.

Режим 7 отображает монохромную текстовую информацию с разрешением 25x80 символов. Матрица пикселей имеет разрешение 9x14. Данный режим практически полностью программно совместим с видеоадаптером MDA. Поддерживаются 8 страниц видеопамяти, а для EGA с 64 Кбайтами - только первые четыре страницы. Страницы располагаются в видеопамяти по тем же адресам, как в режимах 2 и 3.

Для видеоадаптера SVGA предусмотрен режим 7^+, имеющий размер матрицы пикселей 9x16.

Графические режимы, в отличие от текстовых, работают с отдельными пикселями. Стандартные графические режимы - 4, 5, 6, 0Dh, 0Eh, 0Fh, 10h, 11h, 12h и 13h. Режимы 8, 9 и 0Ah предназначены для видеоадаптеров компьютера Pcjr и в настоящее время интереса не представляют. Режимы 0Bh, 0Ch являются резервными.

Режимы 4, 5 и 6 совпадают с одноименными режимами видеоадаптера CGA и описаны выше.

Режим 0Dh. Разрешающая способность 320x200 пикселей. Совпадает с режимом 4, но в отличие от него может отображать 16 цветов. Каждый пиксель отображается четырьмя битами - по одному

биту в каждом слое. В зависимости от объема памяти видеоадаптера поддерживается разное число страниц: при 256 Кбайтах - 8 страниц, при 128 Кбайтах - 4 страницы и при 64 Кбайтах - 2 страницы. Адреса страниц имеют те же значения, что и в режиме 7. Здесь также используется двойное сканирование.

Режим 0Eh. Разрешающая способность режима 640x200 пикселей. Может отображать одновременно 16 цветов. Каждый пиксель отображается четырьмя битами - по одному биту в каждом слое. В зависимости от объема памяти адаптера поддерживает разное количество страниц, но вдвое меньшее, чем в режиме 0Dh: 256 Кбайт - 4 страницы, 128 Кбайт -2 страницы и 64 Кбайт - 1 страница. Адреса страниц имеют следующие значения: нулевая - A000:0000h, а остальные со сдвигом на 4000h каждая - A000:4000h, A000:8000h, A000:C000h. Используется двойное сканирование.

Режим 0Fh. Монохромный режим с разрешением 640x350 пикселей. Поддерживается две страницы видеопамяти (для EGA с 64 Кбайтами - одна страница). Адреса - A000:0000h (страница 0), A000:8000h (страница 1). Каждый пиксель отображается в памяти двумя битами, в связи с этим режим имеет возможность использования 4 цветов - черный, белый, интенсивно-белый или мигающий.

Режим 10h. Разрешение 640x350 пикселей и 16 цветов (для EGA с 64 Кбайтами памяти - 4 цвета). Набор цветов можно менять, используя палитру, путем установки регистра контроллера атрибутов. Каждый пиксель отображается четырьмя битами - по одному биту в каждом слое.

Режим 11h. Разрешение 640x480 пикселей, монохромное изображение. Каждый пиксель отображается одним битом, при этом используется нулевой слой видеопамяти. Одна страница видеопамяти с адресом A000:0000h.

Режим 12h. Разрешение 640x480 пикселей и 16 цветов. Каждый пиксель отображается четырьмя битами - по одному биту в каждом слое. Одна страница видеопамяти с адресом A000:0000h.

Режим 13h. Разрешение 320x200. Имеется возможность одновременного отображения 256 цветов. Несмотря на низкую разрешающую способность, изображения хорошо воспринимаются за счет богатой палитры цветов. Каждый пиксель отображается

одним байтом. Имеет одну страницу видеопамяти с адресом A000:0000h.

Подавляющее большинство **видеоадаптеров SVGA** обеспечивает полную совместимость с VGA на уровне регистров. Поэтому все программное обеспечение, разработанное для VGA, работает и с SVGA без каких-либо изменений. Естественно, адаптер SVGA имеет большее число регистров, чем видеоадаптер VGA, которые дают ему большие возможности. Использование адаптера SVGA в полной мере возможно за счет соответствующего программирования всех его регистров.

К сожалению, SVGA не является стандартом, как видеоадаптеры EGA и VGA. Различные модели видеоадаптеров SVGA обладают разным набором регистров и набором реализуемых функций. Естественно, это вызывает определенные затруднения при использовании программного обеспечения одной модели для другой. Кроме этого, надо уметь правильно определять тип видеоадаптера.

Ассоциация VESA (Video Electronics Standards Association) разработала стандарт на функции BIOS, позволяющий управлять видеоадаптерами SVGA. Текущая версия стандарта VESA пока не позволяет реализовать все возможности современных видеоадаптеров, например отображать геометрические фигуры с помощью аппаратных возможностей графических акселераторов. Наиболее широкие возможности для использования видеоадаптеров SVGA предоставляет система Windows, в которой используются специальные драйверы, выполняющие всю работу по программированию видеоадаптеров на аппаратном уровне. Обычно драйверы разрабатываются самой фирмой, создавшей видеоадаптер.

4.6.4. Архитектура видеоадаптера SVGA

Видеопамять. Видеоадаптеры SVGA превосходят VGA по разрешению экрана и количеству одновременно отображаемых цветов. В табл. 4.4 приведены характеристики текстовых и графических режимов работы видеоадаптера SVGA в соответствии со стандартом VESA.

Таблица 4.4

Режим работы	Тип режима	Количество цветов	Разрешение (пиксель, символ)
100h	Графический цветовой	256	640x400
101h	Графический цветовой	256	640x480
102h	Графический цветовой	16	800x600
103h	Графический цветовой	256	800x600
104h	Графический цветовой	16	1024x768
105h	Графический цветовой	256	1024x768
106h	Графический цветовой	16	1280x1024
107h	Графический цветовой	256	1280x1024
108h	Текстовый цветовой	16	80x60
109h	Текстовый цветовой	16	132x25
10Ah	Текстовый цветовой	16	132x43
10Bh	Текстовый цветовой	16	132x50
10Ch	Текстовый цветовой	16	132x60
10Dh	Графический цветовой	32 768	320x200
10Eh	Графический цветовой	65 536	320x200
10Fh	Графический цветовой	16 777 216	320x200
110h	Графический цветовой	32 768	640x480
111h	Графический цветовой	65 536	640x480
112h	Графический цветовой	16 777 216	640x480
113h	Графический цветовой	32 768	800x600
114h	Графический цветовой	65 536	800x600
115h	Графический цветовой	16 777 216	800x600
116h	Графический цветовой	32 768	1024x768
117h	Графический цветовой	65 536	1024x768
118h	Графический цветовой	16 777 216	1024x768
119h	Графический цветовой	32 768	1024x768
11Ah	Графический цветовой	65 536	1280x1024

Чтобы иметь возможность отображать большое количество цветов при большой разрешающей способности, видеоадаптер SVGA должен иметь значительно больше видеопамяти, чем адаптер VGA. Например, для реализации режима с разрешением 1024x768 пикселей и возможностью одновременного отображения 65 536 цветов необходимо иметь видеопамять объемом 1,6 Мбайт. Для

доступа центрального процессора к видеопамяти обычно резервируется адресное пространство размером всего 64 Кбайт. Как же процессор получает доступ к видеопамяти, объем которой для некоторых режимов достигает 4 Мбайт? Существует несколько вариантов решения этой проблемы, которые могут комбинироваться.

Вариант слоеного пирога. В большинстве стандартных режимов адаптеров EGA и VGA видеопамять организована из четырех слоев. По каждому адресу расположены сразу 4 байта. Благодаря специальным схемам видеоадаптер может получать доступ к отдельным слоям памяти. Простейший способ вместить в адресное пространство объемом 64 Кбайт память большего размера состоит в увеличении числа слоев видеопамяти. Именно таким образом устроена видеопамять некоторых моделей адаптера SVGA - в 8 или даже в 16 слоев. В этом случае каждый байт определяет 8 пикселей, на каждый из которых приходится 8 бит - по одному из каждого слоя. Такая организация позволяет оперировать с 256 цветами, а для 16 слоев - с 65 536. Но этот путь влечет за собой аппаратурное усложнение и отклонение от стандарта VGA, рассчитанного на 4 слоя.

Вариант адресного окна. Многие современные видеоадаптеры применяют давно известный прием, ранее использовавшийся для подключения к компьютеру дополнительной памяти. Центральный процессор получает доступ к видеопамяти через небольшое окно, которое может иметь небольшой размер (до 64 Кбайт) и располагаться в адресном пространстве процессора. Обычно оно занимает адресное пространство A000:0000h - A000:FFFFh, совпадающее с адресами стандартных цветных режимов видеоадаптеров EGA, VGA и SVGA. Процессор компьютера может перемещать это окно по всей видеопамяти адаптера, получая доступ к разным ее участкам. При такой организации памяти процессор может получать доступ только к отдельной ее части. Чтобы обратиться к другому участку видеопамяти, необходимо переместить окно доступа. Обычно для этого достаточно записать в определенный регистр адаптера положение окна относительно начала видеопамяти.

Доступ к видеопамяти описанным методом создает определенные трудности для программного обеспечения. Для отображения пикселя на экране необходимо вычислять не только положение соответствующей ячейки видеопамяти, но и смещение

окна доступа. Дополнительное усложнение процедуры вывода содержимого видеопамяти на экран создается при работе со сложными изображениями, которые могут не умещаться в одно окно. Аналогичные трудности появляются при копировании изображений из одной позиции в другую, так как при этом нередко возникает необходимость перемещения окна вывода. Для облегчения решения этой задачи некоторые реализации видеоадаптера SVGA отводят для доступа к видеопамяти два окна, обозначаемые как окно А и окно В. Обычно в этих случаях через одно можно только записывать в видеопамять, а через другое - читать из нее.

Кодирование цветов. Видеоадаптер SVGA позволяет работать с большим количеством цветов, зависящим от режима работы. Ниже приводится табл. 4.5, определяющая необходимое число бит для изображения одного пикселя.

Таблица 4.5

Количество цветов	**Количество бит на пиксель**
256	8
32 768	15
65 536	16
16 777 216	24
4 294 967 296	32

Для видеоадаптера VGA в 256-цветном режиме каждый пиксель экрана представляется 8 битами данных видеопамяти. Видеоадаптер содержит таблицу цветов (набор из 256 регистров цифроаналогового преобразователя), которая согласно значениям, записанным в ней, преобразует 8-битовые данные видеопамяти в три 6-битовых сигнала. Эти три сигнала поступают на три ЦАП и вырабатывают красный, зеленый и синий компоненты, определяющие цвет пикселя. Таблица цветов позволяет выбирать для одновременного применения на экране монитора любые 256 цветов из 262144 возможных. Видеоадаптеры SVGA для каждого пикселя используют больший объем данных. Обычно на один пиксель выделяется 15, 16 или 24 бита. В этих условиях сложно использовать таблицу цветов, так как , например, для 16 бит

потребуется 65536 18-битовых регистров. Поэтому в большинстве режимов реализована схема прямого кодирования цвета (Direct Color Mode). При этом биты, определяющие пиксель, формируются в три основные группы, соответствующие красной, зеленой и синей компоненте цвета. Эти группы передаются на три ЦАП, которые формируют видеосигнал.

В некоторых режимах используется дополнительная, четвертая группа бит, соответствующая каждому пикселю. Как правило, четвертая группа бит не задействована, и некоторые модели видеоадаптеров используют ее по своему усмотрению. Основные группы могут содержать по 5-6 бит, а резервная – 1 бит при выделении двух байт на пиксель или по 8 бит (основные и резервные) при кодировании тремя и четырьмя байтами на пиксель.

4.6.5. Программирование видеоадаптеров

Программирование видеоадаптеров заключается в формировании соответствующего бита видеопамяти при построении изображения или в определении значения некоторого пикселя при анализе изображения. Каждый пиксель экрана **адаптера Hercules** имеет адресуемые координаты X, Y в следующих пределах:

$$0 \le X \le 719, \quad 0 \le Y \le 347.$$

Имея координаты некоторого пикселя, можно определить местонахождение соответствующего ему бита в видеопамяти. Если взять двоичные записи координат в виде машинных слов, то по координате X можно определить номер байта в соответствующей строке экрана и номер пикселя в байте, а по координате Y - номер строки в сегменте и номер сегмента (рис. 4.5).

Для работы с точкой на экране необходимо найти адрес соответствующего бита в видеопамяти, который задается адресом сегмента, смещением относительно начала сегмента и номером бита в байте. Кроме этого, часто используют байт маски, в котором нужный бит равен 1, а остальные - нулю. Процедуру определения адреса видеопамяти на языке Pascal можно представить в следующем виде:

```
var
  Segment: word;  {адрес сегмента}
  Offset:    word;  {смещение байта в сегменте}
```

```
  Bit:      word;  {номер бита в байте}
  Mask:     byte;  {маска для бита}
Segment := $B000 + (y and 3 ) shl 9;
Offset  := (y shr 2) x 90 + (X shr 3);
Bit     := 7  (X and 7);
Mask    := 1 shl Bit;
```

Последние два оператора можно заменить на один:

```
Mask := $80 shr (X and 7)
```

или на

```
Mask := 1 shl ((X and 7) ⊕ 7)
```

Чтение цвета точки (X, Y) можно выполнить с помощью функции Mem следующим образом:

```
Color := (Mem [Segment : Offset] and Mask) shr Bit,
```

в котором оператор Mem позволяет работать с памятью как с массивом. Индексы массива задаются в форме [сегмент:смещение]. Эта функция возвращает значение байта памяти, расположенного в указанном сегменте и смещенного на заданную величину.

Запись значения пикселя в видеопамять выполняется также с помощью функции Mem. При этом можно использовать вывод пикселей на экран различными стилями. Для этого необходимо выполнить логические операции над соответствующими битами памяти. Вывод точки цветом 1 стилями MOV, XOR и NOT выполняется следующим образом.

Стиль MOV:

```
Mem [Segment:Offset] := Mem [Segment:Offset] or Mask;
```

Стиль XOR:

```
Mem [Segment:Offset] := Mem [Segment:Offset] xor Mask;
```

Стиль NOT:

```
Mem [Segment:Offset] := Mem [Segment:Offset] and not Mask.
```

При этом стиль MOV будет выводить линию цветом 1 (текущим цветом), стиль XOR - цветом, инверсным к цвету фона, а стиль NOT - цветом, инверсным по отношению к текущему (цветом 0). Выполняемые при этом логические операции и значения бита видеопамяти представлены в табл. 4.6.

Таблица 4.6

Исходное	Цвет вывода	Результирующее значение памяти

значение памяти	точки			
		Стиль OR	Стиль XOR	Стиль NOT
0	1	1	1	0
1	1	1	0	0

Расчет адреса сегмента, смещения и номера бита для **адаптеров CGA, EGA и VGA** зависит от количества бит, выделяемых на один пиксель, способа развертки кадра (построчная, чересстрочная) и разрешающей способности данного режима. Эти вычисления выполняются по такой же схеме, которая была рассмотрена выше для видеоадаптера Hercules. Отличия связаны с особенностями в структуре видеопамяти в разных режимах работы. Если у адаптера Hercules требуется вычисление адреса сегмента для каждого блока одной страницы видеопамяти, то в рассматриваемых адаптерах этот адрес фиксированный: в режимах 4, 5, 6 и 0Dh он равен B800h, в режимах 0Eh, 0Fh, 10h, 11h, 12h и 13h - A000h. В многостраничных режимах каждая страница имеет свое смещение относительно начала сегмента. Для режима 0Dh смещение равно 1000h байт, для режима 0Eh - 4000h, для режима 0Fh - 8000h. Смещение байта, в котором расположены биты данного пикселя, определяется расположением блока, числом полных строк в блоке, предшествующих текущей строке экрана, а также смещением байта в текущей строке. Кроме этого, следует еще учитывать смещение страницы для многостраничного режима, если используется ненулевая страница. В приводимых расчетах предполагается использование именно нулевой страницы. В других случаях следует сделать соответствующую корректировку, связанную со смещением данной страницы.

Структура координат пикселя показана на рис. 4.9. Левая часть координаты (серая) определяет смещение байта в текущей строке. Вторая часть координаты (темная) определяет номер пикселя в байте. В зависимости от количества бит, выделяемых на пиксель, и числа используемых слоев видеопамяти эта часть может иметь нуль, два или три бита.

Причем нуль соответствует режиму 13h, два - режимам 4 и 5, а три бита - всем остальным режимам. Координата Y также имеет две определяющие

Рис. 4.9

части. Серая часть указывает число полных строк блока памяти, а темная указывает номер блока. Под номер блока выделяется нуль или один бит. Последний случай имеет место для режимов 4, 5 и 6, в которых кадр отображается на экране двумя фрагментами (чересстрочная развертка). Во всех остальных режимах используется строчная развертка кадра за один проход, и кадр состоит из одного блока.

С учетом изложенного, расчет смещения и маски в различных режимах может быть выполнен следующим образом:

Режимы 4 и 5.

Offset := (Y and 1) shl 13 + (Y shr 1) • 80 + (X shr 2);
Mask := 3 shl (2 • (X and 3) xor 3).

Режим 6.

Offset := (Y and 1) shl 13 + (Y shr 1) • 80 + (X shr 3);
Mask := 1 shl ((X and 7) xor 7).

Режим 0Dh.

Offset := 40 • Y + (X shr 3);
Mask := 1 shl ((X and 7) xor 7.

Режимы 0Eh, 0Fh, 10h, 11h и 12h.

Offset := 80 • Y + (X shr 3);
Mask := 1 shl ((X and 7) xor 7.

Режим 13h.

Offset := 320 • Y + X.

В последнем случае вычисление маски не требуется, так как байт содержит информацию только об одном пикселе.

Организация чтения и записи для видеоадаптера **CGA** выполняется, как и для видеоадаптера Hercules, с помощью функции Mem. Следует только учесть, что в режимах 4 и 5 (цветные режимы) каждый пиксель отображается двумя битами. Поэтому перед записью следует обнулить соответствующие биты в байте памяти, а азатем записать в них значение цвета пикселя. В режиме 6 все действия совпадают с видеоадапетром Hercules. Для видеоадапетров EGA, VGA и SVGA задача чтения и записи выглядит сложнее.

4.6.6. Чтение и запись для видеоадаптеров EGA, VGA и SVGA

Между программированием видеоадаптеров CGA, Hercules и адаптерами, у которых видеопамять организована в виде цветовых слоев, существуют принципиальные различия. Адаптеры ранних моделей допускают возможность прямого доступа к видеопамяти. Адаптеры VGA и другие позволяют обращаться к данным, записанным в видеопамяти, или формировать их только через специальные четыре 8-битовые регистры-защелки. Каждый адрес видеопамяти относится одновременно к байтам четырех цветовых слоев. Поэтому такое обращение приводит к обработке четырех байт видеопамяти. При выполнении операций записи данных в видеопамять и считывания содержимого памяти по какому-либо адресу участвуют регистры видеоадаптера. Видеоадаптер содержит следующее число регистров:

* блок синхронизации - 6,
* контроллер ЭЛТ - 28,
* графический контроллер - 12,
* контроллер атрибутов - 7,
* последовательный преобразователь - 7.

Кроме этого, используются еще 5 внешних регистров, которые адресуются по адресам своих портов без использования индексного регистра. Несмотря на наличие большого числа регистров видеоадаптера, для построения изображений используется в основном 5-6 регистров. Обычно это регистры графического контроллера, которые приводятся ниже в табл. 4.7.

Графический контроллер выполняет обмен данными между видеопамятью и процессором. Над данными, поступающими в видеопамять, контроллер может выполнять простейшие логические операции: И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ, операцию циклического сдвига.

Таблица 4.7

№ п/п	Порт	Индекс регистра	Наименование регистра	Значение по умолчанию
1.	3CEh	-	Индексный регистр (ИР)	00h
2.	3CFh	0	Установка/сброс (УС)	00h
3.	3CFh	1	Разрешение уст./сбр. (РУС)	00h
4.	3CFh	2	Сравнение цветов (СЦ)	00h
5.	3CFh	3	Циклич. сдвиг данных (ЦСД)	00h
6.	3CFh	4	Выбор читаемой пл-ти (ВЧП)	00h
7.	3CFh	5	Режим чт/зап (РЧЗ)	00h
8.	3CFh	6	Смешанный (С)	05h
9.	3CFh	7	Цвет безразличен (ЦБ)	0Fh
10.	3CFh	8	Битовая маска (БМ)	FFh

Операция записи сопровождается предварительно чтением данных из видеопамяти и последующим запоминанием их в регистрах-защелках (шлюзах), расположенных на плате видеоадаптера. В дальнейшем над данными, находящимися в этих регистрах, и над данными, поступающими от процессора, могут выполняться следующие операции:

- запись данных процессора в видеопамять без изменения;
- циклический сдвиг данных, записываемых процессором в видеопамять;
- выполнение логической операции И между данными, записываемыми в видеопамять, и содержимым регистров-защелок. В видеопамять записывается результат операции;
- выполнение логической операции ИЛИ между данными, записываемыми в видеопамять, и содержимым регистров-защелок. В видеопамять записывается результат операции;
- выполнение логической операции ИСКЛЮЧАЮЩЕЕ ИЛИ между данными, записываемыми в видеопамять, и

содержимым регистров-защелок. В видеопамять записывается результат операции.

Таким образом, видеоадаптер может выполнять частичную обработку видеоданных. Байт, записываемый процессором в видеопамять, поступает в графический контроллер. В соответствии со значением регистра циклического сдвига и выбора логической операции происходит циклический сдвиг содержимого байта, записываемого в видеопамять. Полученный результат и содержимое регистров-защелок обрабатывается в соответствии с выбранной логической операцией.

Дальнейшие преобразования происходят в соответствии со значениями регистра разрешения установки/сброса и регистра установки/сброса:

- если бит регистра разрешения установки/сброса, управляющий данным цветовым слоем, равен нулю, тогда байт, записываемый в видеопамять, не изменяется;
- если бит регистра разрешения установки/сброса, управляющий данным цветовым слоем, равен единице, то в видеопамять записывается байт, все биты которого устанавливаются в соответствии со значением регистра установки/сброса для данного цветового слоя.

Затем, в соответствии с состоянием регистра битовой маски, происходит запись данных в видеопамять:

- если данный бит регистра битовой маски содержит единицу, то соответствующие биты для каждого из цветовых слоев поступают от центрального процессора;
- если данный бит регистра битовой маски содержит нуль, то соответствующие биты для каждого из цветовых слоев поступают из регистров-защелок.

При выполнении **операции чтения** данных из видеопамяти графический котроллер может выполнять операцию сравнения цветов, которая позволяет найти на экране пиксели определенного цвета. В отличие от обычной операции чтения, когда за один раз читается только один цветовой слой, при выполнении операции сравнения цветов графический контроллер имеет доступ ко всем четырем цветовым слоям одновременно. Графический контроллер

сравнивает данные из четырех цветовых слоев (или из любого подмножества цветовых слоев) с данными в регистре сравнения цветов контроллера и в случае совпадения вырабатывает определенный сигнал.

Схема взаимодействия процессора, видеопамяти и регистров палитры показана на рис. 4.10 и 4.11, из которых видно, что в работе участвуют регистры видеоадаптера, регистры-защелки, регистр процессора, содержащий данные для видеопамяти, а также регистры палитры цветов. Причем регистры палитры цветов для адаптера EGA содержат цвет, а для VGA - номер регистра ЦАП, в котором записан нужный цвет. Следует подчеркнуть, что значение пикселя, выбираемое из соответствующих бит всех цветовых слоев, представляет собой номер регистра палитры цветов (рис. 4.11). Таким образом, просматриваются следующие цепочки получения цвета: для EGA - значение пикселя в видеопамяти – это номер регистра палитры цветов - цвет; для VGA - значение пикселя в видеопамяти – номер регистра палитры цветов - номер регистра ЦАП - цвет.

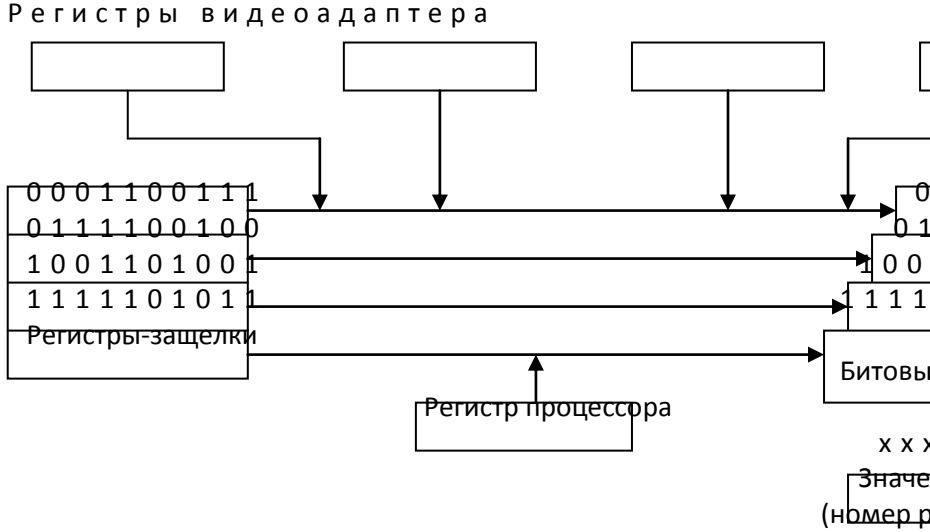
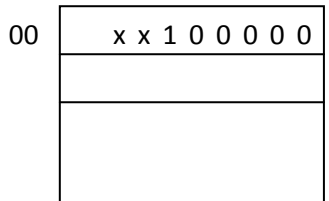


Рис. 4.10



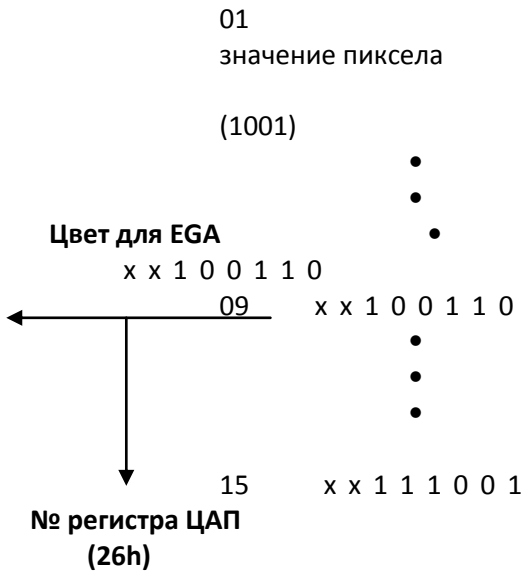


Рис. 4.11

Доступ ко всем регистрам видеоадаптера осуществляется косвенно - через индексный регистр (ИР): для регистров графического контроллера в порт 3CEh засылается индекс требуемого регистра (см. табл. 4.5), а в порт 3CFh - значение, которое заносится в этот регистр. Например, запись значения FFh в регистр битовой маски выполняется следующим образом:

```
{запись в ИР индекса регистра БМ}
MOV    DX, 3CEh
MOV    AL,  8
OUT    DX,  AL
{запись в регистр БМ значения FFh}
MOV    DX, 3CFh
MOV    AL,  Ffh
OUT    DX,  AL
```

Запись можно выполнить и более экономным способом, если учесть структуру регистра AX (рис. 4.12).

```
MOV    DX, 3CEh
MOV    AX,  FF08h
OUT    DX,  AX
```

На языке Паскаль запись выглядит еще короче.

```
PortW[$3CE] := $FF08;
```

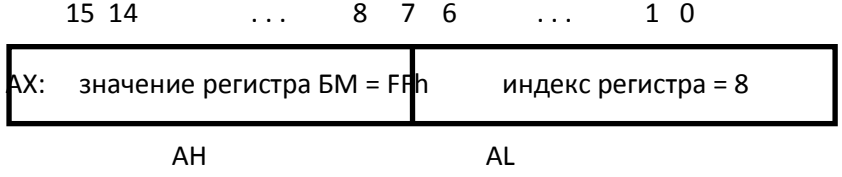


Рис. 4.12

Кроме регистров графического контроллера, часто используется один из регистров блока синхронизации - регистр “Маскирование плоскостей” (МП), четыре младших разряда которого маскируют разрешение-запрещение доступа к четырем битовым плоскостям во время записи. Доступ к регистру выполняется записью его индекса (2) в порт 3C4h, а значения - в порт 3C5h. Значение регистра по умолчанию - 0Fh. Для записи значения по умолчанию надо выполнить следующее:

```
MOV    DX, 3C4h
MOV    AX, 0F02h
OUT    DX, AX
```

или

```
PortW[$3C4] := $0F02;
```

Работа с указанными регистрами позволяет обрабатывать байт какой-либо битовой плоскости, или пиксель с одноименными битами всех плоскостей (побайтовая или попиксельная обработка). При этом можно копировать содержимое регистров-защелок в видеопамять и обратно, пересылать содержимое любого регистра-защелки в регистр процессора, комбинировать четырехбитовое значение из регистра процессора с любым или всеми значениями пикселей в регистрах-защелках, использовать восьмибитовое значение регистра процессора в качестве маски, указывающей, какой из восьми пикселей , записанных в регистрах-защелках, пересылается в видеопамять и т.д.

Побайтовые и попиксельные операции в определенной последовательности используются видеоадаптером в запрограммированных режимах записи и чтения. Для видеоадаптера EGA предусмотрены режимы чтения 0, 1 и записи 0, 1, 2; для видеоадаптера VGA - режимы чтения 0, 1 и записи - 0, 1, 2, 3. Режимы

устанавливаются записью соответствующего значения в регистр “Режим чтения-записи”: в биты 0 и 1 записывается номер режима записи, а в бит 3 - номер режима чтения. По умолчанию BIOS устанавливает в РЧЗ режимы чтения и записи - 0.

Режим чтения 0. Схема режима “Чтение 0” содержимого видеопамяти представлена на рис. 4.13. Вначале происходит загрузка регистров-защелок из соответствующих адресов битовых плоскостей, а из них - в регистр процессора. Выбор соответствующего регистра-защелки производится путем записи номера плоскости в регистр “Выбор читаемой плоскости” (ВЧП), который используется только в режиме “Чтение 0”. Таким образом, в этом режиме идет побайтовая обработка видеопамяти, в результате которой можно прочитать содержимое восьми бит, расположенных в одном цветовом слое (битовой плоскости). Значение пикселя можно сформировать последовательным извлечением байт всех битовых плоскостей и выделением в них требуемых разрядов.

Ниже приводится функция языка Паскаль для организации рассматриваемого режима **“Чтение 0”.**

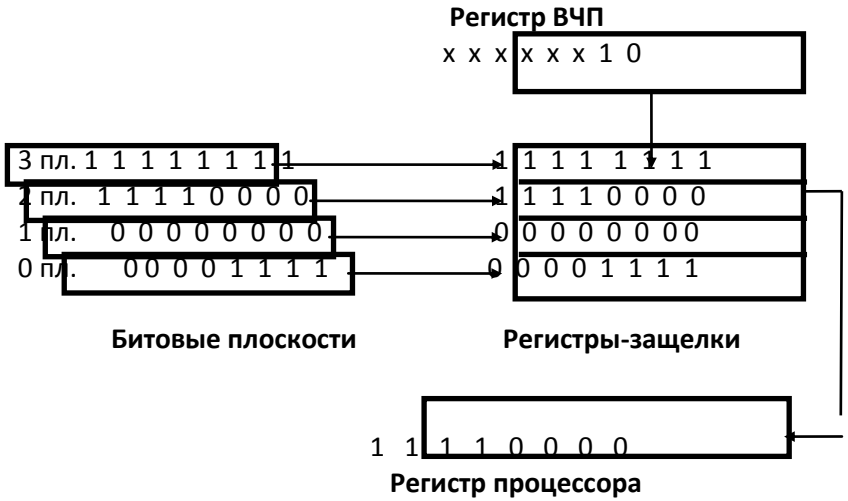
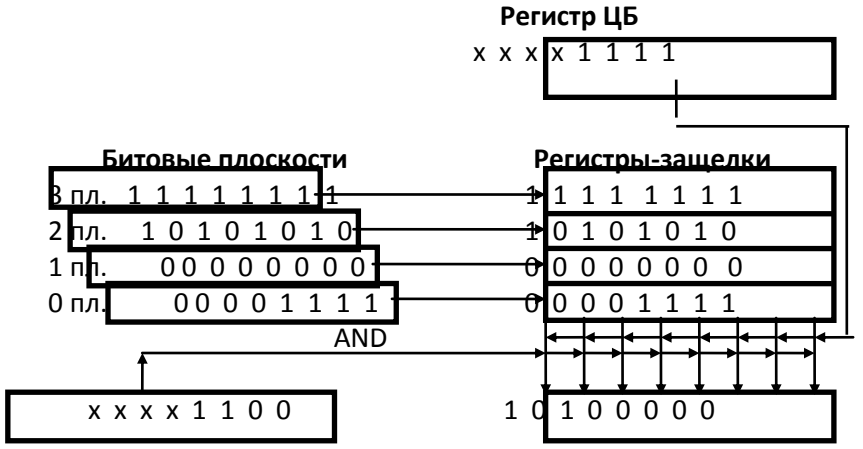


Рис. 4.13

```
Function GetPixel0 (X,Y: integer): word;
{используется регистр ВЧП, значение пикселя формируется по
 одному биту}
var
```

```
    I:    integer;
    R:    registers;
    Off: word;
  begin
    Off    := 80 • Y + X shr 3;          {вычисление смещения}
    R.AH := 1 shl ((X and 7) xor 7);  {AH = битовая маска}
    R.CL := 1;                    {CL - бит в пикселе, соответствующий
                                  читаемой плоскости}
    R.CH := 0;                {обнуление накопителя значения пикселя}
    R.BL := 4;                {BL = индекс регистра ВЧП}
  {чтение битовых плоскостей выполняем последовательно}
  for I := 0 to 3 do
  begin
    R.BH := I;                {BH = номер текущей битовой плоскости}
    PortW[$3CE] := R.BX;                  {установка регистра ВЧП}
    if     Mem[$A000:Off] and R.AH <> 0
    then  R.CH := R.CH or R.CL; {если бит пикселя в соответствующей
плоскости ≠ 0, то этот бит заносим в накопитель значения пикселя}
     R.CL := R.CL shl 1       {перемещаем бит в следующую позицию}
  end;
  GetPixel0 := R.CH                     {присвоение значения пикселя}
  PortW[$3CE] := $0004       {восстановление значения регистра ВЧП
                             по умолчанию}
  end;
```



Регистр СЦ **Регистр процессора**

Рис. 4.14

Режим чтения 1. В этом режиме вначале также выполняется пересылка байтов из битовых плоскостей в регистры-защелки. Затем все восемь пикселей в регистрах-защелках сравниваются со значением, записанным в регистре “Сравнение цветов” (СЦ). Результат сравнения в виде одного байта пересылается в регистр процессора. Этот байт формируется так, что при совпадении цветов пикселя и регистра СЦ устанавливается 1, при несовпадении - 0 (рис. 4.14). На результат сравнения влияет содержимое младших разрядов (0 - 3) регистра “Цвет безразличен” (ЦБ). Если какой-либо бит установлен в 0, то соответствующий регистр-защелка исключаются из сравнения. В примере, показанном на рис. 4.14, четыре младших разряда регистра ЦБ имеют значение ‘1111’, поэтому в сравнении содержимого регистров-защелок и регистра СЦ участвуют все битовые плоскости. Если изменить содержимое регистра ЦБ на значение ‘1011’, то результат обработки, записанный в регистр процессора, будет иметь значение ‘11110000’.

Функция реализации режима **“Чтение 1”** приводится ниже.

```
Function GetPixel1 (X,Y: integer): word;
{значение пикселя формируется по 1 биту, биты читаются из битовых плоскостей поочередно}
var
  I:    integer;
  R:    registers;
  Off: word;
begin
  Off   := 80 • Y + X shr 3;          {смещение}
  R.AH := 1 shl ((X and 7) xor 7);   {AH - битовая маска}
{установка регистра РЧ3}
  PortW[$3CE] := $0805;    {установка индекса РЧ3 = 5, установка
                            режима “Чтение 1”}
{установка регистра СЦ}
  PortW[$3CE] := $0F02;     {установка индекса СЦ = 2, установка
                             цвета = 1111B}
```

```
    R.CH := 0;              {обнуление накопителя значения пикселя}
    R.BL := 7;                  {установка индекса ЦБ = 7}
    R.BH := 1;  {установка начального значения регистра ЦБ -
                 начинаем с битовой плоскости = 0}
    for I := 0 to 3 do
    begin
                 PortW[$3CE] := R.BX;     {установка регистра ЦБ}
  {проверяем, установлен ли бит в пикселе для текущей БП. Если
установлен, то заносят бит в текущую позицию накопителя значения
пикселя}
                 if Mem [$A000:Off] and R.AH <> 0
                 then R.CH := R.CH or R.BH;
                 R.BH := R.BH shl 1;       {BH = номер следующей БП}
    end;
  GetPixel := R.CH;                 {присвоение значения пикселя}
  {восстановление значений регистров по умолчанию}
  PortW[$3CE] :=$0002;     {установка регистра СЦ}
  PortW[$3CE] :=$0005;     {установка регистра РЧЗ}
  PortW[$3CE] :=$0F07;     {установка регистра ЦБ}
  end;
```

Режим записи 0. Данный режим самый сложный и самый богатый по предоставляемым возможностям. Он используется в тех случаях, когда необходимо изменить какой-либо один из восьми пикселей, зашлюзованных (записанных) в регистрах-защелках. В формировании значения пикселя принимают участие регистры “Разрешение установки-сброса” (РУС), “Установка-сброс” (УС), “Битовая маска” (БМ), “Циклический сдвиг данных” (ЦСД), а также байт данных процессора.

Существует широкий набор вариантов работы в этом режиме, определяемый содержимым регистра РУС. Из всего множества способов записи можно выделить два, при которых значение РУС равно 0000B и 1111B. В первом случае все четыре байта в регистрах-защелках перед записью в битовые плоскости комбинируются с байтом данных из регистра процессора (побайтовая комбинация). Во втором случае значение каждого пикселя комбинируется с четырехбитовым значением в регистре “Установка-сброс” (попиксельная комбинация). В обоих случаях в формировании

результата участвуют регистры “Битовой маски” и “Циклического сдвига данных”. Промежуточные значения регистра “Разрешение установки-сброса” представляют собой сложный вариант программирования и используются редко.

Регистр БМ формирует значение каждого пикселя в регистрах-защелках. Если какой-либо бит регистра установлен в 0, то соответствующий пиксель без изменения пересылается из регистров-защелок в видеопамять. При установке бита в 1 значение пикселя перед пересылкой в видеопамять комбинируется либо с байтом данных процессора, либо со значением в регистре “Установка-сброс”.

Способ комбинирования задается регистром ЦСД, который выполняет две функции - указывает логическую операцию и величину циклического сдвига. В битах 3 и 4 этого регистра указывается поразрядная логическая операция в соответствии с таблицей 4.8.

Таблица 4.8

Разряды		**Функция**
Четвертый	**Третий**	
0	0	Замещение
0	1	AND
1	0	OR
1	1	XOR

Эти операции выполняются над байтом данных и содержимым регистров-защелок (при РУС = 0) или над значением регистра УС и содержимым регистров-защелок (при РУС = 1111B).

Биты 0 - 2 регистра ЦСД определяют величину циклического сдвига вправо байта данных из регистра процессора перед комбинированием его с содержимым регистров-защелок.

Рассмотрим работу режима **“Запись 0”** при значении регистра РУС = 1111B (рис. 4.15). В этом случае в регистре УС фактически содержится четырехбитовое значение пикселя, которое пересылается в видеопамять.

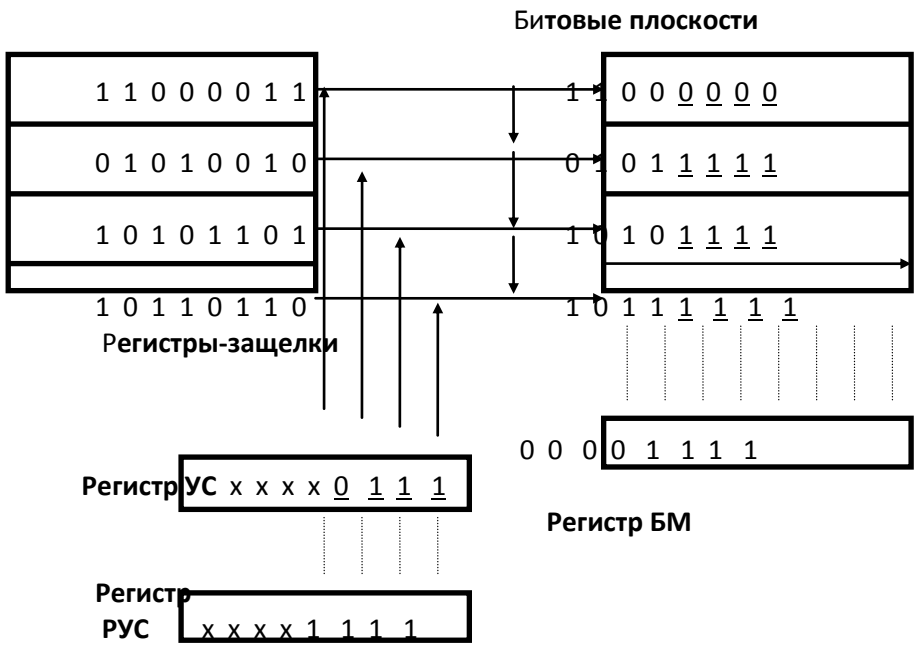


Рис. 4.15

При этом содержимое регистра процессора не имеет значения. В том случае, когда надо изменить только один пиксель, в регистре БМ соответствующий бит устанавливается в 1, а остальные - в 0, и выполняется операция записи в видеопамять. Если в регистре БМ все разряды устанавливаются в 1, то за одну операцию записи в видеопамять будут изменены все восемь пикселей, т.е. прочерчена горизонтальная линия длиной восемь пикселей.

При значении регистра РУС = 0 циклический сдвиг и логическая операция выполняются над содержимым регистра процессора (рис. 4.16).

В соответствии с содержимым регистра ЦСД над байтом регистра процессора выполняется циклический сдвиг, в результате которого получается значение 11000101B. Так как регистр БМ имеет значение 00001111B, то старшая половина байта каждого регистра-защелки пропускается в видеопамять без изменения, а вторая половина получается в результате выполнения логической операции

AND над младшими четырьмя разрядами регистра процессора после сдвига и регистров-защелок.

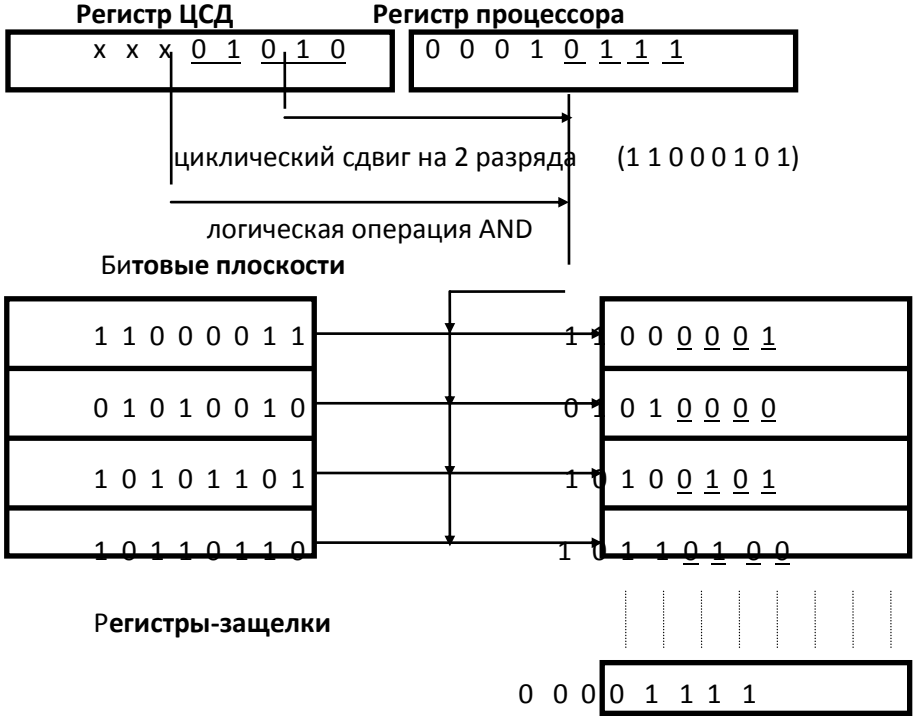


Рис. 4.16

Результат заносится в видеопамять (битовые плоскости). При нулевых значениях 3 и 4 разрядов регистра ЦСД была бы выполнена операция замещения, и в младшие четыре разряда всех битовых плоскостей было бы занесено значение 0101B. В случае нулевого значения и разрядов 0 - 2 регистра ЦСД в младшие разряды всех битовых плоскостей было бы записано значение 0111B.

Запишем процедуру режима **“Запись 0”** при значении регистра РУС = 1111B.

```
Procedure PutPixel0 (X,Y: integer; Pixel: word);
{используется регистр УС, значение РУС = 1111B}
var
```

```
        R:   registers;
        Off: word;
    begin
        Off := 80 • Y + X shr 3;            {смещение}
        {установка регистра БМ}
        R.AL := 8;
        R.AH := 1 shl ((X and 7) xor 7);  {AH = битовая маска}
        PortW[$3CE] := R.AX;
    {установка регистра УС}
        R.AL := 0;
        R.AH := Pixel;                      {AH = значение пикселя}
        PortW[$3CE] := $R.AX;
    {установка регистра РУС}
        PortW[$3CE] := $0F01;    {AH = 0F - значение, AL = 01 - индекс
                                  регистра РУС}
    {загрузка регистров-защелок, их изменение и запись в
видеопамять}
    {значение регистра AL не влияет на результат}
        Mem[$A000:Off] := Mem[$A000:Off] or R.AL;
    {восстановление значений регистров по умолчанию}
        PortW[$3CE] := $FF08;            {регистр БМ}
        PortW[$3CE] := $0001;            {регистр РУС}
    end;
```

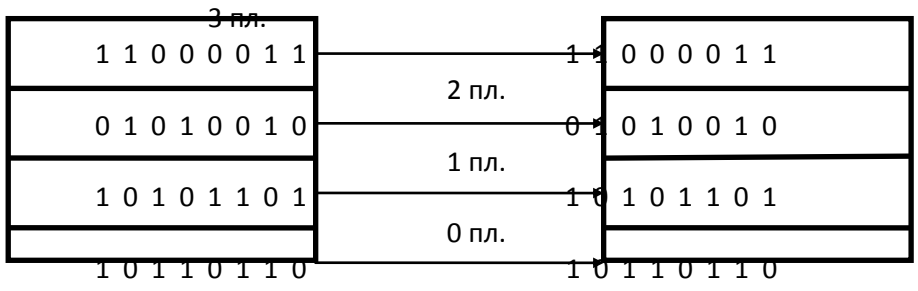


Рис. 4.17

Режим записи 1. В этом режиме выполняется простое копирование содержимого регистров-защелок в видеопамять. В реализации режима “Запись 1” не участвуют никакие регистры,

кроме регистров-защелок. Данный режим используется в случае копирования одной области изображения в другую. При этом регистры-защелки загружаются по исходному адресу и копируются по адресу-приемнику (рис. 4.17).

Режим записи 2. Этот режим похож на режим “Запись 0” при значении РУС = 1111B. В отличие от указанного режима роль регистра УС выполняют четыре младших разряда регистра процессора, а регистр РУС не влияет на результат. В остальном все операции выполняются так же: битовые плоскости изменяются за счет комбинации значений пикселей в регистрах-защелках со значением регистра процессора (попиксельная комбинация) и выполняется логическая операция, установленная в регистре ЦСД, а регистр БМ определяет, какие пиксели изменяются, а какие - нет (рис. 4.18).

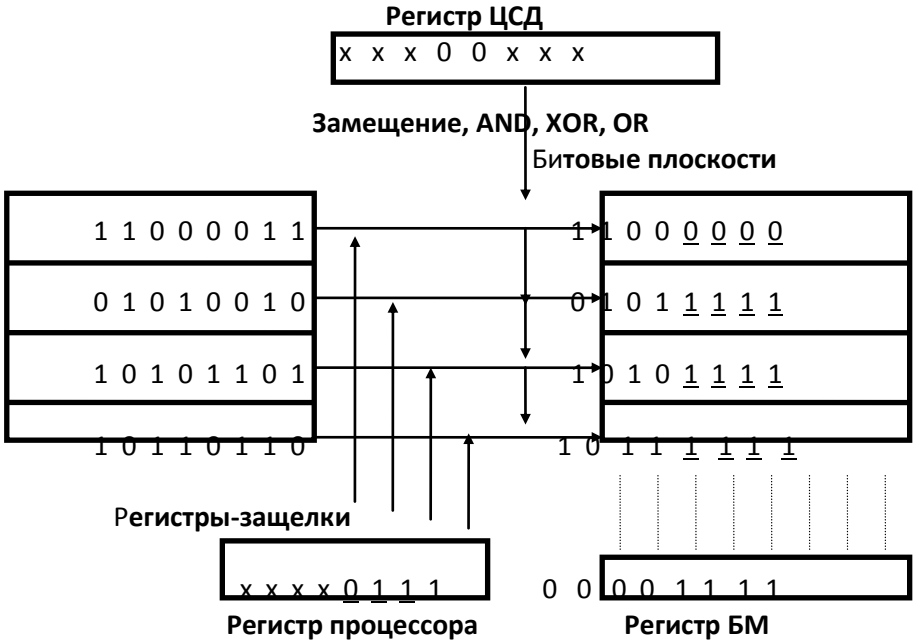


Рис. 4.18

Приведем процедуру реализации режима **“Запись 2”.**

```
Procedure PutPixel2 (X,Y: integer; Pixel: word);
var
```

```
        R:  registers;
        Off: word;
    begin
        Off := 80 • Y + X shr 3;            {смещение}
        {установка регистра БМ}
        R.AL := 8;
        R.AH := 1 shl ((X and 7) xor 7);  {AH = битовая маска}
        PortW[$3CE] := R.AX;
    {установка регистра РЧЗ}
        PortW[$3CE] := $0205;           {режим - 2, индекс регистра - 5}
    {загрузка регистров-защелок байтами из видеопамяти}
        R.AL := Mem[$A000:Off];
    {запись значения пикселя в видеопамять}
                Mem[$A000:Off] := Pixel;
    {восстановление значений регистров по умолчанию}
        PortW[$3CE] := $FF08;           {регистр БМ}
        PortW[$3CE] := $0005;            {регистр РЧЗ}
    end;
```

Режим записи 3. Данный режим поддерживается только видеоадаптером VGA и фактически представляет собой режим “Запись 0” при значении РУС = 1111B с одним отличием - байт данных для регистра процессора циклически сдвигается вправо на число разрядов, заданных в регистре ЦСД, и комбинируется логической операцией AND значением в регистре БМ. В результате получается битовая маска, которая играет роль регистра БМ (рис. 4.19).

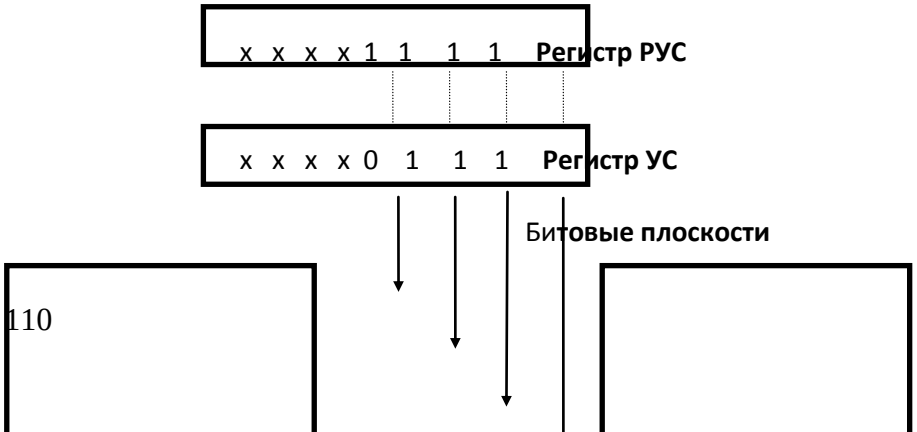
Режим записи с использованием регистра маскирования плоскостей. Во всех трех режимах записи, рассмотренных выше, можно использовать регистр “Маскирование плоскостей” (МП) из блока синхронизации, имеющий индекс 2 и используемый через порты 3C4h и 3C5h. Биты 0 - 3 этого регистра разрешают-запрещают запись в битовые плоскости: установка любого бита в 1 - разрешает запись, в 0 - запрещает. Рассмотрим работу адаптера в этом режиме.

Вначале работы информация из видеопамяти считывается в регистры-защелки. Затем устанавливается нужный бит в регистре “Битовая маска”, и обнуляются все биты пикселя. После этого устанавливаются биты в регистре МП, записывая в него значение пикселя. В регистры-защелки записывается значение FFh. При этом из

этого значения единицы запишутся в те регистры-защелки, которые разрешены для записи, а в остальных сохранятся нули, полученные после обнуления. В завершение работы содержимое регистров-защелок пересылается обратно в видеопамять.

Ниже приведена процедура реализации этого режима.

```
Procedure PutPixel00 (X,Y: integer; Pixel: word);
{используется регистр МП, значение РУС = 0000B}
var
      R:  registers;
      Off: word;
begin
      Off := 80 • Y + X shr 3;          {смещение}
            {установка регистра БМ}
      R.AL := 8;
      R.AH := 1 shl ((X and 7) xor 7);  {AH = битовая маска}
      PortW[$3CE] := R.AX;
{загрузка регистров-защелок и обнуление всех битов пикселя}
      Mem[$A000:Off] := Mem[$A000:Off] and 0;
{установка регистра МП}
      R.AH := byte(Pixel);
      R.AL := 2;                        {AH = значение пикселя}
      PortW[$3C4] := $R.AX;
{установка битов в разрешенные плоскости и запись в видеопамять}
      Mem[$A000:Off] := $FF;
{восстановление значений регистров по умолчанию}
      PortW[$3CE] := $FF08;          {регистр БМ}
      PortW[$3C4] :=  $0F02;          {регистр МП}
end;
```



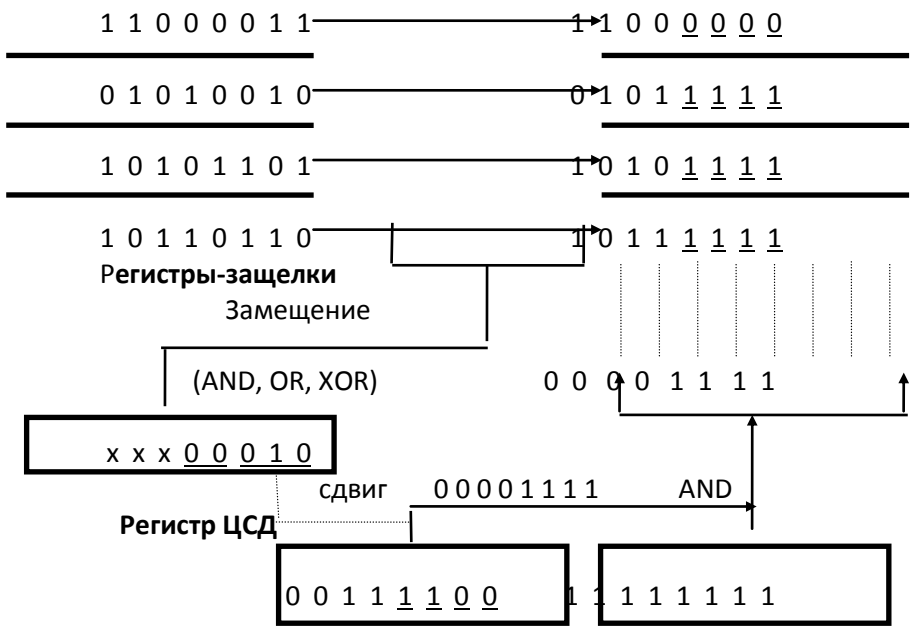


Рис. 4.19

Некоторые замечания по чтению-записи. Для всех режимов записи сохраняется общая схема операций, которая выглядит следующим образом:

- вычисляется адрес видеопамяти, по которому находится пиксель;
- устанавливаются значения регистров, участвующих в данном режиме;
- загружаются регистры-защелки из видеопамяти;
- устанавливается значение пикселя в регистрах-защелках;
- содержимое регистров-защелок пересылается в видеопамять.

Последние три операции (загрузка регистров-защелок, установка пикселя и пересылка в видеопамять) фактически выполняются одной командой процессора. Используя операцию XOR с помощью регистра ЦСД, можно путем двух последовательных записей восстановить исходное изображение, т.е. записать что-то

поверх изображения, а повторной записью операции XOR восстановить исходное изображение.

Представляет интерес анализ статистических данных по быстродействию процедур чтения-записи ассемблерных, паскалевских и стандартных. Пусть быстродействие стандартных процедур PutPixel и GetPixel из библиотеки Graph.tpu принято за 1. Тогда соотношение быстродействия всех процедур чтения-записи по статистике будет следующим:

PutPixel : PutPixel1 : PutPixel0 : PutPixel2 : PutPixelA1 : PutPixelA0 : PutPixelA2 = 1 : 1.42 : 1.46 : 1.55 : 2.32 : 2.32 : 2.55;

GetPixel1 : GetPixel0 : GetPixel : GetPixelA1 : GetPixelA0 =

= 0.87 : 0.93 : 1 : 1.77 : 2.12.

Как видно, для режимов записи даже паскалевские программы с прямым доступом к видеопамяти работают быстрее стандартной процедуры PutPixel, а ассемблеровские - значительно быстрее. Для режимов чтения паскалевские программы медленнее стандартной процедуры GetPixel, а ассемблеровские - быстрее.

Расчеты адресов для SVGA выполняются аналогично. Следует учитывать, что для всех режимов выполняется строчная развертка, а адрес видеопамяти начинается с A000:0000h. Кроме этого, надо знать структуру видеопамяти в данном режиме, которая определяет расположение бит в видеопамяти, отображающих один пиксель. При этом биты могут распределяться по цветовым слоям (4, 8 или 16 слоев) или же располагаться в байтах, последовательно расположенных (1, 2, 3 или 4 байта). И, наконец, следует учитывать, что окно вывода может иметь смещение относительно начального адреса видеопамяти.

4.7. Функции BIOS по прерыванию 10h

Набор программ BIOS (Basic Input/Output System базовая система ввода/вывода) записан на микросхеме ПЗУ BIOS во всех компьютерах, совместимых с IBM PC/XT/AT и PS/2. Он предназначен для управления различными подсистемами компьютера: дисковой, асинхронными последовательными и параллельными портами ввода/вывода, а также видеоадаптером. Микросхема ПЗУ BIOS, расположенная на системной плате, содержит функции для управления видеоадаптерами. Однако почти все видеоадаптеры

имеют собственную микросхему ПЗУ BIOS, расположенную на плате адаптера. На практике для управления видеоадаптером используются функции, записанные в его ПЗУ BIOS.

Использование функций BIOS для управления видеоадаптерами имеет как преимущества, так и недостатки. *Основным преимуществом* функций BIOS является то, что они скрывают всю кропотливую работу по программированию регистров видеоадаптера и видеопамяти, предоставляя программисту достаточно **простой интерфейс**. Кроме этого, разработка функций BIOS, выполняемая самими разработчиками видеоадаптера, позволяет **учитывать все особенности конкретной** модели адаптера. Особенно это проявляется при использовании видеоадаптеров SVGA, у которых наборы регистров и организация видеопамяти разных моделей сильно разнятся.

К недостаткам функций BIOS относятся:

- невысокая скорость их работы; это связано с более медленным доступом к данным, записанным в ПЗУ, чем к оперативной памяти. Многие системные платы позволяют перенести содержимое медленного ПЗУ BIOS в более быструю оперативную память. Эта область оперативной памяти получила название **теневой памяти,** так как ее содержимое полностью повторяет соответствующую область ПЗУ. Подключение теневой памяти выполняется с помощью программы BIOS Setup. ПЗУ BIOS видеоадаптера обычно занимает адресное пространство с адреса C000:0000h до C000:7FFFh;
- невозможность вызова функции BIOS до завершения вызова предыдущей. Это обстоятельство не позволяет пользоваться многими функциями BIOS в резидентных программах;
- отсутствие контроля результата выполнения функций BIOS и сообщений о происшедших ошибках. Любая неточность в исходном тексте программы приводит к ошибке, причину которой трудно выявить;
- возможность работы с функциями BIOS только в реальном режиме процессора.

Доступ к функциям BIOS видеоадаптера выполняется с помощью прерывания INT10h. Для этого в регистр AH загружается номер функции BIOS видеоадаптера, которую предполагается исполнить. Затем загружаются остальные регистры процессора в соответствии с вызываемой функцией, и выполняется прерывание INT 10h.

В качестве примера рассмотрим использование функции 00h.

Выбор режима работы - функция 00h. Функция 00h прерывания 10h позволяет задать любой стандартный режим работы видеоадаптера. При описании функций в качестве входной информации указываются устанавливаемые регистры и их содержимое, а в качестве выходной информации - формируемые регистры в результате выполнения описываемой функции.

На входе: AH 00h

AL Номер устанавливаемого режима работы видеоадаптера, если бит D7 = 1, то при установке режима видеопамять не очищается

На выходе: Не используется

4.8. Стандарт VESA

Этот стандарт описывает расширение прерывания BIOS INT 10h (VESA BIOS Extention - VBE), отвечающего за управление видеоадаптерами. Поддержка VBE обычно включается производителями видеоадаптеров в ПЗУ самого адаптера или поставляется в виде отдельной резидентной программы. Во втором случае перед использованием функций VBE необходимо загрузить эту резидентную программу в оперативную память компьютера. Ниже приводятся краткое описание функций VBE версии 1.2 и некоторых функций версии 2.0.

Перед вызовом функций VBE следует записать в регистр AH значение 1Fh. Если данная реализация VBE поддерживает требуемую функцию, то в регистре AL возвращается значение 4Fh. В противном случае в этом регистре будет иное значение. Результат выполнения функции записывается в регистр AH. В случае успешного завершения функции в регистр AH возвращается нулевое значение. При возврате значения 1h функция завершилась с ошибкой. И, наконец, если в регистре возвращается значение 2h, то аппаратура видеоадаптера не

поддерживает данную функцию. Кроме этого, возможна ситуация, при которой VBE может выполнить запрошенную функцию, а аппаратура видеоадаптера - нет. В этом случае после завершения функции регистр AH содержит значение 4Fh, а регистр AH - 2h.

Для примера ниже рассматривается функции 4F00h стандарта VESA.

Функция “Получить информацию о реализации VBE и видеоадаптере” (4F00h). Функция позволяет получить информацию о возможностях VBE и видеоадаптере. Для VBE версии 1.2 (и боле ранних версий):

На входе: AH 4Fh
AL 00h
ES:DI Указатель на буфер размером 256 байт. В этот буфер записывается различная информация о видеоадаптере SVGA и реализации VBE

На выходе: AL 4Fh
AH 0 - в случае успешного завершения, 1 - в случае ошибки

Значения остальных регистров сохраняются.

Для VBE версии 2.0 указатель определяет буфер в 512 байт, первые 4 байта которого должны содержать строку “VBE2”. Форматы этих буферов имеют как совпадения, так и различия.

Проектное задание

Провести анализ алгоритмов растровой графики с точки зрения выполняемых операций, их сложности и эффективности.

ТЕСТ РУБЕЖНОГО КОНТРОЛЯ №1

Тест содержит 10 заданий, на выполнение которых отводится 5 минут. Для ответа нужно выбрать наиболее правильный, на Ваш взгляд, вариант ответа, сделав соответствующую пометку в бланке ответов. Весовой коэффициент каждого вопроса составляет 0,1.

1. Укажите наиболее эффективный алгоритм растровой развертки отрезка			
1)	цифровой дифференциальный анализатор	2)	по уравнению прямой
3)	алгоритм Брезенхема	4)	использование коэффициента наклона прямой
2. Укажите наиболее эффективный алгоритм растровой развертки окружности			
1)	вписанный многоугольник	2)	алгоритм Брезенхема
3)	по уравнению окружности	4)	в полярных координатах
3. За счет чего достигается эффективность алгоритмов растровой развертки отрезка и окружности?			
1)	использование вещественных чисел двойной точности	2)	использование целых чисел
3)	минимальная погрешность	4)	использование простейших операций
4. Укажите наиболее эффективный алгоритм затравочного заполнения сплошных областей			
1)	затравочный с поточечным заполнением	2)	затравочный интервальный
3)	закраска перед помещением точки в стек	4)	закраска после извлечения точки из стека
5. На каких принципах строится растровая развертка многоугольника?			
1)	на поточечном заполнении области многоугольника	2)	на интервальном заполнении области многоугольника

3)	на закраске точек, окружающих текущую	4)	на блочном заполнении области многоугольника
6. Алгоритм растровой развертки круга основывается на заполнении области интервалами, порожденными текущей точкой окружности в первом октанте. Какая проблема может возникать в этом случае с интервалами, расположенными во 2-3 и 6-7 октантах?			
1)	неточное заполнение круга	2)	дублирование работы предыдущего шага
3)	недопустимая погрешность	4)	увеличение объема вычислений
7. Укажите элементарные операции преобразования точки на плоскости			
1)	перенос, масштабирование или поворот относительно начала координат	2)	перенос, масштабирование или поворот относительно произвольной точки плоскости
3)	композиция преобразований переноса, масштабирования или поворота относительно начала координат	4)	композиция преобразований переноса, масштабирования или поворота относительно произвольной точки плоскости
8. Какой размер имеет вектор точки на плоскости в однородных координатах?			
1)	2	2)	3
3)	4	4)	1
9. Какому условию должен отвечать коэффициент W в однородных координатах?			
1)	равен 1	2)	не равен 0
3)	произвольной число в интервале от 0 до 1	4)	больше 0
10. Если определитель третьего порядка |P P1 P2| = 0, то			
1)	точка P расположена справа от прямой,	2)	точка P расположена слева от прямой, проходящей

	проходящей через точки P_1 и P_2		через точки P_1 и P_2
3)	точка Р расположена на прямой, проходящей через точки P_1 и P_2	4)	точка Р расположена на прямой, проходящей через точки P_1 и P_2, но за пределами отрезка P_1 P_2.

Бланк ответов

№	1	2	3	4	5	6	7	8	9	10
1)										
2)										
3)										
4)										

Другие виды тестирования по модулю

1. ***Выполнение лабораторных работ***
   - лабораторная работа «Построение динамического изображения»
   - лабораторная работа «Алгоритмы растровой графики»
2. ***Подготовка курсовой работы***
   - определение темы курсовой работы
   - подбор литературных источников
   - изучение теоретического материала
3. ***Выполнение двух контрольных работ***
   - стартовая контрольная работа
   - контрольная работа по алгоритмам растровой графики

ВВЕДЕНИЕ

Учебное пособие предназначено для изучения основ работы с трехмерной графикой. С этой целью излагаются математические преобразования элементов пространства и алгоритмы получения реалистического изображения трехмерных сцен на плоском экране дисплея.

В главе 1 даются представления однородных координат, используемых в работе с такими графическими примитивами как точка, прямая, плоскость. Рассматриваются элементарные преобразования точки в пространстве в виде переноса, масштабирования и поворота, а также более сложные преобразования в виде их композиции. Даются основы построения проекций объектов для случаев центрального и параллельного проектирования.

Глава 2 посвящена задаче трехмерного отсечения. При этом приводятся формы отсекающих объемов, условия полной видимости и невидимости отрезков относительно отсекающего объема, методика выявления невыпуклых объемов и разбиения их на выпуклые части. Алгоритмы трехмерного отсечения базируются на методах отсечения для плоской задачи.

Алгоритмы удаления невидимых линий и поверхностей предлагаются в главе 3. Рассматриваются алгоритмы для различных трехмерных сцен, состоящих из плоских граней, криволинейных поверхностей, описываемых математическими зависимостями, а также объёмных тел, ограниченных плоскими гранями.

В заключительной, четвертой, главе предлагаются модели освещенности и алгоритмы закраски плоских поверхностей по методу Гуро и Фонга.

В учебном пособии даются вопросы тестового контроля. При изложении алгоритма Робертса приводятся многочисленные примеры, иллюстрирующие его работу.

ОБЩИЕ ПОЛОЖЕНИЯ

Трехмерная компьютерная графика решает задачи представления трехмерных тел на плоском экране дисплея. Для этого необходимо построить проекцию трехмерной сцены и сделать ее реалистической, вычислив освещенность её поверхностей и тени от объектов. Наложение текстуры на изображение позволяет отобразить материалы, из которых состоят объекты и качество обработки их поверхностей.

При построении реалистического изображения нужно учитывать как физические, так и психологические особенности восприятия окружающей среды. К физическим особенностям относятся свойства человеческого глаза по приему и обработке светового потока. Рецепторы глаза в виде колбочек и палочек имеют разную чувствительность к большим и малым уровням освещенности, что влияет на восприятие среды днем и в сумерках. Кроме того при низких уровнях освещенности цветные предметы кажутся черно-белыми.

Чувствительность глаза к яркости света достаточно высокая – в пределах, порядка, 10^{10}. Однако одновременно он может воспринимать существенно меньший диапазон перепада яркости. Поэтому глаз приспосабливается к средней яркости сцены. При этом область с некоторой освещенностью воспринимается по-разному в зависимости от фона, на котором она наблюдается.

При отображении объектов на экране следует учитывать и законы освещенности. Учитывая конечные размеры дисплея, его разрешающую способность и цветовые характеристики, приходится вносить коррективы в расчеты освещенности по законам оптики, т.к. прямое вычисление по ним дает изображение, которое отличается от обычного восприятия. Это проявляется особенно, когда объекты сцены сильно различаются по размерам и расстояниям между ними. Мелкие предметы трудно изобразить на проекции, а большие могут заслонять большую часть сцены.

Указанные выше физические и психологические особенности диктуют свои законы в реалистических представлениях трехмерных сцен.

Модуль № 2

ОСНОВЫ ТРЕХМЕРНОЙ ГРАФИКИ

Комплексная цель

Изложить основные математические и алгоритмические методы создания растровых изображений трехмерной объектов на экране дисплея; научить различным подходам в решении задачи удаления невидимых линий и поверхностей; дать представление реалистического изображения и алгоритмы расчета освещенности и закраски поверхностей по методам Гуро и Фонга.

Содержание модуля

ГЕОМЕТРИЯ ПРОСТРАНСТВА

1.1. Трехмерные однородные координаты и преобразования точки в пространстве

Преобразования точки в пространстве выполняются, также как и на плоскости, в однородных координатах. В этом случае точка **P** представляется четырехмерным вектором (X Y Z W), где W – произвольный множитель, не равный нулю. Связь декартовых и однородных координат выражается следующими соотношениями:

$$x = \frac{X}{W}; \quad y = \frac{Y}{W}; \quad z = \frac{Z}{W};$$

Из этого определения следует, что две точки (X_1,Y_1,Z_1) и $\mathbf{P_2}(X_2,Y_2,Z_2)$ представляют собой одну и ту же точку, если $\mathbf{P_1} = c\mathbf{P_2}$ для любого $c \neq 0$. Использование однородных координат позволяют вычислять преобразования композиции элементарных преобразований путем умножения на результирующую матрицу, полученную перемножением матриц соответствующей последовательности преобразований. Матрицы трехмерных однородных координат имеют порядок 4 x 4.

Перенос. Координаты точки после переноса можно вычислить в векторной форме

$$\mathbf{P}^* = \mathbf{P} \cdot \mathbf{T},$$

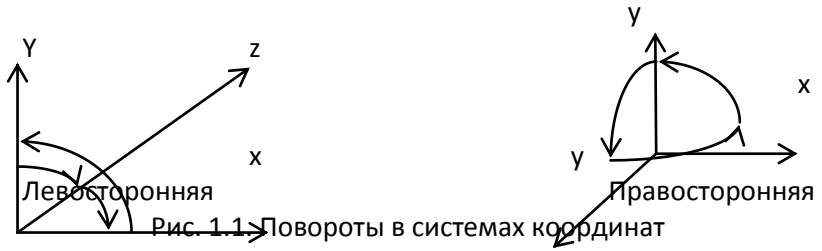
где $T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ D_x & D_y & D_z & 1 \end{bmatrix}$.

Масштабирование. Аналогично вычисляется координаты преобразованной точки при масштабировании

$$\mathbf{P}^* = \mathbf{P} \cdot \mathbf{S},$$

где $S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

Поворот. В отличие от плоскости поворот в пространстве возможен вокруг каждой из трех осей – X, Y, Z. Можно рассмотреть, кроме этого, повороты в левосторонней и правосторонней системах координат.



Левосторонняя Правосторонняя

Рис. 1.1. Повороты в системах координат

Стрелками показано направление положительного вращения вокруг соответствующих осей. Для определения направления положительного вращения удобно использовать простую табличку следующего вида.

Ось вращения	**Направление положительного вращения**
X	$Y \rightarrow Z$
Y	$Z \rightarrow X$
Z	$X \rightarrow Y$

Для правосторонней системы координат положительное направление – это вращение против часовой стрелки при взгляде с конца положительной полуоси вращения, а для левосторонней – по

часовой стрелке. В компьютерной графике обычно используют обе системы координат. Расположение их осей выбирают таким образом, чтобы направление оси Z совпадало с направлением взгляда наблюдателя.

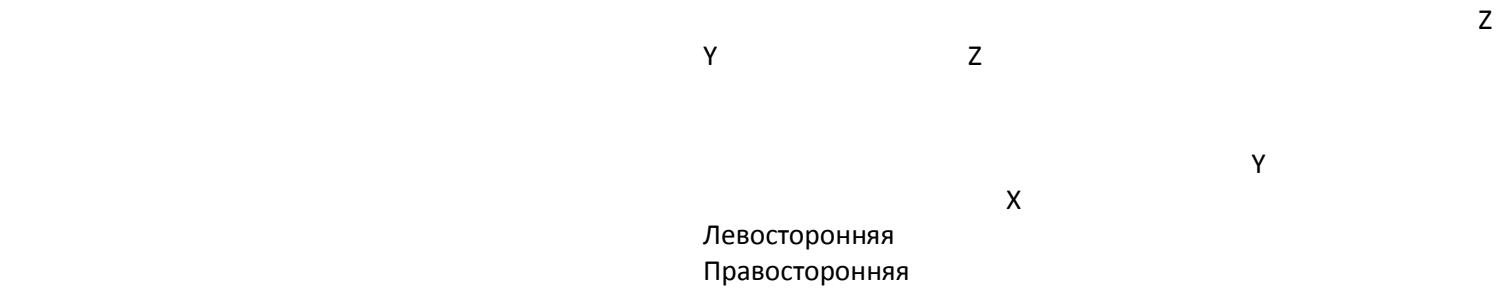


Рис. 1.2. Ориентация координат на экране

Матрицы поворота вокруг осей X, Y, Z имеют следующие значения.

$$R_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \mathrm{Cos}(\varphi) & \mathrm{Sin}(\varphi) & 0 \\ 0 & -\mathrm{Sin}(\varphi) & \mathrm{Cos}(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

Вокруг оси Y:

$$R_y(\varphi) = \begin{bmatrix} \mathrm{Cos}(\varphi) & 0 & -\mathrm{Sin}(\varphi) & 0 \\ 0 & 1 & 0 & 0 \\ \mathrm{Sin}(\varphi) & 0 & \mathrm{Cos}(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

Вокруг оси Z:

$$R_z(\varphi) = \begin{bmatrix} \mathrm{Cos}(\varphi) & \mathrm{Sin}(\varphi) & 0 & 0 \\ -\mathrm{Sin}(\varphi) & \mathrm{Cos}(\varphi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Тогда координаты точки после поворота будут вычисляться перемножением вектора точки на соответствующую матрицу поворота

$$\mathbf{P}^* = \mathbf{P} \cdot \mathbf{R}(\varphi).$$

Например, поворот единичного вектора оси X на угол 90^0 вокруг оси Z можно вычислить следующим образом.

$$\begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix}.$$

После поворота получили единичный вектор оси Y.

Обратные матрицы. Все матрицы преобразований M точки в пространстве имеют обратные матрицы M^{-1}. Для переноса – это матрица преобразования для переноса точки в обратном направлении

$$T^{-1} = T(-D_x, -D_y, -D_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -D_x & -D_y & -D_z & 1 \end{bmatrix}.$$

Аналогично можно получить обратную матрицу для масштабирования точки.

$$S^{-1} = S(1/S_x, 1/S_y, 1/S_z) = \begin{bmatrix} 1/S_x & 0 & 0 & 0 \\ 0 & 1/S_y & 0 & 0 \\ 0 & 0 & 1/S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Обратная матрица поворота для каждой оси получается поворотом точки в обратном направлении. Например, для поворота вокруг оси Z, матрица имеет следующий вид.

$$R_z^{-1} = R_z(-\varphi) = \begin{bmatrix} \mathrm{Cos}(\varphi) & -\mathrm{Sin}(\varphi) & 0 & 0 \\ \mathrm{Sin}(\varphi) & \mathrm{Cos}(\varphi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Аналогично получаются обратные матрицы для поворота вокруг других осей. Следует отметить, что матрицы поворотов содержат подматрицы размером 3х3, строки или столбцы которой являются взаимно-ортогональными единичными векторами. Такая матрица называется ортогональной. В этом случае обратная матрица может быть получена транспонированием исходной матрицы. Это свойство можно использовать, так как прямое вычисление обратной матрицы занимает значительной бóльший объем вычислений.

Композиции элементарных преобразований. Последовательность поворотов вокруг произвольных осей координат дает результирующую матрицу вида

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Здесь подматрица поворотов 3х3 также является ортогональной. При повороте, задаваемым этой матрицей, единичные вектора совпадают с осями x, y, z. Матрицы поворота сохраняют размеры объекта и его углы.

Композиция любых элементарных преобразований дает матрицу следующего вида.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}.$$

Прямое вычисление преобразования точки по этой матрице требует 16 операций умножения и 12 операций сложения. Однако, учитывая наличие в матрице столбца из нулей и единицы, этот объем можно уменьшить вычислением координат по следующим выражениям.

$$x^* = x \cdot r_{11} + y \cdot r_{21} + z \cdot r_{31} + t_x;$$
$$x^* = x \cdot r_{12} + y \cdot r_{22} + z \cdot r_{32} + t_y;$$
$$x^* = x \cdot r_{13} + y \cdot r_{23} + z \cdot r_{33} + t_z.$$

1.2. Уравнение прямой и плоскости в пространстве

Каноническое уравнение прямой в пространстве, проходящей через две точки P1 (x1,y1,z1) и P2(x2,y2,z2), имеет вид

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1} = \frac{z - z_1}{z_2 - z_1}.$$

Часто в компьютерной графике используют запись уравнения прямой в параметрическом виде:

$$\mathbf{P}(t) = \mathbf{P}_1 + (\mathbf{P}_2 - \mathbf{P}_1)\cdot t,$$

где параметр t лежит в интервале от 0 до 1. В случае, когда t < 0, точка находится на прямой перед точкой P1, а при t > 1 она лежит также на прямой, но после точки P2.

Плоскость в пространстве задается каноническим уравнением

$$Ax + Dy + Cz + D = 0.$$

Любая плоскость однозначно определяется вектором нормали n и произвольной точкой на плоскости P(x,y,z). В этом случае она задается скалярным произведением вектора нормали n и вектора точки P:

$$\mathbf{n} \cdot \mathbf{P} = 0.$$

Если в левую часть канонического уравнения плоскости подставить координаты некоторой точки P(x,y,z) пространства, то значение выражения может оказаться равным или не равным нулю. В случае равенства нулю, точка принадлежит плоскости, а при неравенстве – она расположена по одну или другую сторону от плоскости (в положительном или отрицательном полупространстве). Для точки, лежащей в плоскости определитель, составленный для трех точек этой плоскости, будет равен нулю.

$$\begin{vmatrix} x & x_1 & x_2 & x_3 \\ y & y_1 & y_2 & y_3 \\ z & z_1 & z_2 & z_3 \\ w & w_1 & w_2 & w_3 \end{vmatrix} = 0.$$

Аналогичный определитель для трех плоскостей пространства равен нулю

$$\begin{vmatrix} a & a_1 & a_2 & a_3 \\ b & b_1 & b_2 & b_3 \\ c & c_1 & c_2 & c_3 \\ d & d_1 & d_2 & d_3 \end{vmatrix} = 0.$$

Если раскрыть определитель, то коэффициенты при переменных a, b, c, d будут определять значения координат точки пересечения трех плоскостей (x, y, z, w).

Пусть в плоскости задан вектор (P – P1), тогда скалярное произведение этого вектора на вектор нормали, как уже говорилось, будет задавать плоскость.

$$\mathbf{n} \cdot (\mathbf{P} - \mathbf{P}_1) = 0.$$

Вектор нормали можно заменить векторным произведением двух векторов, лежащих в данной плоскости. Тогда

$$((\mathbf{P}_2 - \mathbf{P}_1) \times (\mathbf{P}_3 - \mathbf{P}_1)) \cdot (\mathbf{P} - \mathbf{P}_1) = 0.$$

Векторное произведение двух векторов P1 и P2: (P1 x P2) – представляет собой вектор с компонентами ($y_1z_2 – y_2z_1$ $z_1x_2 – z_2x_1$ $x_1y_2 - x_2y_1$).

Точка пересечения прямой с плоскостью может быть определена с помощью скалярного произведения векторов

$$\mathbf{n} \cdot (\mathbf{P}(t) - \mathbf{A}) = 0.$$

Здесь P(t) – точка пересечения прямой с плоскостью, а вектор A – вектор точки плоскости: A = ($x_Ay_Az_A$). Точка пересечения задана в параметрической форме. Если выражение в правой части представить некоторой функцией f(P), то точка A, лежащая в плоскости дает значение f(A) = 0, а для остальных случаев - $f(A) \neq 0$. Причем, когда она находится в положительном полупространстве, то f(A) > 0, а в отрицательном - f(A) < 0. Это обстоятельство можно использовать для анализа расположения двух точек A и B по отношению к некоторой плоскости. Если $f(A) \cdot f(B) > 0$, то точки A и B лежат по одну сторону от плоскости, а если $f(A) \cdot f(B) < 0$, то – по разные стороны.

1.3. Изображения трехмерных объектов

Проекции. Визуализация двумерных изображений сопровождается указанием окна видимости, выполнением задачи

отсечения для этого окна и выдачей результата на экран в заданную область. Для случая трехмерных изображений задача отсечения выполняется не по окну видимости, а по объему видимости. По результату отсечения затем строится проекция объема видимости на плоскость вывода (картинная плоскость). Полученная проекция может также подвергаться двумерному отсечению с последующим выводом в заданную область экрана. В процессе вывода на экран могут выполняться и некоторые преобразования (перемещения, поворота и масштабирования).

Построение проекций трехмерных изображений представляет собой процесс отображения точек трехмерного пространства в точки плоскости. Проекция объекта складывается из проекций его составляющих. Точка пространства соединяется с точкой проекции при помощи прямых проецирующих лучей – проекторов. Для построения проекции отрезка достаточно получить проекции его концевых точек, а затем выполнить их соединение прямой. Таким образом, получают плоские геометрические проекции.

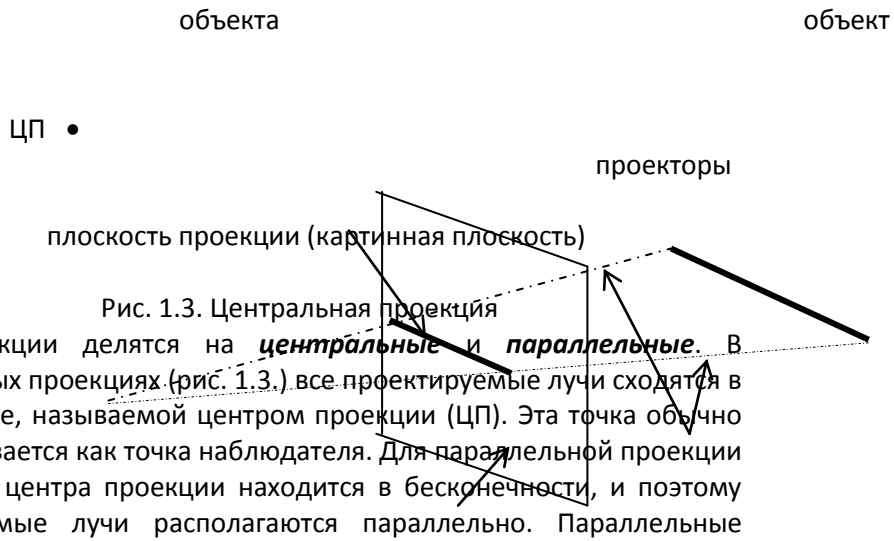
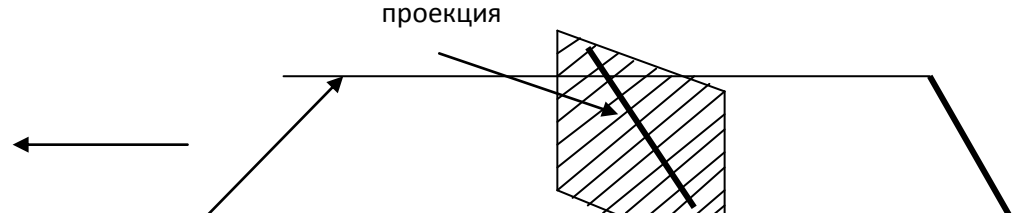


Рис. 1.3. Центральная проекция

Проекции делятся на ***центральные*** и ***параллельные***. В центральных проекциях (рис. 1.3.) все проектируемые лучи сходятся в одной точке, называемой центром проекции (ЦП). Эта точка обычно рассматривается как точка наблюдателя. Для параллельной проекции (ПП) точка центра проекции находится в бесконечности, и поэтому проецируемые лучи располагаются параллельно. Параллельные проекции (рис. 1.4.) дают менее реалистические изображения, однако, используются чаще, особенно в конструировании.



ЦП

проекторы

Рис. 1.4. Параллельная проекция

1.4. Параллельные проекции

Параллельные проекции разделяются на два вида в зависимости от направления нормали плоскости проекции и проекторов. Если эти направления совпадают, то этот вид проектирования называется ***ортогональной*** параллельной проекцией, при несовпадении – ***косоугольной*** ПП.

Ортогональные ПП. При ортогональном проектировании плоскость проекции расположена перпендикулярно линии проекторов. Обычно систему координат располагают таким образом, чтобы проекторы совпадали с осью Z или были ей параллельны, а плоскость проекции совмещалась с плоскостью XOY. В этом случае координата Z не имеет отображения на плоскости проекции и точка **P**(x,y,z) пространства отображается на плоскости проекции точкой $\mathbf{P}^*(x^*,y^*)$. Справедливо соотношение $x = x^*$, $y = y^*$. Построение указанной проекции точек пространства может быть выполнено с помощью матрицы преобразования ортогональной параллельной проекции

$$M_{орт} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Проекция точки получается умножением вектора точки на матрицу преобразования

$$\mathbf{P} = \mathbf{P}^* \cdot M_{орт}.$$

Если направления проекторов не совпадает с направление оси Z, то задачу можно свести к предыдущей преобразованием системы координат, таким что эти направления совпадут. Пусть имеется произвольная точка пространства, понимаемая как точка расположения наблюдателя $\mathbf{P}(x_н,y_н,z_н)$. Необходимо выполнить

поворот осей координат до совмещения вектора наблюдения (проектора) с направлением оси Z.

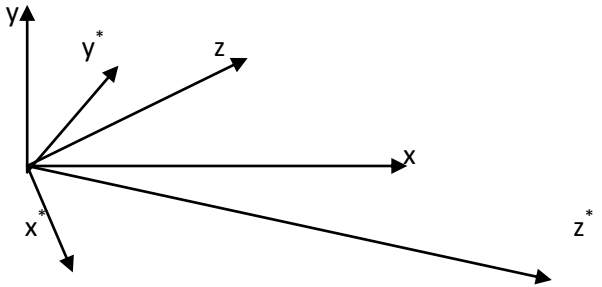


Рис. 1.5. Преобразование системы координат

Такое преобразование можно выполнить последовательными поворотами координат, например, вокруг осей X и Y. Схема этих поворотов показана на рис. 1.6. В результате поворотов ось Z должна совпасть с осью Z^*. То есть необходимо совместить ось Z с вектором, проходящим через точку начала координат и точку расположения наблюдателя.

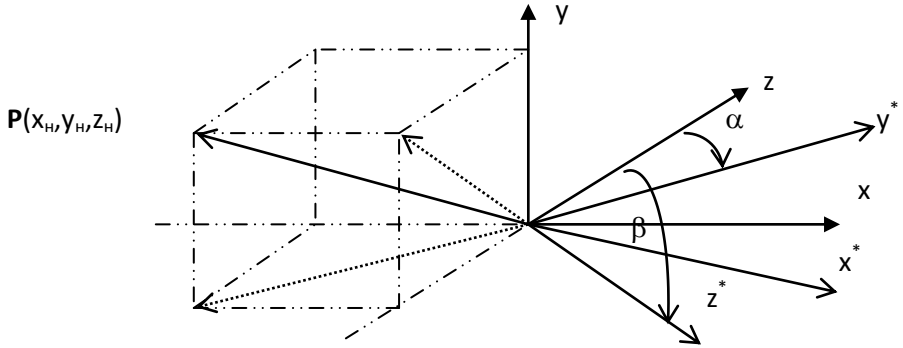


Рис. 1.6. Схема поворотов осей координат

Первый поворот OZ выполняется вокруг оси Y в плоскости XOZ на угол α – угол между осью OZ и продолжением проекции исходного вектора на плоскость XOZ. Можно составить матрицу преобразования для этого поворота. Так как поворот системы координат в положительном направлении приводит к повороту пространства в отрицательном направлении, то необходимо вычислить матрицу $R_y(-\alpha)$.

$$R_y(-\alpha) = \begin{bmatrix} \mathrm{Cos}\alpha & 0 & \mathrm{Sin}\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\mathrm{Sin}\alpha & 0 & \mathrm{Cos}\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Здесь тригонометрические функции можно определить по координатам следующим образом.

$$\mathrm{Cos}\alpha = -\frac{z_н}{r_y}; \quad \mathrm{Sin}\alpha = -\frac{x_н}{r_y}; \quad r_y = \sqrt{x_н^2 + z_н^2}.$$

Второй поворот выполняется вокруг оси X на угол β между осью OZ и продолжением проекции исходного вектора на плоскость ZOY.

$$R_x(-\beta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \mathrm{Cos}\beta & -\mathrm{Sin}\beta & 0 \\ 0 & \mathrm{Sin}\beta & \mathrm{Cos}\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Аналогично вычисляются тригонометрические функции.

$$Cos\beta = -\frac{z_н}{r_x}; \quad Sin\beta = -\frac{y_н}{r_x}; \quad r_x = \sqrt{y_н^2 + z_н^2}.$$

Теперь можно получить результирующую матрицу преобразования пространства.

$$M_{орт}(x_н\ y_н\ z_н) = R_y(-\alpha) \cdot R_x(-\beta) \cdot M_{орт}.$$

Таким образом, $M_{орт}(x_н\ y_н\ z_н)$ представляет собой матрицу преобразования для ортогонального проецирования с наблюдателем в точке $(x_н\ y_н\ z_н)$.

Косоугольное ПП. Рассмотрим случай косоугольного проецирования, при котором направление проецирования совпадает с осью Z, а проекционная плоскость проходит через начало координат (0, 0) и имеет вектор нормали **n**. Поворотом системы координат можно свести эту задачу к эквивалентной, в которой проекционная плоскость совпадает с плоскостью XOY, а направление проецирования совпадает с осью Z^*.

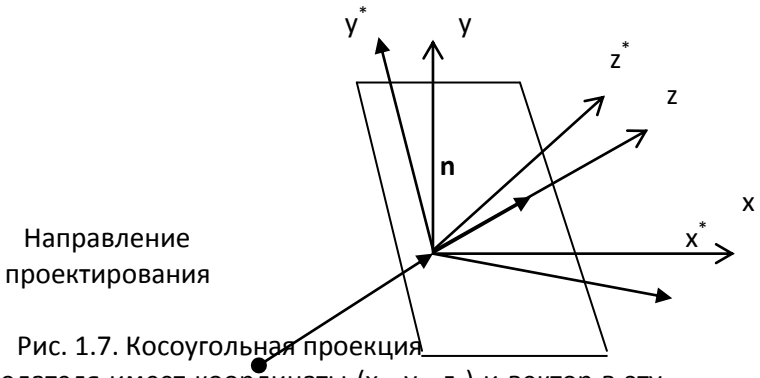
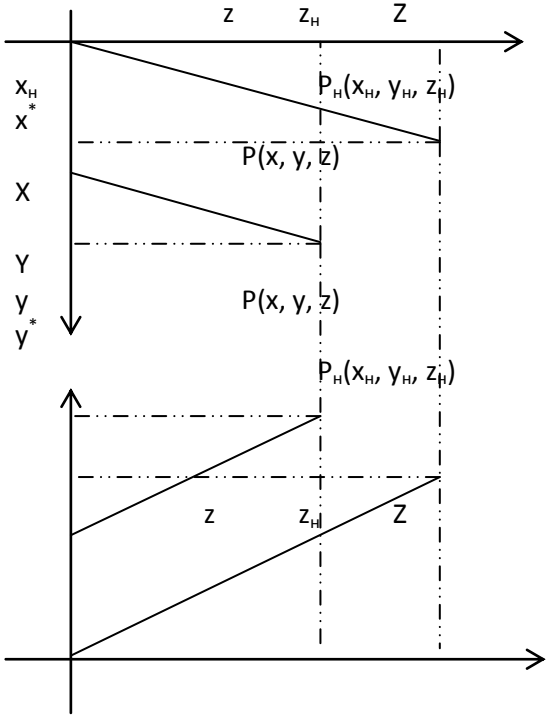


Рис. 1.7. Косоугольная проекция

Точка наблюдателя имеет координаты ($x_н$, $y_н$, $z_н$) и вектор в эту точку может быть получен преобразованием единичного вектора оси Z исходной системы координат в новую систему координат, т.к. направление проецирования теперь совпадает с осью OZ^*.

Проведем вычисление координат проекции (x^*, y^*) произвольной точки пространства (x, y, z). Рассмотрим последовательно проекции точки в плоскостях XOZ и YOZ. На рис. 1.8. проектируемая точка - (x, y, z), а вектор проектирования - ($x_н$, $y_н$, $z_н$).



Обозначим расстояние между проекциями точки на плоскость XOY и соответствующими координатами x и y через a и b, тогда можно записать:

$$x^* = x - a;\ y^* = y - b.$$

Из подобия треугольников следует, что

$$\frac{a}{z} = \frac{x}{z_н};\quad \frac{b}{z} = \frac{y}{z_н}.$$

С учетом этого получим

$$x^* = x - z\frac{x_н}{z_н};\quad y^* = y - z\frac{y_н}{z_н}.$$

Эти преобразования можно получить с помощью следующей матрицы

$$M_{кос}(x_н, y_н, z_н) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{x_н}{z_н} & -\frac{y_н}{z_н} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

С помощью этой матрицы можно получить координаты проекции точки при параллельном косоугольном проектировании следующей операцией

$$P^* = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot M_{кос}(x_н, y_н, z_н) = \begin{bmatrix} x - z \cdot \frac{x_н}{z_н} & y - z \cdot \frac{y_н}{z_н} & 0 & 1 \end{bmatrix}.$$

В более **общем случае** направление проектирования определяется точкой наблюдателя ($x_н$, $y_н$, $z_н$) и проекционной плоскостью, задаваемой вектором нормали **n**. Решение задачи построения проекции точки может быть сведено к предыдущей поворотом системы координат таким образом, чтобы проекционная плоскость совпала с плоскостью XOY. В результате этого преобразования вектор направления проецирования изменит свое положение. Его координаты будут иметь значения ($x^*_н$, $y^*_н$, $z^*_н$), которые можно вычислить следующим образом

$$\left[x^*_н \quad y^*_н \quad z^*_н \quad 1\right] = \left[x_н \quad y_н \quad z_н \quad 1\right] \cdot M(\mathbf{n}).$$

Окончательные вычисления выполняются перемножением полученной вектор-строки на матрицу косоугольного проектирования $M_{кос}(x_н, y_н, z_н)$, т.е.

$$M_{кос}(x^*_н, y^*_н, z^*_н) = = \left[x_н \quad y_н \quad z_н \quad 1\right] \cdot M(\mathbf{n}) \cdot M_{кос}(x_н, y_н, z_н) =$$
$$\left[x^*_н \quad y^*_н \quad z^*_н \quad 1\right] \cdot M_{кос}(x_н, y_н, z_н).$$

1.5. Центральные проекции

При построении центральных проекций обычно центр проецирования располагают на оси Z в точке $z_н$, а проекционная плоскость совпадает с плоскостью XOY.

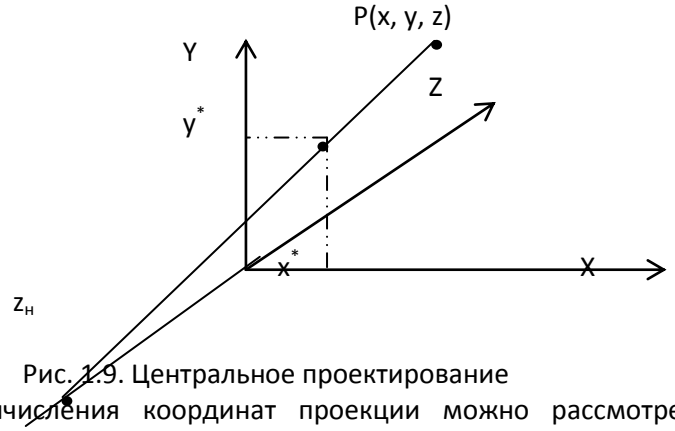


Рис. 1.9. Центральное проектирование

Для вычисления координат проекции можно рассмотреть плоскости YOZ и XOZ.

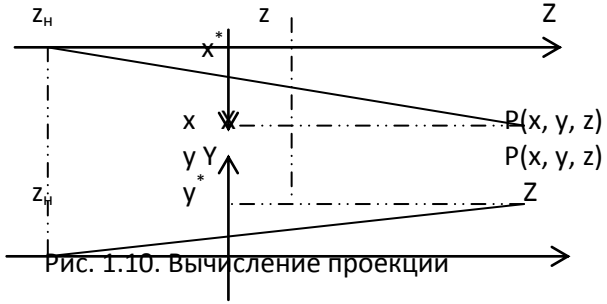


Рис. 1.10. Вычисление проекции

Из подобия треугольников можно записать:

$$\frac{x^*}{-z_н} = \frac{x}{z - z_н}; \frac{y^*}{-z_н} = \frac{y}{z - z_н}.$$

Отсюда получаем значения координат проекции точки.

$$x^* = \frac{x}{1 - \frac{z}{z_н}}; \quad y^* = \frac{y}{1 - \frac{z}{z_н}}.$$

Соответствующая матрица преобразований имеет вид

$$M_{цент}(z_н) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{z_н} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Тогда преобразование точки получается с помощью матрицы следующим образом

$$P^* = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot M_{цент}(z_н) = \begin{bmatrix} x & y & 0 & 1 - \frac{z}{z_н} \end{bmatrix} = \begin{bmatrix} \frac{x}{1 - \frac{z}{z_н}} & \frac{y}{1 - \frac{z}{z_н}} & 0 & 1 \end{bmatrix}.$$

В более общем случае центр проекции может располагаться в произвольной точке пространства ($x_н$, $y_н$, $z_н$) совпадать с плоскостью XOY. Если произвести сдвиг системы координат по оси X на величину $x_н$, а по оси Y – на величину $y_н$ и произвести проектирование точки с последующим возвратом системы координат в исходное состояние, то задача сводится к предыдущей и матрица преобразования может быть получена следующим образом.

$$M_{цент}(x_н, y_н, z_н) = T(-x,-y,0) \cdot M_{центр}(z_н) \cdot T(x_н, y_н, 0) =$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_н & -y_н & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{z_н} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_н & y_н & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{x_н}{z_н} & -\frac{y_н}{z_н} & 1 & -\frac{1}{z_н} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Случай, когда плоскость проектирования не совпадает с плоскостью XOY, является самым случаем центральной проекции. При этом плоскость проектирования задается вектором нормали **n**, а центр проекции – точкой ($x_н$, $y_н$, $z_н$). Матрица преобразования получается умножением предыдущей результирующей матрицы на матрицу (вектор) нормали M(**n**).

$$M_{центр}(x^*_н, y^*_н, z^*_н) = M_{центр}(x_н, y_н, z_н) \cdot M(\mathbf{n}).$$

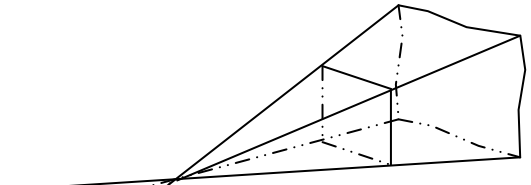
2. ТРЕХМЕРНОЕ ОТСЕЧЕНИЕ

2.1. Формы отсекающих объемов

Как уже говорилось, при формировании трехмерного изображения используется операция трехмерного отсечения. Причем, рассматривают два вида отсекающих объемов по способам проецирования – объемы отсечения при параллельном и центральном проектировании. В первом случае такой объем представляет собой неограниченный параллелепипед, ребра которого параллельны направлению проектирования, а боковые грани проходят по сторонам окна вывода. Обычно, окно вывода – это ортогональное окно.

Рис. 2.1. Отсекающий объем при параллельном проектировании

Для центральной проекции отсекающий объем представляет собой две неограниченные пирамиды, вершины которых сходятся в центре проекции, а боковые грани проходят по сторонам окна вывода.



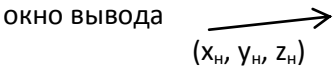


Рис. 2.2. Отсекающий объем при центральном проектировании

Обычно видимый объем ограничивается по глубине плоскостями параллельными окну вывода. Это необходимо по следующим причинам. Во-первых, можно не изображать детали сцены, расположенные за центром проецирования, что соответствует практике наблюдения. Во-вторых, это упрощает изображение за счет меньшего числа его деталей. В-третьих, объекты, близко расположенные к центру проектирования могут заслонять другие и выглядеть непропорционально большими. И, наконец, объекты далеко расположенные от центра проектирования могут оказаться слишком мелкими на изображении.

Ограничение видимого объема по глубине превращает неограниченные отсекающие объемы в прямоугольный параллелепипед и усеченную пирамиду. Обычно усечение производится плоскостями, параллельными плоскости XOY. Систему координат для прямоугольного параллелепипеда размещают так, что ось Z параллельна ребрам параллелепипеда и проходит через центр окна вывода. Аналогично размещают систему координат в усеченной пирамиде – центр проектирования – на оси Z, и ось Z проходит через центр окна вывода.

2.2. Условия полной видимости/невидимости отрезков

Рассмотренные варианты отсекающих объемов упрощают решение задачи трехмерного отсечения. Отсекающий объемы задаются координатами хл, хп, ув, ун, zб, zд., которые получаются в результате пересечения левой, правой, верхней, нижней, ближней и дальней гранями осей координат. Для центральной проекции следует указывать также точку центра проектирования, расположенную на оси Z – zц.

Решение задачи видимости/невидимости отрезка по отношению к отсекающему объему можно облегчить введением шестибитовых кодов для концевых точек отрезка. Биты кодов соответствуют положению концевой точки отрезка по отношению к шести граням отсекающего объема. Если концевая точка располагается во внешнем полупространстве по отношению к

рассматриваемой грани, то бит, соответствующий этой грани принимает значение 1, в противном случае – 0. В этом случае условия полной видимости и полной невидимости отрезка будут иметь такой же вид, как и для плоского случая, т.е.

$$(M(P_1) = 0) \,\&\, (M(P_2) = 0);$$

$$(M(P_1) \,\&\, (M(P_2) \neq 0.$$

Отрезок P_1P_2 будет полностью видимым, если коды концевых точек отрезка являются нулевыми, и отрезок P_1P_2 будет полностью невидимым, если конъюнкция кодов концевых точек отрезка - ненулевая. В остальных случаях отрезок будет полностью или частично невидимым.

Вычисление кодов точки **P**(x, y, z) при параллельном проектировании можно выполнить проверкой условий

$$x_л \le x \le x_п;$$

$$y_н \le y \le y_в;$$

$$z_б \le z \le z_д.$$

Если условия выполняются, то соответствующий код равен 0, иначе – 1.

В центральном проектировании определение положения точки относительно левой и правой граней можно определить, рассмотрев проекцию пирамиды на плоскость XOZ (вид сверху, рис. 5.2.).

Уравнение прямой, являющейся проекцией левой грани отсекающего объема, полученное из подобных треугольников, имеет вид

$$X - \frac{Z - Z_н}{- Z_н} X_л = 0.$$

Левая часть уравнения дает проверочную функцию $f_л(x, z)$ для определения положения точки относительно левой грани.

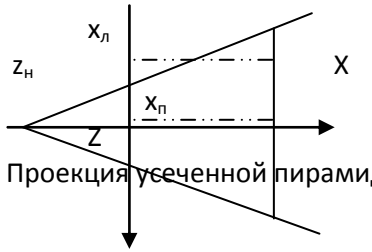


Рис. 2.2. Проекция усеченной пирамиды

При значениях функции fл (x, z) < 0 точка P(x, z) лежит левее грани. В случае равенства нулю – на грани, и, наконец, при значении fл (x, z) > 0 – правее грани. Аналогично решается задача и по определению положения точки относительно других граней. Причем, следует учитывать случай, когда z < zн. В этом случае точка одновременно оказывается левее левой и правее правой граней. Для правильного определения положения точки необходимо проинвертировать четыре младших разряда шестибитового кода точки, которые определяют положение точки по координатам X и Y.

2.3. Трехмерное отсечение

Алгоритмы трехмерного отсечения нетривиально расположенных отрезков могут быть получены естественным обобщением двумерных алгоритмов. Например, алгоритм Сазерленда-Коэна отсечения отрезка ортогональным окном можно реализовать последовательно на двух проекциях сцены на соответствующие плоскости. При этом точка пересечения отрезка с плоскостью заменяется точкой пересечения отрезка с гранью ортогонального окна. Аналогичное применение алгоритма Кируса-Бека отсечения отрезка выпуклым многоугольником в трехмерном случае приводит к решению плоской задачи на проекциях. При этом рассматривается задача отсечения отрезка не многоугольником, а ортогональным окном для случая отсекающего объема в виде параллелепипеда. Однако можно применять и любые выпуклые объемы.

Можно найти несложные обобщения для трехмерного случая и для других операций, рассматриваемых при решении плоской задачи. Например, вычисление нормали к граням объема можно получить вычислением векторного произведения двух векторов V1 и V2, лежащих в плоскости. Такими векторами могут быть вектора двух ребер, рассматриваемой грани объема. Если угол между векторами меньше 1800, то вектор произведения будет совпадать с направление оси Z, в противном случае – будет направлен в противоположном направлении.

2.4. Определение выпуклости и разбиение невыпуклых объемов

Так как многие алгоритмы работают только с выпуклыми объемами, то есть необходимость в определении выпуклости объемов, и, в случае их невыпуклости, разбиения объемов на выпуклые части.

Выявление выпуклости объема производится преобразованием пространства. Алгоритм можно сформулировать следующим образом. Выбирается произвольная грань трехмерного тела и выполняется преобразование пространства таким образом, что плоскость этой грани совмещается с плоскостью XOY. Затем вычисляются знаки положения вершин относительно плоскости XOY. Если они одного знака, то многогранник является выпуклым относительно донной грани. В противном случае – нет. Если многогранник выпуклый относительно всех его граней, то он является выпуклым.

Этот же алгоритм можно использовать для вычисления нормали к грани, а также для разрезания тела на выпуклые части. При вычислении нормалей используется тот факт, что при совмещении грани с плоскостью XOY нормаль совпадает с направлением оси Z. После возвращения грани в исходное состояние нормаль можно пересчитать обратным преобразованием пространства.

Разбиение невыпуклого тела на выпуклые части выполняется по следующему алгоритму. Последовательно производится совмещение граней объема с плоскостью XOY. Если при этом вершины имеют разные знаки, то объем можно разбить на две части плоскостью XOY. Эта операция производится для всех граней, относительно которых тело является невыпуклым.

3. УДАЛЕНИЕ НЕВИДИМЫХ ЛИНИЙ И ПОВЕРХНОСТЕЙ

3.1. Постановка задачи удаления невидимых линий и поверхностей

Построение реалистических изображений связано с решением двух задач – удаления скрытых линий и граней и закраской лицевых поверхностей. Эти задачи достаточно самостоятельны, но

взаимосвязаны между собой, объемны по вычислению и требуют высокой эффективности алгоритмов.

Удаление невидимых элементов изображения устраняет двойственность изображения. Этот факт иллюстрируется на рис. 4.1.

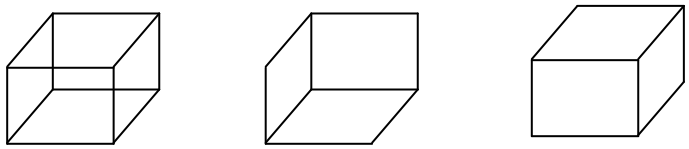


Рис. 3.1. Два варианта удаления невидимых элементов

На рисунке показаны варианты удаления невидимых элементов, реализация которых дает два разных изображения, полученных из одного каркасного представления объемного тела. Для работы с объектом необходимо иметь его описание. Решение задачи удаления невидимых линий и поверхностей предполагает определение лицевых и нелицевых граней, а также - экранированных отрезков. Криволинейные поверхности обрабатывают после аппроксимации их плоскими гранями.

В зависимости от выбора системы координат, в которой работают алгоритмы удаления, различают алгоритмы, работающие в пространстве объектов или в пространстве изображения.

Пространство объектов предполагает систему координат, в которой описаны сами объекты. Определение видимости отдельной грани выполняется путем сравнения ее с другими. В зависимости от их взаимного положения определяется видимость-невидимость указанной грани. После решения этой задачи на экран выдаются видимые элементы. Алгоритмы этого класса отличаются высокой точностью результатов, ограниченной лишь точностью расчетов, и легкостью масштабирования.

Пространство изображения оперирует с системой координат экрана, на котором строится изображение. Алгоритмы, работающие в этом пространстве, для каждой точки экрана определяют грань, видимую в этой точке. Точность вычисления ограничивается разрешающей способностью экрана. При масштабировании проявляются искажения в виде несовпадающих концов отрезков.

Возможны алгоритмы, использующие оба пространства. Сложность вычислений двух классов рассмотренных алгоритмов

следующая. Для пространства объектов n2, где n – число объектов, а для пространства изображения – n · N, где N – число точек на экране. Для сравнения сложности этих классов алгоритмов можно взять число объектов 100 000, число точек на экране более 250 000 (для Hercules), тогда сложность реализации алгоритмов, работающих в пространстве объектов (n2), составляет 1010, в пространстве изображения (n · N) – более 2,5 · 1010. Объем вычислений в первом случае меньше, но с учетом более сложной реализации одного шага алгоритма их общий объем выравнивается. Сложность конкретной реализации в сильной степени зависит от эффективности построения шагов алгоритма.

3.2. Алгоритм плавающего горизонта

Алгоритм плавающего горизонта чаше всего используется для удаления невидимых линий трехмерного представления функций, описывающих поверхность в виде:

$$F(x, y, z) = 0.$$

Предложено много алгоритмов, использующих этот подход (см. [2]-[6]). Поскольку в приложениях в основном интересуются описанием поверхности, этот алгоритм обычно работает в пространстве изображения. Главная идея данного метода заключается в сведении трехмерной задачи к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат x, y или z.

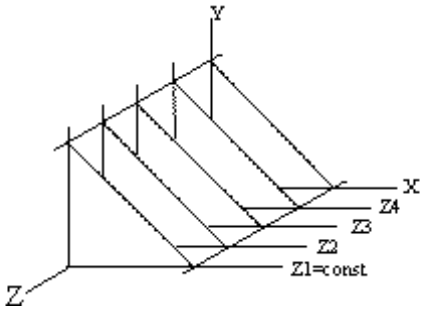


Рис. 3.2. Секущие плоскости

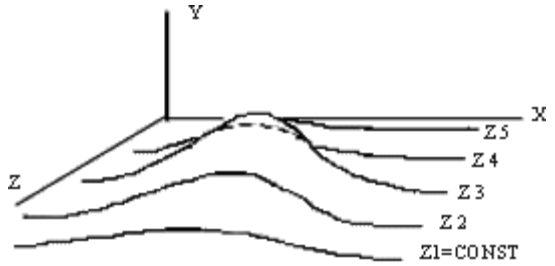


Рис. 3.3. Плоские кривые в секущих плоскостях

На рис. 3.3. приведен пример, где указанные параллельные плоскости определяются постоянными значениями z. Функция F(x,y,z) = 0 сводится к последовательности кривых, лежащих в каждой из этих параллельных плоскостей, например к последовательности

$$y = f(x, z) \text{ или } x = g(y, z),$$

где z постоянно на каждой из заданных параллельных плоскостей.

Поверхность теперь складывается из последовательности кривых, лежащих в каждой из этих плоскостей, как показано на рис. 3.4. Здесь предполагается, что полученные кривые являются однозначными функциями независимых переменных. Если спроецировать полученные кривые на плоскость z=0, как показано на рис.3.5, то сразу становится ясна идея алгоритма удаления невидимых участков исходной поверхности. Алгоритм сначала упорядочивает плоскости z=const по возрастанию расстояния до них от точки наблюдения. Затем для каждой плоскости, начиная с ближайшей к точке наблюдения, строится кривая, лежащая на ней, т. е. для каждого значения координаты x в пространстве изображения определяется соответствующее значение y. Алгоритм удаления невидимой линии заключается в следующем:

Рис. 3.4. Проекция кривых на плоскость XOY

Если на текущей плоскости при некотором заданном значении x соответствующее значение y на кривой больше значения y для всех предыдущих кривых при этом значении x, то текущая кривая видима в этой точке; в противном случае она невидима.

Невидимые участки показаны. Реализация данного алгоритма достаточно проста. Для хранения максимальных значений y при каждом значении x используется массив, длина которого равна числу различимых точек (разрешению) по оси x в пространстве изображения. Значения, хранящиеся в этом массиве, представляют собой текущие значения "горизонта". Поэтому по мере рисования каждой очередной кривой этот горизонт "всплывает". Фактически этот алгоритм удаления невидимых линий работает каждый раз с одной линией.

Аналогичная процедура выполняется и для случая «опускания» горизонта, т.е. получения значений y, меньших по сравнению со значениями кривой в плоскости z = 0. Подобные кривые, естественно, видимы и представляют собой нижнюю сторону исходной поверхности. Нижняя сторона поверхности делается видимой, если модифицировать этот алгоритм, включив в него нижний горизонт, который опускается вниз по ходу работы алгоритма. Это реализуется при помощи второго массива, длина которого равна числу различимых точек по оси x в пространстве изображения. Этот массив содержит наименьшие значения y для каждого значения x. Алгоритм теперь становится таким:

Если на текущей плоскости при некотором заданном значении x соответствующее значение y на кривой больше максимума или меньше минимума по y для всех предыдущих кривых при этом x, то текущая кривая видима. В противном случае она невидима.

3.3. Алгоритм Варнока

Поскольку алгоритм Варнока нацелен на обработку картинки, он работает в пространстве изображения. В пространстве изображения рассматривается окно и решается вопрос о том, пусто ли оно, или его содержимое достаточно просто для визуализации. Если это не так, то окно разбивается на фрагменты до тех пор, пока содержимое подокна не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения, В последнем

случае информация, содержащаяся в окне, усредняется, и результат изображается с одинаковой интенсивностью или цветом. Устранение лестничного эффекта можно реализовать, доведя процесс разбиения до размеров, меньших, чем разрешение экрана на один пиксель, и усредняя атрибуты подпикселей, чтобы определить атрибуты самих пикселей.

Конкретная реализация алгоритма Варнока зависит от метода разбиения окна и от деталей критерия, используемого для того, чтобы решить, является ли содержимое окна достаточно простым. В оригинальной версии алгоритма Варнока каждое окно разбивалось на четыре одинаковых подокна. В данном разделе обсуждаются эта версия алгоритма, а также общий вариант, позволяющий разбивать окно на полигональные области. Другая разновидность алгоритма, предложенная Вейлером и Азертоном, в которой окно тоже разбивается на полигональные подокна, обсуждается в следующем разделе. Кэтмул применил основную идею метода разбиения к визуализации криволинейных поверхностей.

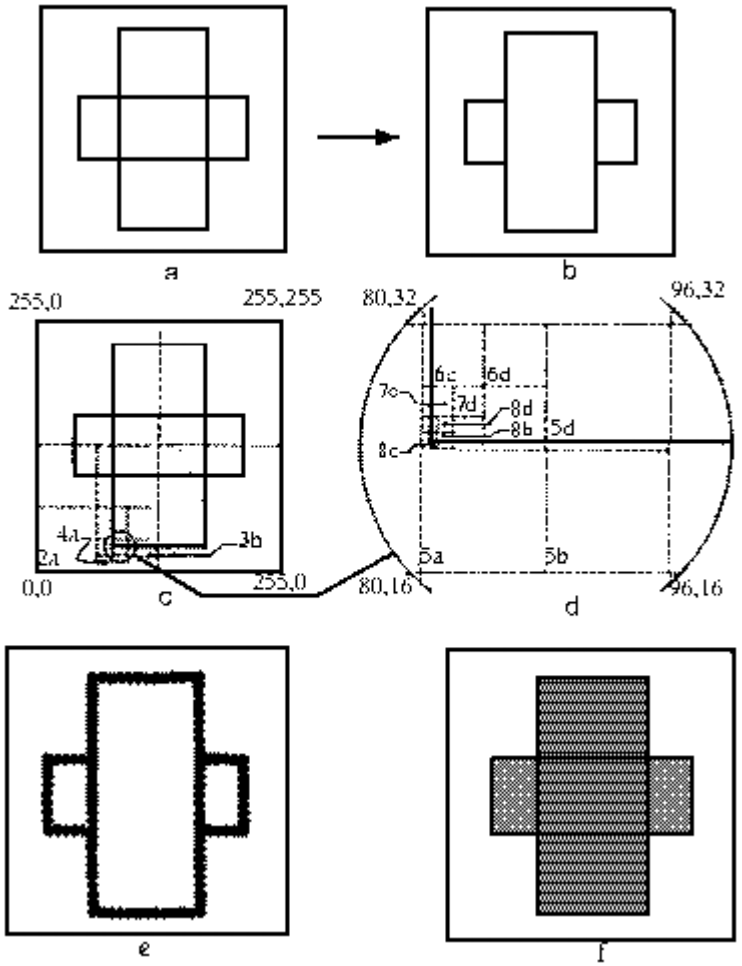


Рис. 3.5. Алгоритм разбиения Варнока

На рис. 3.5. показан результат простейшей реализации алгоритма Варнока. Здесь окно, содержимое которого слишком сложно изображать, разбито на четыре одинаковых подокна. Окно, в котором что-то есть, подразбивается далее то тех пор, пока не будет достигнут предел разрешения экрана. На рис. 3.5, a показана сцена, состоящая из двух простых многоугольников. На рис. 3.5,b показан результат после удаления невидимых линий. Заметим, что горизонтальный прямоугольник частично экранирован вертикальным. На рис. 3.5., c и d показан процесс разбиения окон на

экране с разрешением 256x256. Поскольку 256 = 2^8, требуется не более восьми шагов разбиения для достижения предела разрешения экрана. Пусть подокна рассматриваются в следующем порядке: нижнее левое, нижнее правое, верхнее левое, верхнее правое. Будем обозначать подокна цифрой и буквой,' цифра – это номер шага разбиения, а буква – номер квадранта. Тогда для окна 1a подокна 2a, 4a, 4c, 5a, 5b оказываются пустыми и изображаются с фоновой интенсивностью в процессе разбиения. Первым подокном, содержимое которого не пусто на уровне пикселей, оказывается 8a. Теперь необходимо решить вопрос о том, какой алгоритм желательно применить: удаления невидимых линий или удаления невидимых поверхностей. Если желательно применить алгоритм удаления невидимых линий, то пиксель, соответствующий подокну 8a, активируется, поскольку через него проходят видимые ребра. В результате получается изображение видимых ребер многоугольников в виде последовательности точек размером с пиксель каждая (рис. 3.5., e).

Следующее рассмотрение окна, помеченного как 8d на рис. 3.5., d, лучше всего проиллюстрирует различие между реализациями алгоритмов удаления невидимых линий и поверхностей. В случае удаления невидимых линий окно 8d размером с пиксель не содержит ребер ни одного многоугольника сцены. Следовательно, оно объявляется пустым и изображается с фоновой интенсивностью или цветом. Для алгоритма удаления невидимых поверхностей проверяется охват этого окна каждым многоугольником сцены. Если такой охват обнаружен, то среди охватывающих пиксель многоугольников выбирается ближайший к точке наблюдения на направлении наблюдения, проходящем через данный пиксель. Проверка проводится относительно центра пикселя. Затем этот пиксель изображается с интенсивностью или цветом ближайшего многоугольника. Если охватывающие многоугольники не найдены, то окно размером с пиксель пусто. Поэтому оно изображается с фоновым цветом или интенсивностью. Окно 8d охвачено вертикальным прямоугольником. Поэтому оно изображается с цветом или интенсивностью этого многоугольника. Соответствующий результат показан на рис. 3.5., f.

Возвратившись к рассмотрению окна 8а на рис. 3.5., d, покажем, как включить в алгоритм удаления невидимых поверхностей устранение лестничного эффекта. Разбиение этого окна дает четыре подокна с размерами, меньшими, чем размеры пикселя. Только одно из этих подокон – правое верхнее – охвачено многоугольником. Три других пусты. Усреднение результатов для этих четырех подпикселей показывает, что окно 8а размером с пиксель нужно изображать с интенсивностью, равной одной четверти интенсивности прямоугольника. Аналогично пиксель 8Ь следует высвечивать с интенсивностью, равной половине интенсивности прямоугольника. Конечно, окна размером с пиксель можно разбивать более одного раза, чтобы произвести взвешенное усреднение характеристик.

Процесс подразбиения порождает для подокон структуру данных, являющуюся деревом, которое показано на рис. 3.6. (*В алгоритме Варнока впервые была реализована такая структура данных, как кватернарное дерево*). Корнем этого дерева является визуализируемое окно. Каждый узел изображен прямоугольником, содержащим координаты левого нижнего угла и длину стороны подокна. Предположим, что подокна обрабатываются в следующем порядке: *abcd,* т. е. слева направо на каждом уровне разбиения в дереве. На рис. 3.6. показан активный путь по структуре данных дерева к окну 8а размером с пиксель. Активные узлы на каждом уровне изображены толстой линией. Рассмотрение рис. 3.5. и 3.6. показывает, что на каждом уровне все окна слева от активного узла пусты. Поэтому они должны были визуализироваться ранее с фоновым значением цвета или интенсивности. Все окна справа от активного узла на каждом уровне будут обработаны позднее, т. е. будут объявлены пустыми или будут подразделены при обходе дерева в обратном порядке.

3.4. Алгоритм Робертса

3.4.1. Постановка задачи. Алгоритм работает в пространстве объектов, является хорошо математизированным и решает следующие задачи:

- удаление ребер и граней, экранируемых самим телом;
- удаление ребер и граней, экранируемых другими телами;
- прорисовка линий сопряжения тел.

Вычислительная сложность алгоритма n^2, где n – число объектов трехмерной сцены. По этой причине он может применяться в ограниченных масштабах. Однако математически метод прост, решение достаточно точно, и он может использоваться для иллюстрации некоторых важных концепций и в решении частных задач. Применение некоторых улучшений алгоритма дают линейную зависимость от числа объектов, что расширяет сферу его применения.

Алгоритм работает только для выпуклых объектов. Поэтому невыпуклые объекты предварительно подвергаются разбиению на выпуклые части. Тела представляются набором пересекающихся плоскостей, в которых расположены его грани.. В общем виде уравнение плоскости описывается уравнением

ax + by + cz + d = 0,

а в матричной форме оно имеет следующий вид

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} * \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = 0.$$

Поэтому любое выпуклое твер4дое тело можно выразить матрицей тела, состоящей из коэффициентов уравнений плоскостей, т. е.

$$[V] = \begin{bmatrix} a1 & a2 & \dots & an \\ b1 & b2 & \dots & bn \\ c1 & c2 & \dots & cn \\ d1 & d2 & \dots & dn \end{bmatrix},$$

где каждый столбец содержит коэффициенты одной плоскости.

3.4.2. Положение точки относительно тела. Если точка P лежит в плоскости, то для нее выполняется условие

P*[S] = 0,

где [S] – вектор соответствующей плоскости. В противном случае знак указывает на положение точки относительно данной плоскости – по одну или другую сторону от нее. В данном алгоритме принято считать, что для точек, лежащих внутри тела выполняется условие

P*[S] > 0.

3.4.3. Проверка корректности описания тела. В связи с тем, что уравнение плоскости можно умножить на (-1) без потери равенства, то результат умножения вектора точки на вектор плоскости изменит свой знак. В связи с этим описание тела необходимо проверять на корректность так, чтобы плоскости отвечали условию P*[S] > 0 для точек, лежащих внутри тела. Для этого выбирается точка, заведомо лежащая внутри тела и выполняются последовательные вычисления произведений P*[S] для всех плоскостей. Те коэффициенты, плоскости, для которых произведения имеют отрицательные значения, все коэффициенты уравнения этих плоскостей умножают на (-1).

Пример 1. Пусть дан единичный куб, который отсекает своими гранями на осях координат отрезки $x_1 = ½$, $x_2 = -½$, $y_3 = ½$, $y_4 = -½$, $z_5 = ½$, $z_6 = -½$. Центр системы координат помещается в центре куба. Можно записать уравнение плоскостей тела в виде

$$\frac{x}{a}+\frac{y}{b}+\frac{z}{c}=1.$$

Тогда для первой плоскости получим

$$\frac{x}{\frac{1}{2}}+\frac{y}{\infty}+\frac{z}{\infty}=1.$$

Отсюда получаем уравнение 2x – 1 = 0. Аналогично получаем для второй плоскости:

$$\frac{x}{-\frac{1}{2}}+\frac{y}{\infty}+\frac{z}{\infty}=1 \text{ и } 2x + 1 = 0.$$

Отсюда имеем матрицу описания единичного куба

$$V=\begin{bmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}.$$

В качестве пробной точки можно взять точку начала координат P = [0 0 0 1]. Результат умножения вектора точки на матрицу тела дает результат P*V = [-1 1 -1 1 -1 1], который говорит о необходимости

корректировки 1, 3 и 5 плоскостей. После корректировки матрица принимает следующий вид:

$$V = \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Повторное умножение вектора точки на матрицу дает результат P*V = [1 1 1 1 1 1], который говорит о корректности матрицы.

3.4.4. Видовые преобразования. Перед решением задачи удаления невидимых линий и поверхностей может применяться видовое преобразование тела с целью получения удачного его изображения. Такое преобразование математически выглядит следующим образом:

$$V' = T^{-1}*V,$$

где T – матрица видового преобразования.

Пример 2. Выполним видовое преобразование путем переноса единичного куба вдоль оси X на 3 единицы в положительном направлении. Для этого нужна матрица преобразования вида

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 \end{bmatrix}.$$

Тогда обратная матрица будет иметь вид:

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 0 & 0 & 1 \end{bmatrix}.$$

Получим результат видового преобразования:

$$V' = T^{-1}*V =$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 7 & -5 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Из полученного следует, что 1-я и 2-я плоскости описываются уравнениями

$$-2x + 7 = 0 \text{ и } 2x - 5 = 0,$$

откуда получаем $x_1 = 3{,}5$, $x_2 = 2{,}5$. Проверка положения точки P = [0 0 0 1] относительно тела показывает

$$P*V' = [0\ 0\ 0\ 1] * \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 7 & -5 & 1 & 1 & 1 & 1 \end{bmatrix} = [7\ -5\ 1\ 1\ 1\ 1],$$

что точка лежит левее второй грани (снаружи), а точка P = [3 0 0 1] – внутри тела:

$$P*V' = [3\ 0\ 0\ 1] * \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 7 & -5 & 1 & 1 & 1 & 1 \end{bmatrix} = [1\ 1\ 1\ 1\ 1\ 1].$$

3.4.5. Определение нелицевых граней. Если наблюдатель находится на положительной полуоси Z и смотрит на начало координат, то направление взгляда идет в сторону отрицательной полуоси и имеет вектор

$$E = [0\ 0\ -1\ 0],$$

который соответствует любой точке в плоскости $z = -\infty$, т.е. точкам (x, y, $-\infty$). Поэтому, если

$$E*S < 0,$$

то точки плоскости z = -∞ лежат по отрицательную сторону от плоскости S, и плоскость невидима для наблюдателя. Из этого же следует, что все точки плоскости z = -∞ также невидимы. Такие плоскости называются нелицевыми, и они подлежат удалению. Метод эквивалентен вычислению нормали к поверхности для каждой плоскости многогранника. Отрицательность нормали к поверхности показывает, что нормаль направлена в сторону от наблюдателя и, следовательно, данная грань не видна.

Пример 3. Точка наблюдателя находится на положительной полуоси Z в +∞: P = [0 0 1 0], вектор наблюдения соответственно равен E = [0 0 -1 0]. Вычислим скалярное произведение

$$E*V = [0\ 0\ -1\ 0\ ] * \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} = [0\ 0\ 0\ 0\ 2\ -2].$$

Из чего следует, что шестая грань нелицевая.

3.4.6. Определение интенсивности закраски. Метод вычисления нормали к плоскости можно использовать и для решения простой закраски граней. Интенсивность или оттенок закраски делается пропорциональными проекции нормали к поверхности на направление вектора наблюдения.

3.4.7. Нелицевые отрезки. После определения нелицевых плоскостей определяются нелицевые отрезки, которые образуются как линии пересечения нелицевых плоскостей.

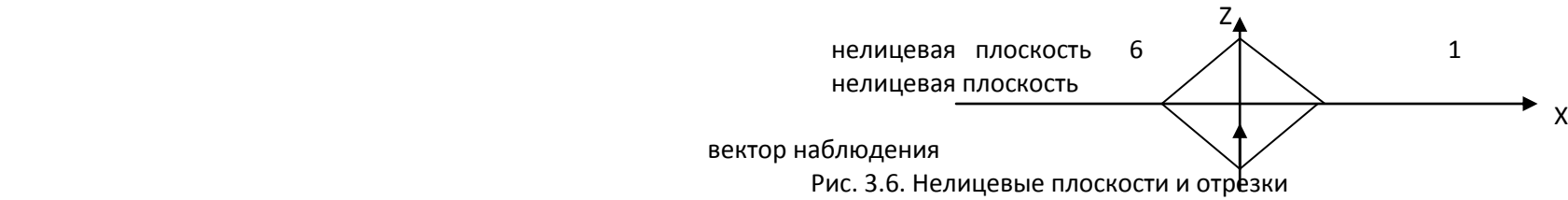


Рис. 3.6. Нелицевые плоскости и отрезки

Пример 4. Выполним видовое преобразования для начального положения единичного куба поворотом его на угол 45^0 вокруг оси У.

$$T_y = \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ тогда } T_y^{-1} = \begin{bmatrix} \cos\varphi & 0 & \sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Вычислим видовое преобразование поворота на угол 45^0.

$V' = T^{-1} * V =$

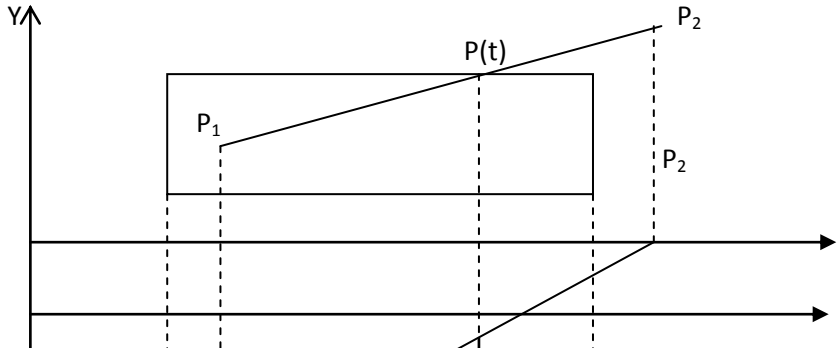
$$= \begin{bmatrix} 1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 \\ 0 & 1 & 0 & 0 \\ -1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -2/\sqrt{2} & 2/\sqrt{2} & 0 & 0 & -2/\sqrt{2} & 2/\sqrt{2} \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 2/\sqrt{2} & -2/\sqrt{2} & 0 & 0 & -2/\sqrt{2} & 2/\sqrt{2} \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Для точки наблюдения P = [0 0 1 0] вектор наблюдения E = [0 0 -1 0]. Скалярное произведения E*V' будет иметь следующее значение

E*V'= [-2/√2 2/√2 0 0 2/√2 -2/√2].

Из результата видно, что первая и шестая плоскости являются нелицевыми.

3.4.8. Экранизация отрезков другими телами. Следующей задачей этого алгоритма является решение задачи экранизации отрезков другими телами. Для этого необходимо провести сравнение отрезков и ребер с другими телами. Предварительно можно провести обработку сцены путем удаления группы отрезков или тел за счет применения минимаксных оболочек и приоритетной сортировки по глубине. При сортировке руководствуются тем принципом, что никакое тело не может экранировать отрезок, дальняя точка которого находится к наблюдателю ближе тела. Кроме того, если оболочка не перекрывает отрезок, то тело также не экранирует его. Такие приемы позволяют получать линейную зависимость сложности вычисления от числа объектов.



X

X

P(t)

P_1

Z

Рис. 3.7. Экранирование отрезков телами

Решение задачи экранирования отрезков используют параметрическую запись

$$P(t) = P_1 + (P_2 – P_1)*t.$$

В векторной форме это выражение имеет вид:

$$\mathbf{W} = \mathbf{f} + \mathbf{d}*t,$$

где $\mathbf{W} = P(t)$;

$\mathbf{f} = P_1$;

$\mathbf{d} = (P_2 – P_1)$.

Отрезок может быть полностью экранированным или частично. В последнем случае необходимо найти t при котором происходит переход отрезка из видимой части в невидимую.

Для решения этой задачи строится параметрическая плоскость. С этой целью проводится вектор из точки P(t) до точки наблюдателя.

$$\mathbf{R}(\alpha) = \mathbf{g}* \alpha.$$

Тогда параметрическая плоскость будет определена выражением

$$Q(\alpha,t) = \mathbf{W} + \mathbf{R} = \mathbf{f} + \mathbf{d}*t + \mathbf{g}* \alpha.$$

Конкретные значения α и t определяют некоторую точку плоскости, в которой находятся указанные вектора, а все множество значений этих параметров – плоскость в целом. Значения параметра α положительны, т.к. они определяют расстояние между точкой наблюдателя и точкой P(t).

Пример 5. Построить параметрическую плоскость для отрезка P_1P_2 и точки наблюдателя P = [0 0 1 0], расположенной в $+\infty$ на положительной полуоси Z. Пусть P_1 = [-2 0 -2 1], P_2 = [2 0 -2 1].

Вычислим **W** = **f** + **d***t = [-2 0 -2 1] + [4 0 0 0]* t. Тогда

Q(α,t) = **W** + **R** = **f** + **d***t + **g*** α = [-2 0 -2 1] + [4 0 0 0]* t + [0 0 1 0]* α.

Например, для t = 0,5 и α = 3 получим точку плоскости

Q(3; 0,5) = [-2 0 -2 1] + [4 0 0 0]* 0,5 + [0 0 1 0]* 3 = [0 0 1 1].

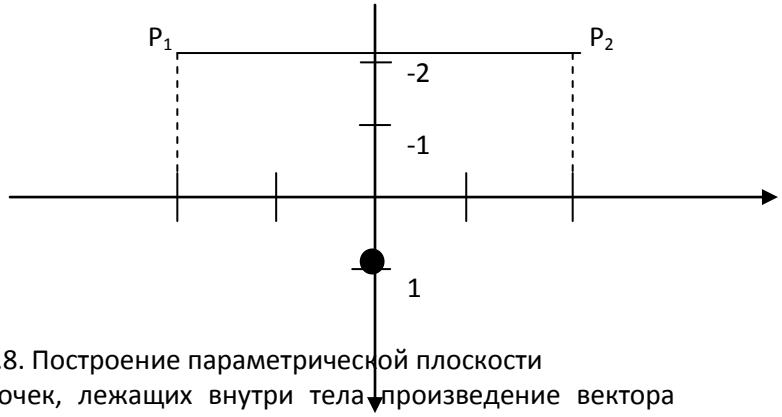


Рис. 3.8. Построение параметрической плоскости

Т.к. для точек, лежащих внутри тела, произведение вектора точки на плоскость больше нуля, то исходя из этого можно определить экранированную часть отрезка. Для этого вычислим произведение Q(α,t) на матрицу тела V:

h = **Q*** **V** = **f*** **V** + **d*** **V***t + **g*** **V*** α ≥ 0, (0 ≤ t ≤ 1, α ≥ 0).

Неотрицательные значения h соответствуют экранированию отрезка телом. Для плоскости j можно записать это в следующем виде:

$h_j = p_j + q_j*t + r_j* \alpha$,

где p_j = **f*** $\mathbf{V}_j$;

q_j = **d*** $\mathbf{V}_j$;

r_j = **g*** $\mathbf{V}_j$.

Для точки, расположенной внутри тела, условие $h_j \geq 0$ должно выполняться для всех j плоскостей. Значения $h_j = 0$ соответствуют точкам отрезка, изменяющим видимость отрезка. Они же одновременно принадлежат и плоскостям j. Нахождение этих граничных точек можно выполнить вычислением $h_j = 0$. Число таких уравнений для j плоскостей относительно α и t (сочетания из j по 2) равно j *(j-1)/2. Каждое решение, удовлетворяющее условиям 0 ≤ t ≤ 1, α ≥ 0 подставляется в остальные уравнения для проверки условия $h_j \geq 0$. При этом, в соответствии с методами линейного программирования, находятся минимальное значение t среди

максимальных и максимальное среди минимальных (ограничения сверху и снизу). Тогда отрезок невидим при $t_{max\ min} < t < t_{min\ max}$.

Пример 6.1. Выполним проверку экранирования отрезков телами. Пусть P_1 = [-2 0 -2 1], P_2 =[2 0 -2 1]. Тогда

P(t) = [-2 0 -2 1] + [4 0 0 0] * t.

Для точки наблюдения, расположенной в +∞ на положительной полуоси Z вектор в эту точку **g** = [0 0 1 0]. Вычислим $h_j = p_j + q_j * t + r_j * \alpha$:

$$p = \mathbf{f}^* \mathbf{V} = [-2\ 0\ -2\ 1] * \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} = [5\ -3\ 1\ 1\ 5\ -3].$$

q = **d*** **V =** [4 0 0 0] * **V =** [-8 8 0 0 0 0].

r = **g*** **V =** [0 0 1 0] * **V =** [0 0 0 0 -2 2].

Найденные значения используем для составления уравнений $h_j = p_j + q_j * t + r_j * \alpha \geq 0$:

$5 - 8 * t \geq 0$;

$-3 + 8 * t \geq 0$;

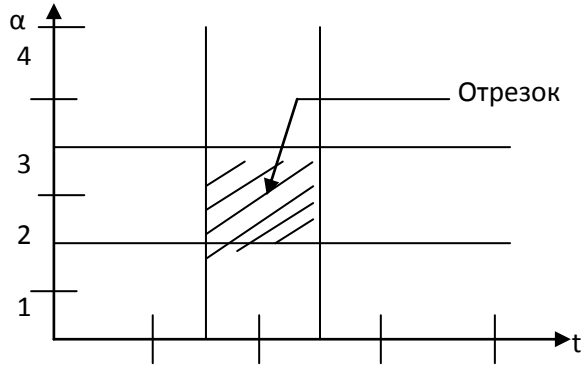
$1 \geq 0$;

$1 \geq 0$;

$5 - 2 * \alpha \geq 0$;

$-3 + 2 * \alpha \geq 0$.

При значении $h_j = 0$ из этой системы получим значения $t_1 = 5/8$, $t_2 = 3/8$, $\alpha_1 = 5/2$, $\alpha_2 = 3/2$. Найдем область значений α и t, соответствующих невидимости отрезка.



2/8 4/8 6/8 1

Рис. 3.9. График α и t для случая нескольких значений t

Можно определить $t_{max\ min}$ = 3/8 и $t_{min\ max}$ = 5/8. Координаты отрезка, соответствующие этим значениям, равны

P(3/8) = [-2 0 -2 1] + [4 0 0 0] * 3/8 = [-1/2 0 -2 1],

P(5/8) = [-2 0 -2 1] + [4 0 0 0] * 5/8 = [1/2 0 -2 1].

Пример 6.2. Рассмотрим случай существования единственного значения t. Пусть отрезок P_1P_2 имеет значения: P_1 = [1 0 -1 1], P_2 =[0 0 -1 1]. Проводим вычисления:

P(t) = [1 0 -1 1] + [-1 0 0 0] * t

p = **f*** **V** = [1 0 -1 1] * **V** = [-1 3 1 1 3 -1].

q = **d*** **V =** [-1 0 0 0] * **V =** [2 -2 0 0 0 0].

r = **g*** **V =** [0 0 1 0] * **V =** [0 0 0 0 -2 2].

Система неравенств имеет вид:

-1 + 2* t ≥ 0;

3 - 2* t ≥ 0;

1 ≥ 0;

1 ≥ 0;

3 – 2* α ≥ 0;

-1 + 2* α ≥ 0.

При значении h_j = 0 получим значения t_1 = 1/2, t_2 = 3/2, α_1 = 3/2, α_2 = 1/2. Т.к. t должно удовлетворять условию 0 ≤ t ≤ 1, то значение t_2 = 3/2 отбрасывается, как не подходящее. Для нахождения области, удовлетворяющей невидимости отрезка, можно добавить еще два уравнения: t = 0 и t = 1.

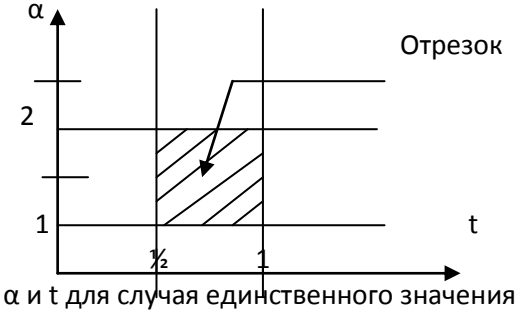


Рис. 3.10. График α и t для случая единственного значения t

Имеем $t_{max\ min}$ = ½ и $t_{min\ max}$ = 1, а координаты отрезка, соответствующие видимой части, равны

P(0) = [1 0 -1 1] + [-1 0 0 0] * 0 = [1 0 -1 1],

P(½) = [1 0 -1 1] + [-1 0 0 0] * ½ = [½ 0 -1 1].

3.4.9. Определение линий сопряжения. Линии сопряжения определяются нахождением точек протыкания ребер одного тела через грани другого. Это выполняется поиском решения для h_j при значении α =0. Затем точки протыкания соединяются отрезками. Для этого каждая точка протыкания соединяется со всеми остальными. После этого решается задача экранирования этих отрезков данными телами и удаления невидимых частей. В процессе решения этой задачи система из равно j *(j-1)/2 уравнений дополняется еще тремя: t = 0, t = 1 и α =0.

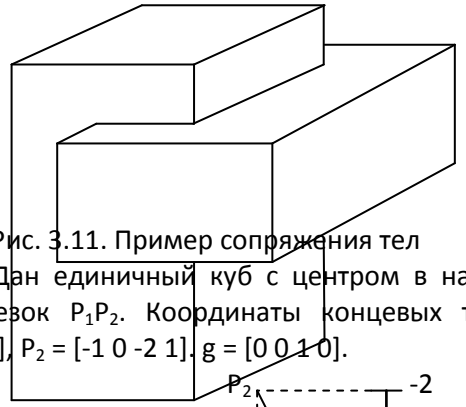


Рис. 3.11. Пример сопряжения тел

Пример 7. Дан единичный куб с центром в начале системы координат и отрезок P_1P_2. Координаты концевых точек отрезка равны: P_1 = [1 0 2 1], P_2 = [-1 0 -2 1]. g = [0 0 1 0].

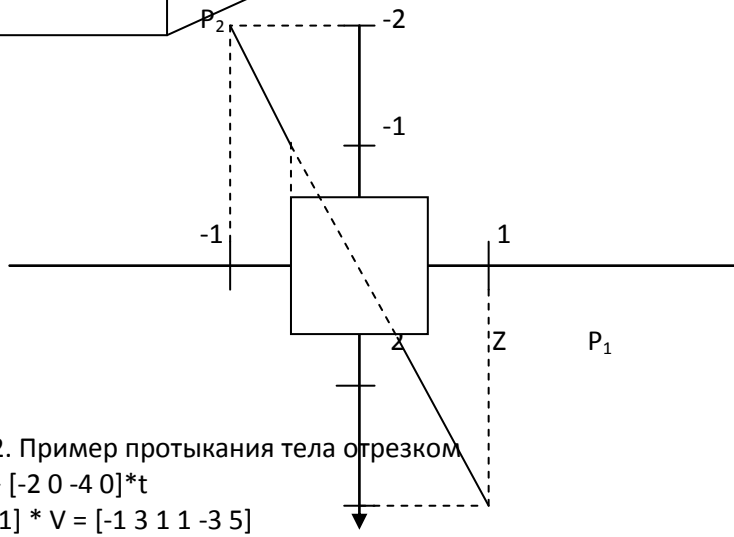


Рис. 3.12. Пример протыкания тела отрезком

P(t) = [1 0 2 1] + [-2 0 -4 0]*t

p = **f***V = [1 0 2 1] * V = [-1 3 1 1 -3 5]

q = **d***V = [-2 0 -4 0] * V = [4 -4 0 0 8 -8]

r = **g***V = [0 0 1 0] * V = [0 0 0 0 -2 2]

-1 + 4*t ≥ 0;

3 – 4*t ≥ 0;

1 ≥ 0;

1 ≥ 0;

-3 + 8*t – 2*α ≥ 0;

5 – 8*t + 2*α ≥ 0;

t_1 = ¼ ; t_2 = ¾ ;

Подставляем найденные значения t в уравнения и вычисляем значения α.

-3 + 8*¼ - 2* α = 0; α_1 = -1/2; (недопустимое значение)

-3 + 8*¾ - 2* α = 0; α_2 = 3/2;

5 - 8*¼ + 2* α = 0; α_3 = -3/2; (недопустимое значение)

5 - 8*¾ + 2* α = 0; α_4 = 1/2;

Отсюда t = ¾ ; α_1 = 3/2; α_2 = 1/2; К полученным значениям добавляем два уравнения:

t = 0 и t = 1.

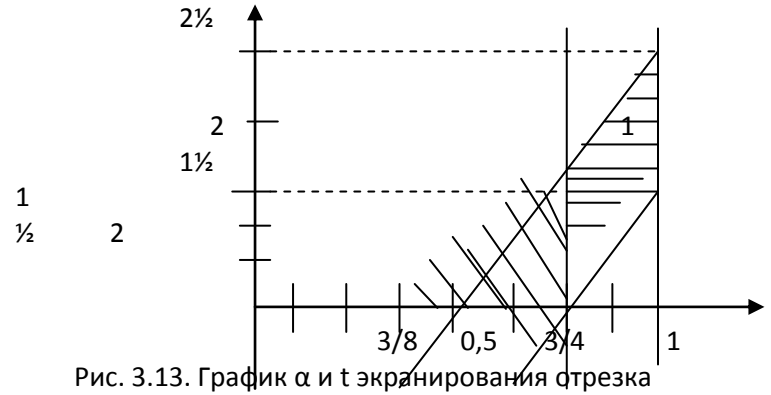


Рис. 3.13. График α и t экранирования отрезка

Область 1 некорректна, потому что при t = ¾ не удовлетворяется условие $h_j \geq 0$ для второй плоскости. Добавление границы α =0 позволяет получить корректную область 2, для которой $t_{max\ min}$ = 3/8 и $t_{min\ max}$ = ¾ .

Отрезок виден от t = 0 до t = 3/8 и от t = ¾ до t = 1:

P(0) = [1 0 2 1] + [-2 0 -4 0]*0 = [1 0 2 1]

P(3/8) = [1 0 2 1] + [-2 0 -4 0]*3/8 = [1/4 0 1/2 1]

P(3/4) = [1 0 2 1] + [-2 0 -4 0]*3/4 = [-1/2 0 -1 1]

P(1) = [1 0 2 1] + [-2 0 -4 0]*1 = [-1 0 -2 1].

3.4.10. Определение полностью видимых отрезков. Полностью видимые отрезки определяются как отрезки, расположенные между наблюдателем и видимой плоскостью. Расчет таких отрезков делают на основе анализа параметрической плоскости

$$Q(\alpha,t) = \mathbf{W} + \mathbf{R} = \mathbf{f} + \mathbf{d}*t + \mathbf{g}*\alpha.$$

Из уравнения плоскости получаем отрезок при $\alpha = 0$:

$$\mathbf{w} = \mathbf{f} + \mathbf{d}*t,$$

а концевые точки его – при $t = 0$ и $t = 1$:

$$\mathbf{w}_0 = \mathbf{f}, \mathbf{w}_1 = \mathbf{f} + \mathbf{d}.$$

Аналогичные рассуждения можно провести для $h = \mathbf{Q}*\mathbf{V} = \mathbf{f}*\mathbf{V} + \mathbf{d}*\mathbf{V}*t + \mathbf{g}*\mathbf{V}*\alpha$. При $\alpha = 0$, $t = 0$ и $t = 1$ получаем $h_j = p_j$ $(t = 0)$, $h_j = p_j + q_j$ $(t = 1)$. Последние выражения являются скалярными произведениями концевых точек отрезка на уравнение плоскости, ограничивающей тело. Плоскость видима, когда $r_j \le 0$. Поэтому, если отрезок находится между наблюдателем и видимой плоскостью j, то для него выполняются следующие условия:

$$(r_j \le 0) \,\&\, (p_j \le 0) \,\&\, (p_j + q_j \le 0).$$

Так как для полностью невидимых отрезков нет простого решения, то отдельно эта задача не решается. Невидимые отрезки выявляются на этапе решения общего алгоритма.

Пример 8. Для отрезка P_1P_2 (P_1 = [-2 0 2 1], P_2 = [2 0 2 1]) и единичного куба решим задачу нахождения полностью видимых отрезков.

Вектор наблюдения g = [0 0 1 0].

P(t) = [-2 0 2 1] + [4 0 0 0]*t

p = **f***V = [-2 0 2 1] * V = [5 -3 1 1 -3 5]

q = **d***V = [4 0 0 0] * V = [-8 8 0 0 0 0]

r = **g***V = [0 0 1 0] * V = [0 0 0 0 -2 2]

Условия $(r_j \le 0) \,\&\, (p_j \le 0) \,\&\, (p_j + q_j \le 0)$ выполняются только для пятой плоскости, т.е. отрезок является полностью видимым перед пятой плоскостью.

Аналогичный расчет для другого отрезка дает следующий результат.

P_1 = [-1 1 1 1], P_2 = [1 1 -1 1], g = [0 0 1 0].

P(t) = [-1 1 1 1] + [2 0 -2 0]*t

p = **f***V = [-2 0 2 1] * V = [3 -1 -1 3 -1 3]

q = **d***V = [4 0 0 0] * V = [-4 4 0 0 4 -4]

r = **g***V = [0 0 1 0] * V = [0 0 0 0 -2 2].

Анализ показывает выполнение условий для третьей плоскости. Это подтверждается тем, что отрезок расположен над единичным кубом по оси Y.

3.5. Алгоритм Вейлера-Азертона

Алгоритм работает в пространстве объектов и предназначен для минимизации объема вычислений в алгоритмах типа Варнока. Алгоритм состоит из четырех этапов:

- предварительная сортировка по глубине;
- отсечение по границе ближайшего к наблюдателю многоугольника (сортировка многоугольников на плоскости);
- удаление многоугольников, полностью экранированных многоугольником, ближайшим к точке наблюдения;
- при необходимости делать рекурсивное подразбиение и окончательная сортировка для устранения всех неопределенностей (анализ и корректировка результата).

Предварительная сортировка по глубине нужна для формирования списка приблизительных приоритетов. Если точка наблюдения находится на положительной полуоси Z, то ближайшим к наблюдателю многоугольком считается тот, который имеет вершину с максимальной ккординатой Z.

Отсечение объектов сцены выполняют многоугольником, который является копией ближайшего к наблюдателю многоугольника (первый многоугольник в приоритетном списке). Отсечению подлежат все многоугольники списка, включая и первый многоугольник. В результате отсечения образуется два списка – внутренний и внешний. В первый список попадают внутренние отсеченные области, а во второй – внешние. Этот этап алгоритма называется сортировкой на плоскости (XY-сортировкой).

На третьем этапе выполняют анализ глубины многоугольников из внутреннего списка. Анализ заключается в сравнении глубины каждой вершины многоугольника из внутреннего списка с минимальной Z-координатой Z_{min} отсекающего многоугольника. Если

глубина ни одной из вершин этого многоугольника не больше Z_{min}, то он полностью экранируется отсекающим многоугольником. Если этому условию отвечают все многоугольники внутреннего списка, то из внутреннего списка на изображение выводится только отсекающий многоугольник.

На заключительном этапе выполняется работа с внешним списком. Если Z-координата какого-либо многоугольника окажется больше Z_{min}-координаты, такой многоугольник, по крайней мере, частично, экранирует отсекающий многоугольник.

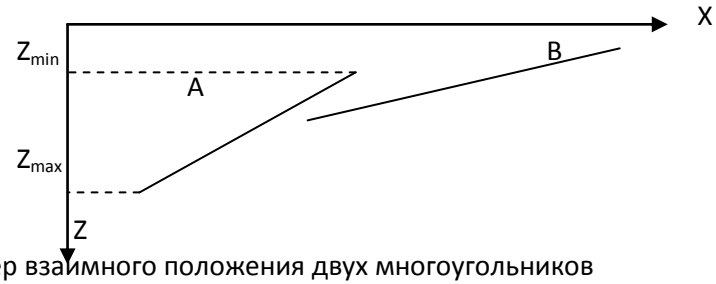
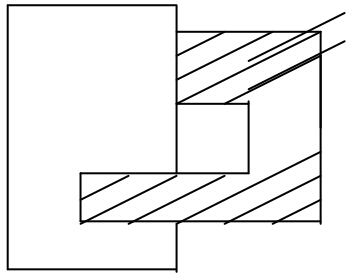


Рис. 3.14. Пример взаимного положения двух многоугольников

Например, на представленном рисунке вершина многоугольника A обладает наибольшей Z-координатой из всего приоритетного списка многоугольников. Этот многоугольник будет отсекающим. Кроме того, имеется многоугольник B, у которого Z-координата одной из вершин не меньше Z_{min}-координаты отсекающего многоугольника. В этом случае требуется замена отсекающего многоугольника A на отсекающий многоугольник B с переходом на процедуру XY-сортировки (второй этап). При этом необходимо предварительно восстановить начальный вид многоугольника B, который он имел до отсечения отсекающим многоугольником A.

Трехмерная сцена может содержать циклические многоугольники. Ниже на рисунке представлены два варианта таких многоугольников.



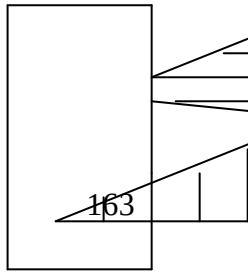


Рис. 3.15. Примеры циклических многоугольников

Здесь показаны многоугольники, циклически расположенные по отношению к отсекающему многоугольнику. В этих случаях после отсечения многоугольников отсекающим окном выполняется отсечение самого отсекающего окна циклическим многоугольником.

3.6. Алгоритм разбиения криволинейных поверхностей

Алгоритмы Варнока и Вейлера-Азертона рассчитаны на обработку сцен, составленных из плоских граней. Эти алгоритмы можно использовать и для кля криволинейных поверхностей. В этом случае поверхности аппроксимируют плоскими гранями. Однако такой подход не всегда дает адекватное представление о поверхности. Кэтмулом был предложен алгоритм для криволинейных поверхностей, аналогичный алгоритму Варнока, который не предполагает аппроксимации самой поверхности. Алгоритм заключается в последовательном разбиении поверхности на элементарные поверхности. Рекурсивное разбиение выполняется до тех пор, пока проекция каждого элемента поверхности на плоскость изображения не будет покрывать не более одного центра пиксела.. После этого вычисляются атрибуты поверхности в этом пикселе и выполняется вывод изображения соответствующего элемента поверхности в этой точке.

Эффективность метода в сильной степени завист от сложности метода разбиения криволинейной поверхности. Предложены разщными авторами методы такого разбиения в виде бикубических элементов, B-сплайнов и другие.

3.7. Алгоритм, использующий z-буфер

Алгоритм предложен Кэтмулом. Работает в пространстве изображения, и является наиболее простым алгоритмом удаления невидимых линий и поверхностей. Идея алгоритма заключается в естественном обобщении идеи буфера кадра. Здесь предполагается использование двух буферов. Буфер кадра предназначен для хранения изображения, а z-буфер – для хранения z-координат видимых элементов изображения. Видимыми элементами сцены являются те, которые являются ближайшими к наблюдателю. Таким образом, алгоритм представляет собой поиск по плоскости XY

максимального значения z = z(x,y). Основной недостаток метода – большой объем требуемой памяти. Например, для экрана с матрицей пикселей 512x512 буферы будут иметь следующие объемы. Для z-кадра – 512x512x24, для z-буфера – 512x512x20. Здесь предполагается палитра 24 бита на пиксель и значение z-координат в 20 бит. Общий объем памяти для этих буферов составляет почти 1,5 Мбайт. При современных темпах удешевления стоимости памяти данный недостаток вполне преодолим. Улучшение качества изображения путем устранения лестничного эффекта достигается за счет дополнительной детализации изображения в некоторых областях. Эта детализация выполняется повышением разрешающей способности, более высокой, чем у экрана, с последуюшим усреднением результата.

Алгоритм формулируется следующим образом.

1. Буфер кадра заполняется фоновым значением интенсивности или цвета.
2. z-буфер заполняется минимально возможным значением z по всем координатам X, Y.
3. Каждый многоугольник преобразуется в раствое изображение.
4. Для текущего пиксела растрового изображения многоугольников вычисляется z-координата.
5. Выполняется сравнение текущей z-координаты z(x,y) с соответствующим значением Z(X,Y) z-буфера. Если z(x,y) > Z(X,Y), то принимают Z(X,Y) = z(x,y).
6. Если просмотр пикселей растрового изображения многоугольников не закончен, то переход к п. 4.
7. Конец работы.

Эффективную процедуру вычисления z-координат можно построить по уравнению плоскости

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

Для z можно записать

$$z = - (ax + by + d)/c.$$

Для отдельной сканирующей **y** не меняется, а **x** можно наращивать на единицу. Тогда

$$z_{i+1} - z_i = - (ax_{i+1} + d)/c + (ax_i + d)/c = - a(x_{i+1} - x_i)/c$$

Или

$$z_{i+1} = z_i - (a/c).$$

Данный алгоритм пригоден и для построения изображения сечения объектов плоскостью $z = Z_{сеч}$. В этом случае меняется только условие сравнения в п. 5. Если $(z(x,y) > Z(X,Y)$ & $(z(x,y) \le Z_{сеч})$, то $Z(X,Y) = z(x,y)$. При таком подходе z-буфер заполняется только элементами сечения и элементами, видимыми за сечением.

3.8. Алгоритм, использующий список приоритетов

Рассмотренные ранее алгоритмы используют приоритетную сортировку по глубине – по удалению от наблюдателя. Иногда требуется повторная сортировка, предназначенная для получения окончательного списка приоритетов, при котором два любых элемента взаимно не перекрывают друг друга. Если это удается, то все элементы можно записать в буфер кадра последовательно, начиная с наиболее удаленного от точки наблюдения. При этом естественным образом решается задача определения видимых элементов сцены, так как в процессе вывода невидимые элементы накрываются выводимыми объектами сцены. При этом можно решить и задачу прозрачности за счет частичной корректировки содержимого буфера кадра вместо стирания невидимых элементов. По такой же технологии реализуется и алгоритм художника, при котором изображения строится последовательным переходом от фона к элементам переднего плана наложением элементов сцены одного на другой.

Упорядочивание по разным критериям дает различные результаты. Например, многоугольники, представленные на рисунке, при упорядочивании по Z_{min} дает приоритетный список удаленности многоугольников от наблюдателя в виде B, A. При упорядочивании по Z_{max} этот приоритетный список имеет вид A, B. Первый вариант для решения задачи лучше, т.к. многоугольник B частично экранирует многоугольник A.

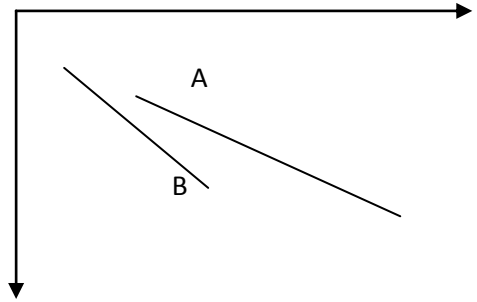


Рис. 3.16. Случай экранирования многоугольников

Возможны и более сложные случаи, когда любой критерий сортировки не дает нужного результата. Это возможно в случаях циклического экранирования или протыкания одного многоугольника другим. В этом случае для решения задачи выполняют разрезание одних многоугольников плоскостями других, так, чтобы полученные части оказывались полностью по разные стороны от циклически экранирующих многоугольников.

Для разрешения подобных конфликтных ситуаций предложен также алгоритм, использующий тесты экранизации. Список приоритетов при этом формируется динамически перед обработкой каждого кадра. Алгоритм не имеет каких-либо ограничений и может использоваться для двух и трехмерных случаев.

По данному алгоритму формируется предварительный список приоритетов по глубине с использованием минимальной координаты Z_{min} для каждого многоугольника. Самый дальний от наблюдателя многоугольник обозначается через P, а предыдущий – через Q. Выполняется проверка взаимного расположения многоугольников P и Q. Если $P_{z\_max} \le Q_{z\_min}$, то никакая часть P не экранирует Q. В этом случае P заносится в буфер кадра. В противном случае возможно экранирование многоугольника Q. Тогда Q заносится во множество {Q} многоугольников, которые возможно экранированы.

После формирования множества {Q} выполняется тестирование многоугольников, входящих в это множество. Тестирование выполняется по пяти тестам.

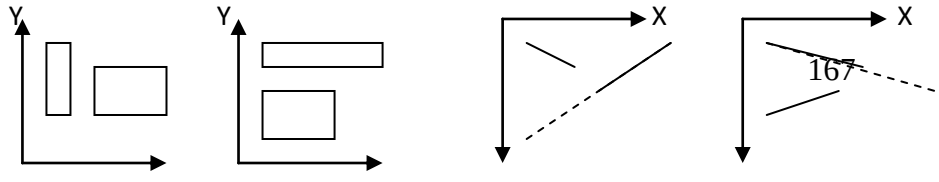
ТЕСТ1. Верно ли, что ортогональные минимаксные оболочки многоугольников P и Q не перкрываются по X?

ТЕСТ2. Верно ли, что ортогональные минимаксные оболочки многоугольников P и Q не перкрываются по Y?

ТЕСТ3. Верно ли, что P целиком лежит по ту сторону плоскости, несущей Q, которая расположена дальше от точки наблюдения?

ТЕСТ4. Верно ли, что Q целиком лежит по ту сторону плоскости, несущей P, которая расположена ближе к точке наблюдения?

ТЕСТ5. Верно ли, что проекции P и Q на плоскость XOY не пересекаются.



X X Z Z X

Рис. 3.17. Тесты для многоугольников

На рисунках представлены случаи, когда указанные тесты выполняются. Тесты применяются для всех многоугольников множества {Q}. Если выполняется хотя бы один тест, то данный многоугольник Q не экранируется. В случаях, когда все многоугольники Q не экранируется многоугольником P, то многоугольник P заносится в буфер кадра. В противном случае возможно экранирование. Тогда многоугольники P и Q меняют местами, составляют множество многоугольников {P} и подвергают их тестам. Если возникает повторная необходимость смены P и Q местами, имеет место циклического экранирования. В этом случае многоугольник P разрезается плоскостью, несущей Q и вместо многоугольника P обработку ведут для полученных частей многоугольника P. После этого работа алгоритма начинается заново.

3.9. Алгоритмы построчного сканирования

Ряд алгоритмов построен на принципе обработки сцены вдоль сканирующей строки. Они работают в пространстве изображения. Для работы этих алгоритмов проводится растровая развертка многоугольников. Использование сканирующей строки сводит трехмерную задачу удаления невидимых линий и поверхностей к анализу плоскости сканирования и решению двумерной задачи. Плоскость сканирования параллельная плоскости XOZ. Пересечение этой плоскостью многоугольников дает в сечении отрезки.

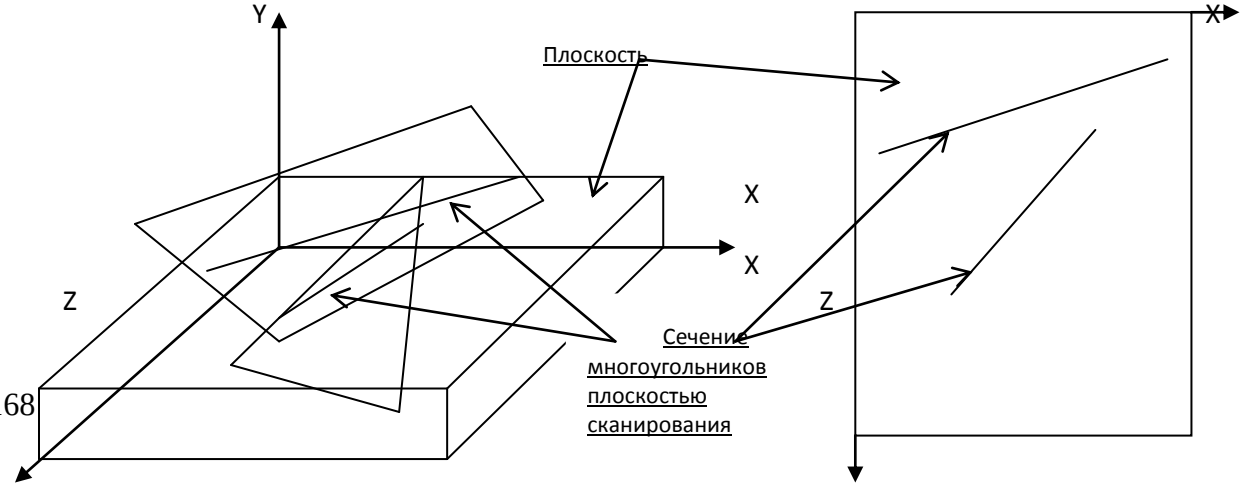


Рис. 3.18. Пересечение сцены плоскостью сканирования

Двумерная задачи видимости отрезков в плоскости сканирования из точки наблюдения сводится к определению максимальных z-координат точек отрезков. Для решения этой задачи используют два метода – с использованием Z-буфера и интервальный алгоритм.

Использование Z-буфера. В данном алгоритме, в отличие от описанного ранее алгоритма, буфер кадра имеет размер 1x512x24, а Z-буфер – 1x512x20. Эти размеры определяется размерами плоскости сканирования, палитрой и глубиной z-координат. Исходное значение буфера кадра – фоновое, а Z-буфера – минимально возможное значение z-координат. Для решения задачи выполняют последовательное сравнение z-координат текущей точки отрезков и соответствующего содержимого Z-буфера. Если значение Z-буфера меньше теущей z-координаты, то z-координаты заменяет содержимое Z-буфера. При этом в буфер кадра заносят значение пикселя, соответствующего этой координате.

Интервальный алгоритм. Данный подход отличается меньшим объемом вычислений за счет анализа не отдельных точек плоскости сканирования, а интервалов. В этом случае алгоритм определяет видимость этих интервалов. Для решения задачи плоскость сканирования разбивается на такие интервалы, характер которых, с точки зрения видимости, не меняется вдоль всего интервала. Интервалы могут быть трех типов: пустые, не содержащие отрезков, с одним отрезком и с несколькими отрезками.

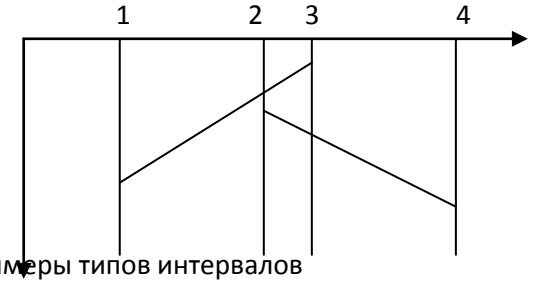


Рис. 3.19. Примеры типов интервалов

Здесь на рисунке представлена плоскость сканирования с двумя отрезками. Плоскость разбита на 4 интервала: 1 – нулевой, 2 и

4 с одним отрезком и 3 – с двумя отрезками. Анализ видимости достаточно провести на границе интервалов.

В случае, когда отрезки не имеют общих точек, то видимость их из точки наблюдения определяется наибольшими z-координатами. Если отрезки касаются в одной точке или пересекаются, то видимость отрезков определяют в середине совместных интервалов.

Рис. 3.20. Случаи наличия и отсутствия пересечений отрезков

В реализации алгоритмов возможны различные улучшения, снижающие объемы вычислений. Например, за счет использования предыдущих значений вдоль сканирующей строки.

Имеются модификации интервального алгоритма для криволинейных поверхностей. Сечение плоскости сканирования с криволинейной поверхностью дает кривую (или кривые), которая называется линией уровня или контуром. Необходимо решить задачу видимости точек этих линий уровня. Основная сложность в этом случае заключается в математическом описании линий уровня. Обычно используются параметрические формы описания кривых и плоскости.

4. РЕАЛИСТИЧЕСКОЕ ИЗОБРАЖЕНИЕ ОБЪЕКТОВ

4.1. Понятие реалистического изображения

Задача реалистического изображения трехмерных объектов и сцен тесно связана с физиологическими особенностями зрения человека. Глаз человека имеет два типа рецепторов – колбочки и палочки, которые имеют разную чувствительность к уроням освещенности. Чувствительными к высокому уровню освещенности являются колбочки, а палочки – к низким уровням. Цвет воспринимается только колбочками. Поэтому при низкой освещенности цветные предметы воспринимаются черно-белыми.

Чувствительность глаза к яркости изменяется по логарифмическому закону. Диапазон чувствительности равен примерно 10^{10}. Однако в каждом конкретном случае

чувствительность зависит от конкретной сцены. Глаз адаптируется к средней яркости сцены, и одна и та же интенсивность будет восприниматься разной на разном фоне.

В задачу реалистического изображения входит и задача отображения материала объектов, их отражательной способности, цветопередачи, расчета тени на самих объектах и окружающем пространстве от этих объектов. При этом должны решаться задачи прозрачности, удаления невидимых элементов сцены и другие.

4.2. Простая модель освещенности

Модель освещения это математическое представление физических свойств источников света и поверхностей, а также их взаимного расположения. Для моделирования освещения трехмерных объектов используются различные модели освещения.

Простая модель освещения основана на вычислении интенсивности отражаемого объектом света от точечного источника. Отражаемый от объекта свет может быть диффузным или зеркальным.

Диффузное отражение происходит при равномерном по всем направлениям рассеивании света, поэтому создается иллюзия, что поверхности имеют одинаковую яркость независимо от углов обзора. Рассеянный свет практически всегда присутствует в реальной обстановке. Свет точечного источника отражается от идеального рассеивателя по закону косинусов Ламберта:

$$I = I_l k_d \cos\theta,$$

где I_l – интенсивность источника света, I – интенсивность отраженного света, k_d – коэффициент диффузного отражения $(0 \le k_d \le 1)$, θ – угол между направлением света и нормалью к поверхности.

Ламбертова модель диффузного отражения дает блеклое и матовое изображение поверхности. При освещении объекта точечным источником на него падает также свет, отраженный от соседних объектов. Так как источник является точечным, то поверхности, на которые не падает прямой свет, будут темными. Однако, если учесть, что окружающие предметы, плоскости и поверхности, принимая свет от источника, отражают его в окружающее пространство. При этом создается рассеянное освещение. Точный расчет рассеянного света достаточно сложен.

Поэтому используют упрощенную модель рассеянного света в виде некоторого источника света. В соответствии с этим интенсивность рассеянного света вычисляется по формуле вида:

$$I = I_a k_a + I_l k_d \cos\theta,$$

где I_a – интенсивность рассеянного света, k_a – коэффициент диффузного отражения рассеянного света $(0 \le k_a \le 1)$.

В этой модели не учитывается расстояние d между источником света и объектом. По законам физики интенсивность света обратно пропорциональна квадрату расстояния. С учетом этого, при d = ∞ второй член выражения становится равным нулю, а при d стремящимся к нулю интенсивность освещенности становится большой. Такая модель плохо отражает фактическую картину. Поэтому используют линейную зависимость освещенности от расстояния.

$$I = I_a k_a + I_l k_d \frac{\cos\theta}{d+k}$$

где k – произвольная постоянная, подбираемая экспериментально.

Интенсивность зеркального отражения света зависит от угла падения, длины волны и свойств вещества поверхности, на которую падает свет. Расчет ведется по формуле Френеля.

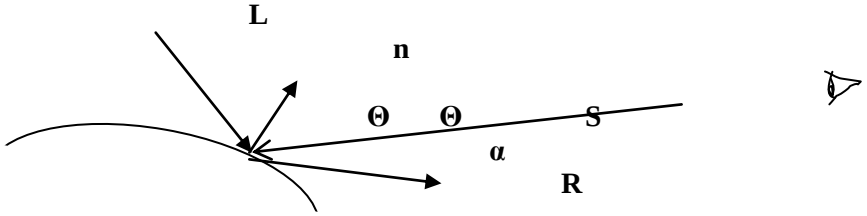


Рис. 4.1. Схема падения и отражения лучей

Зеркально отраженный свет не рассеивается. Угол отражения от идеальной отражающей поверхности равен углу падения. Зеркально отраженный свет виден только при таком угле наблюдения, когда направление взгляда S совпадает с вектором отражения R (угол α = 0). Благодаря зеркальному отражению, на блестящих предметах появляются блики. Если поверхность отражения не идеальна, то отраженный луч распределяется в пространстве вокруг вектора R, и

величина этого пространства зависит от качества поверхности, у гладких - меньше, у шероховатых – больше.

Физические свойства зеркального отражения достаточно сложны. На практике используют эмпирическую модель Фонга.

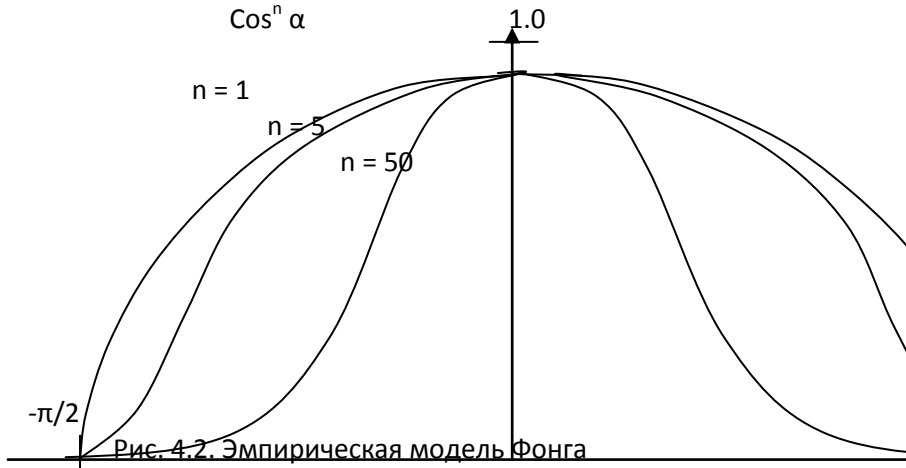


Рис. 4.2. Эмпирическая модель Фонга

Эту модель можно выразить математической зависимостью

$I_s = I_l * w(i, \lambda) * Cos^n \alpha,$

где $w(i, \lambda)$ – функция отражения;

i – угол падения;

λ - длина волны;

n- степень, аппроксимирующая пространственного распределения зеркального отражения света.

Большие значении n характерны для металлов и других блестящих поверхностей, а меньшие – для неметаллических поверхностей (бумага, дерево и т.п.). Сложную функцию $w(i, \lambda)$ обычно заменяют константой, определяемой экспериментально. С учетом этого формула, учитывающая как зеркально отраженный, так и диффузный свет имеет вид:

$$I = I_a k_a + I_l k_d \cdot \frac{k_d Cos\theta + k_s Cos^n \alpha}{d+k}.$$

В машинной графике эта модель называется функцией закраски и применяется для расчета интенсивности или тона точек объекта. Если используется несколько источников света, то их эффекты суммируются. Вычислительная сложность алгоритма построения объекта с использованием простой модели освещенности довольно высока.

При цветном изображении расчет интенсивности для каждого цвета производится отдельно, что предопределяет существенные вычислительные затраты. Для упрощения расчетов интенсивности в точках объектов используется интерполяция.

4.3. Закраска методами Гуро и Фонга

При использовании модели освещения для полигональной поверхности получают изображение, состоящее из отдельных многоугольников. Метод Гуро позволяет получить сглаженное изображение. При изображении объекта методом построчного сканирования можно рассчитать интенсивность каждого пиксела вдоль сканирующей строки. Для расчета интенсивности необходимо вычислить значение нормалей в вершинах многоугольников, которые зависят от плоскостей, примыкающих к этим вершинам. Уравнение плоскости в каноническом виде имеет вид

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

В соответствии с уравнениями для смежных плоскостей (в данном случае – трех) вектор нормали в вершине V рассчитывается по выражению

$$\mathbf{n_v} = (a_0 + a_1 + a_2)*\mathbf{i} + (b_0 + b_1 + b_2)*\mathbf{j} + (c_0 + c_1 + c_2)*\mathbf{k},$$

где a_i, b_i, c_i – коэффициенты уравнений плоскостей, примыкающих к вершине V.

Если уравнения плоскостей не заданы, то вектор нормали можно рассчитать усреднением векторов нормалей для указанных плоскостей. В свою очередь вектора нормалей к плоскостям можно вычислить векторным произведением векторов, лежащих в этих плоскостях. В качестве таких векторов удобно использовать ребра соответствующих граней поверхности.

Расчет интенсивности пикселей выполняется последовательно. Сначала выполняется расчет интенсивности в вершинах. Затем по интенсивности вершин вычисляются интенсивности необходимых

точек ребер граней. На заключительном вычисляются интенсивности отдельных пикселей. Например, для рисунка, приведенного ниже, последовательность вычислений будет следующей.

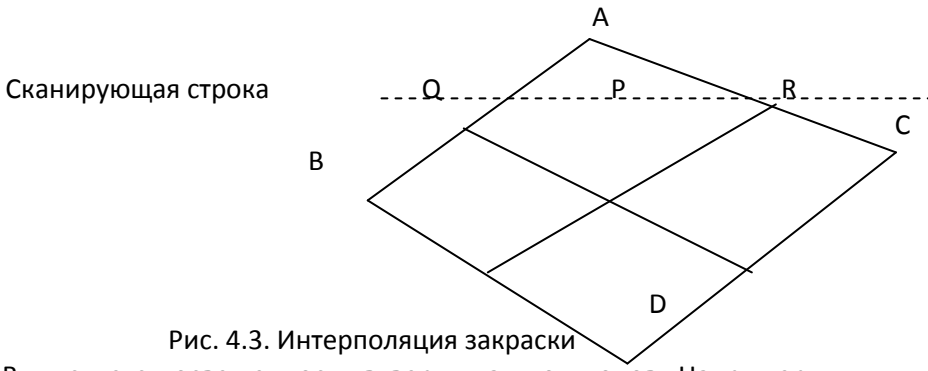


Рис. 4.3. Интерполяция закраски

Вычисляется освещенность в вершинах полигонов. Например, для грани 1 – в вершинах A, B, C и D (I_a, I_b, I_c, I_d). Линейная аппроксимация интенсивностей вершин A и B позволяет определить интенсивность в любой точке ребра AB. Расчет интенсивности в точке Q выполняется по выражению

$$I_Q = u*I_A + (1 – u)*I_B,$$

где u = QB/AB. Причем $0 \le u \le 1$.

Аналогично, по интенсивностям в вершинах B и C можно вычислить интенсивность в произвольной точке ребра BC.

$$I_R = w*I_B + (1 – w)*I_C,$$

где w=RC/BC. Коэффициент w также удовлетворяет условию $0 \le w \le 1$.

Вычисленные интенсивности в точках Q и R позволяют вычислить интенсивность в произвольной точке P грани, расположенной на сканирующей строке, выполнив вторичную линейную аппроксимацию.

$$I_P = t*I_Q + (1 – t)*I_R,$$

где t = PR/QR. Коэффициент t удовлетворяет условию $0 \le t \le 1$.

Вычисление интенсивностей точек вдоль сканирующей строки можно упростить. Для этого, например, для точек P_1 и P_2 вычисляются интенсивности

$$I_{P1} = t_1*I_Q + (1 – t_1)*I_R,$$
$$I_{P2} = t_2*I_Q + (1 – t_2)*I_R,$$

где $t_1 = P_1R/QR$, $t_2 = P_2R/QR$.

Тогда

$$I_{P2} - I_{P1} = (t_2 - t_1)*I_Q - (t_2 - t_1)*I_R = (t_2 - t_1)*(I_Q - I_R) = \Delta t*\Delta I.$$

Таким образом, если известна интенсивность в точке P_1, то интенсивность в точке P_2 определяется по простому выражению

$$I_{P2} = I_{P1} + \Delta t*\Delta I.$$

А если расчет идет от точки к точке с шагом 1, то второй член выражения становится константой $\Delta I/QR$.

Данный метод закраски имеет недостаток в виде появления эффекта «полос Маха», который проявляется в изменении интенсивности при фактически постоянной освещенности.

Возможны и другие нежелательные эффекты. Например, грани поверхности ориентированы в пространстве по-разному, но так что их нормали в вершинах дают одинаковый результат. Тогда величина ΔI будет иметь нулевое значение и, следовательно, интенсивность закраски этих граней будет одинаковой, и они будут неразличимы. Устранение этого эффекта добиваются специальными приемами разбиения граней на дополнительные многоугольники. Наиболее рационально применение этого метода для диффузного отражения, т.к. при зеркальном отражении форма бликов сильно зависит от выбора аппроксимирующих многоугольников.

Метод закраски Фонга строится по тому же принципу, что и метод Гуро. Отличие заключается в том, что билинейная аппроксимация выполняется не для интенсивностей освещения, а для векторов нормалей к граням. В вышеприведенных вычислениях по методу Фонга следует заменить интенсивность в точке на вектор нормали в этой точке. Метод отличается большим объемом вычислений, но имеет ряд преимуществ. В этом методе слабее проявляется эффект полос Маха. Но на сферах метод Фонга дает еще больший эффект полос Маха. Кроме того, оба метода приводят к ошибкам при закраске невыпуклых многоугольников, т.к. нарушается равномерность закраски.

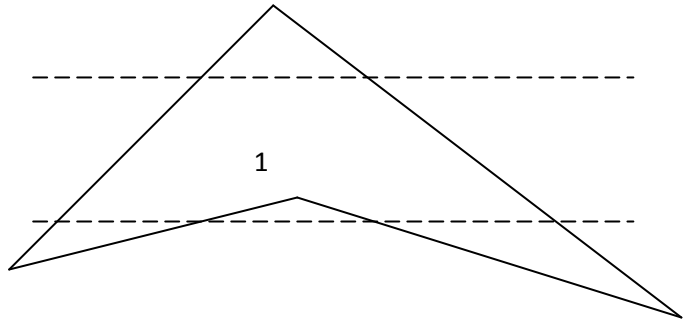


Рис. 4.4. Пример скачка интенсивностей

Например, на представленном рисунке сканирование на участке 1 и участке 2 различаются. Расчет интенсивностей на них ведется по разным ребрам с разными интенсивностями. Прохождение сканирующей строки через вершину **A** будет сопровождаться скачком интенсивностей точек, что практически является необоснованным.

Проектное задание

Разработать программу построения изображения трехмерной сцены на экране дисплея, содержащей несколько объектов. Объекты должны иметь разную конфигурацию (криволинейная поверхность плоские многоугольники, тела, ограниченные плоскими гранями). Сцена имеет источник освещенности с переменными координатами и интенсивностью света. Выполнить расчет тени и наложить текстуру на изображение объектов.

ТЕСТ РУБЕЖНОГО КОНТРОЛЯ №1

Тест содержит 10 заданий, на выполнение которых отводится 5 минут. Для ответа нужно выбрать наиболее правильный, на Ваш взгляд, вариант ответа, сделав соответствующую пометку в бланке ответов. Весовой коэффициент каждого вопроса составляет 0,1.

1. Какому условию должен отвечать коэффициент W в однородных координатах?			
1)	W = 0	2)	W ≠ 0
3)	W ≤ 0	4)	W ≥ 0
*2. Какому условию должны отвечать две точки A и B пространства, расположенные по разные стороны от плоскости, если f(C) = **n*****C**. Здесь **n** – вектор нормали к плоскости, а **C** – вектор точки пространства?*			
1)	f(A)* f(B) = 0	2)	f(A)* f(B) < 0
3)	f(A)* f(B) > 0	4)	f(A)* f(B) ≠ 0
3.Укажите условие полной видимости отрезка для параллельных проекций в трехмерном пространстве			
1)	$(M(P_1) = 0) \vee (M(P_2) = 0)$;	2)	$(M(P_1) \neq 0) \& (M(P_2) \neq 0)$;
3)	$(M(P_1) \neq 0) \vee (M(P_2) \neq 0)$;	4)	$(M(P_1) = 0) \& (M(P_2) = 0)$;
4. Если отрезок в пространстве не отвечает условиям полной видимости и полной невидимости, то что можно сказать о его видимости/невидимости?			
1)	Отрезок частично видимый	2)	Отрезок частично невидимый
3)	Отрезок частично видимый или полностью невидимый	4)	Отрезок полностью невидимый
5. Для каких трехмерных объектов используется алгоритм плавающего горизонта?			
1)	Для тел, ограниченных плоскими гранями	2)	Для криволинейных поверхностей
3)	Для плоских многоугольников, произвольно ориентированных в пространстве	4)	Для произвольных тел

6. Как можно вычислить нормаль к плоскости?			
1)	Вычислением векторного произведения векторов, лежащих в плоскости	2)	Вычислением скалярного произведения векторов, лежащих в плоскости
3)	Перемножением модулей векторов на синус угла между ними	4)	Перемножением модулей векторов на косинус угла между ними
7. Как определяется интенсивность закраски плоских граней?			
1)	Величиной проекции нормали к поверхности грани на направление вектора наблюдения	2)	Величиной проекции нормали к плоскости на направление к источнику света
3)	Скалярным произведением вектора нормали на вектор наблюдения	4)	Векторным произведением вектора нормали на вектор наблюдения
8. Какой размер имеет вектор точки пространства в однородных координатах?			
1)	2	2)	3
3)	4	4)	1
9. Как проверить корректность описания граней трехмерного тела в матрице по алгоритму Робертса?			
1)	умножить коэффициенты уравнения плоскости на W	2)	умножить коэффициенты уравнения плоскости на (-1)
3)	разделить коэффициенты уравнения плоскости на W	4)	разделить коэффициенты уравнения плоскости на наибольший коэффициент
10. Укажите параметрическую форму описания плоскости			
1)	$Q(\alpha,t) = \mathbf{f} + \mathbf{d}*t + \mathbf{g}*\alpha$	2)	$\mathbf{w} = \mathbf{f} + \mathbf{d}*t,$
3)	$\mathbf{w} = \mathbf{f} - \mathbf{d}*t,$		$Q(\alpha,t) = \mathbf{f} - \mathbf{d}*t + \mathbf{g}*\alpha$

Бланк ответов

№	1	2	3	4	5	6	7	8	9	10
1)										
2)										
3)										
4)										

ДРУГИЕ ВИДЫ ТЕСТИРОВАНИЯ ПО МОДУЛЮ

Выполнение лабораторных работ

По разделу трехмерной графики студенты могут выполнять дополнительно лабораторные работы:

- «Двумерная обработка изображений» (методическое руководство № 3054)
- «Графический интерфейс Windows» (методическое руководство № 3622)
- «Программирование на OpenGL» (методическое руководство № 3973)
- «Программирование SVGA-графики» (методическое руководство № 4245)

Выполнение курсовой работы

Курсовая работа предполагает дополнительную самостоятельную работу студентов по разделам «Компьютерной графики», которые недостаточно охвачены лекционным материалом. В основном это разделы, связанные с трехмерной графикой, обработкой изображений, фильтрацией и работой с графическими библиотеками OpenGL, DirectX. Студент должен подобрать соответствующую литературу, освоить методику работы по теме задания, разработать программу, отладить её до рабочего состояния и написать пояснительную записку. На заключительном этапе выполняется защита курсовой работы.

СПИСОК ЛИТЕРАТУРЫ

1. Роджерс Д. Алгоритмические основы машинной графики.- М.: Мир, 1989, 504 с.
2. Е.В.Шикин, А.В.Боресков. Компьютерная графика. Динамика, реалистические изображения. М.:"Диалог-МИФИ", 1995, 287 с.
3. В.П.Иванов, А.С.Батраков. Трехмерная компьютерная графика. М.:Радио и связь, 1995, 224 с.
4. Джон Корриган. Компьютерная графика. Секреты и решения, пер. с англ., М."Энтроп", 1995, 350 с.
5. Начала компьютерной графики. Под ред. Шикина Е.В. М.: "Диалог-МИФИ", 1993, 138 с.
6. Роджерс Д., Адамс Дж. Математические основы машинной графики.- М.: Мир, 1989.
7. Аммерал Л. Принципы программирования в машинной графике. Пер. с англ. М.: "Сол Систем", 1992, 224 с.
8. Аммерал Л. Машинная графика на персональных компьютерах. Пер. с англ. М.: "Сол Систем", 1992, 232 с.
9. Аммерал Л. Интерактивная трехмерная машинная графика. Пер. с англ. М.: "Сол Систем", 1992, 317 с.
10. Аммерал Л. Программирование графики на Турбо Си.- Пер. с англ. М.: "Сол Систем", 1992, 222 с.
11. Фролов А.В.,Фролов Г.В. Программирование видеоадаптеров CGA, EGA и VGA. М.: "Диалог-МИФИ", 1992, 287 с.
12. Павлидис Т. Алгоритмы машинной графики и обработки изображений.- М.: Радио и связь, 1986.
13. Фоли Дж., А. вэн Дэм. Основы интерактивной машинной графики. Кн.1.- М.: Мир, 1985.
14. Фоли Дж., А. вэн Дэм. Основы интерактивной машинной графики. Кн.2.- М.: Мир, 1985.
15. В.Ю.Романов. Популярные форматы файлов для хранения графических изображений на IBM PC. М.:"Унитех", 1992, 156 с.

16. А.В. Фролов, Г.В. Фролов. Графический интерфейс GDI в MS WINDOWS. М.: «Диалог-МИФИ», 1994, 288 с.
17. Порев В.Н. Компьютерная графика. – СПб: БХВ-Петербург, 2002. – 432 с.
18. Иванов В.П., Батраков А.С. Трехмерная компьютерная графика. – М.: Радио и связь, 1995. – 224 с.
19. Соколенко П.Т. Программирование SVGA-графики для IBM PC. – СПб: БХВ-Петербург, 2001. – 432 с.
20. Хилл Ф. OpenGl. Программирование компьютерной графики. Для профессионалов. – СПб: Питер, 2002. – 1088 с.
21. Ламот Андре. Программирование трехмерных игр для Windows. Советы профессионала по трехмерной графике и растеризации/ Пер. с англ. – М.: Издат. дом «Вильямс», 2006. – 1424 с.

ТОЛКОВЫЙ СЛОВАРЬ

Антиалиасинг (Antialiasing) — устранение ступенчатого эффекта растровых изображений, сглаживание путем задания цветов отдельных угловых пикселов.

Аффинное преобразование — линейное преобразование, например, преобразование координат.

Битовый массив (bitmap) — растр, который сохраняется в памяти или на диске.

Векторизация (vectorization) — преобразование в векторную форму описания из растровой или другой формы.

Векторная графика — создание изображений на основе векторного описания отдельных объектов.

Видеоадаптер — устройство, с помощью которого непосредственно формируется изображение на экране монитора компьютерной системы. Создание изображения осуществляется на основе данных, которые присылаются из процессора и памяти.

Видовое преобразование (view transform) — преобразование координат согласно ракурсу показа пространственных объектов.

Воксел (voxel — volume picture element) — элемент объемного растра. Массивы в. используются для моделирования объемных трехмерных объектов.

Глубина цвета (Color Depth, Bit Depth)

Количество двоичных разрядов, отводимых для кодирования цветовых оттенков для каждого пиксела изображения. Черно-белое штриховое изображение имеет глубину цвета, равную 1 биту, изображение в градациях серого — 8 битов и т. д. Количество цветовых, оттенков равняется двум в степени глубины цвета, например, количество градаций серого (256) получается и ;

"восьмой степени двойки".

Градации серого (Grayscale)

Цветовой режим с 256 уровнями (градациями) серого цвета.

Графический интерфейс пользователя (Graphical User Interface, GUI)— набор графических элементов, которые предусмотрены для пользователей компьютерной системы для выполнения некоторых операций.

Гуро метод — способ закрашивания граней трехмерных объектов, который использует интерполяцию интенсивностей отражения света в вершинах граней.

Дизеринг (dithering) — иллюзия оттенка цвета, созданная смешением близко расположенных точек различных цветов.

Заливка (Fill)

Заполнение выделенного объекта оттенком серого цвета, сплошным цветом или декоративными образцами. К заливкам относятся и градиенты (растяжки).

Интерактивная компьютерная графика — понятие, которое использовалось для того, чтобы подчеркнуть наличие аппаратных и программных способов диалога с человеком в графической компьютерной системе.

Интерфейс графического устройства (Graphic Device Interface, GDI)— подсистема операционной системы Windows.

Компьютерная графика — создание изображений с помощью компьютера.

Контекст (context) — указывает место графического вывода. С контекстом ассоциируется поверхность отображения и структура данных, которые описывают основные параметры.

- *К.* графического устройства (device context) — фундаментальное понятие графики для функций API Windows.
- *К.* отображения (rendering context) — контекст для функций библиотеки OpenGL.

Линиатура— количество точек (линий) на единицу длины. Используется для характеристики растеризации методом дизеринга.

Метафайл (metafile) — описание изображения в файле, которое содержит операторы графики в соответствующей последовательности.

Муар — видеоэффект, узор, который возникает вследствие взаимодействия растровых структур изображения и растровых элементов устройства отображения.

Окно (window) — фрагмент плоскости графического вывода. В операционной системе Windows это фундаментальное понятие, которое ассоциируется с выполняемой программой.

Операционная система — программа, которая управляет ресурсами компьютера и другими программами. Пример *о.с.* — Windows, Unix, MacOS.

Палитра (pallete) —набор цветов, важных для определенного изображения.

Пиксел (pixel — picture element) — элемент растра.

Плоттер (plotter) — векторное устройство для отображения на бумаге.

Полигон (polygon) — многоугольник, фигура, которая ограничивается контуром связанных отрезков прямых.

Полилиния (polyline)—ломаная линия связанных отрезков прямых.

Растеризация (rasterization) — создание растрового изображения на основе векторного (или другого) описания элементов изображения.

Растровое изображение — изображение, созданное множеством близко расположенных точек различного цвета (пикселов).

Рендеринг (rendering)— процесс отображения информации в графическом виде. Как правило, это относится к созданию изображений трехмерных объектов.

Разрешающая способность растра (resolution) — характеристика растров и растровых устройств. Измеряется в количестве пикселов на единицу длины, например, в дюймах (dpi).

- Оптическая ***р. сп. р.*** — характеризует оптическую систему растровых устройств ввода-вывода.
- Интерполированная ***р. сп. р.***— выше оптической благодаря интерполяции.

Сканер (scanner) — устройство для ввода графических изображений в компьютер.

Спрайт (sprite) — растровое изображение отдельного объекта рисунка, которое сохраняется в битовом массиве и быстро копируется в нужное место. Спрайты широко используются в анимации.

Тексел (texel — texture element) — элемент растровой текстуры.

Текстура (texture) — стиль закрашивания, который создает иллюзию рельефности поверхности объекта. Часто используется в виде растровых образцов (битмапов).

Трассировка лучей (raytracing) — методы создания реалистичных изображений, основанные на отслеживании распространения световых лучей.

- Обратная ***тр. л.*** — отслеживание световых лучей в обратном порядке, то есть, начиная от точки наблюдения и далее, к источникам света.

Триангуляция — формирование модели поверхности в виде множества связанных треугольников.

Фильтрация текстур — способ коррекции, интерполяции изображения при наложении текстуры на поверхность объектов.

Фонга метод — способ закрашивания граней трехмерных объектов, который основывается на интерполяции векторов нормалей в вершинах.

Фракталы (fractals) — объекты сложной формы, которые описываются простыми циклами итераций.

Шрифт (font) — набор знаков символов для представления текста в полиграфии, компьютерных системах, причем для этих знаков характерны единство стиля, размеров, одинаковость способов отображения.

- **Растровый *шр.*** — набор растровых изображений символов.
- **Векторный *шр.*** — использует описание символов в векторной форме, благодаря чему может гибко изменять размеры и форму текста.
- **TrueType—** формат шрифтов для программ операционной системы Windows, является разновидностью векторного шрифта— использует описание формы символов B-сплайнами.

Цветовая модель (Color Model, Color Mode) – визуальное и цифровое представление параметров цвета в зависимости от требований практики.

Цветовая модель CMYK - цветовое пространство, основанное на четырех цветах полиграфического процесса — голубом, пурпурном, желтом и черном.

Цветовая модель HSB – цветовое пространство, основанное на трех характеристиках цвета: цветовом тоне (hue), насыщенности (saturation) и яркости (brightness).

Цветовая модель RGB – цветовое пространство, основанное на трех цветах — красном (red), зеленом (green) и синем (blue).

Цветовой охват (Gamut) – диапазон цветов, которые способны обеспечить устройства ввода (сканер, цифровая камера) и вывода (монитор, принтер, печатный пресс).

Цветовой тон (Hue) – основная характеристика цвета, отличающая его от других цветов, например, оранжевый от синего, фиолетовый от розового и т. д. Используется в модели HSB.

Яркость (Brightness) – характеристика цвета, определяющая интенсивность цвета. Используется в цветовой модели **HSB.**

СПИСОК СОКРАЩЕНИЙ

AGP (Accelerated Graphics Port) — локальная шина обмена информацией между видеоадаптером, процессором и оперативной памятью.

API (Application Program Interface) — интерфейс для разработки прикладных программ.

BMP — растровый графический формат файлов, который широко используется программами в операционной среде Windows. Изображение сохраняется в форме битового массива.

CMYK (Cyan Magenta Yellow blacK) — субтрактивная цветовая модель.

DirectX— подсистема Windows для графического вывода.

Direct3D — программный интерфейс API, разработанный в Microsoft для трехмерной графики в Windows.

DPI (dots per inch) — количество точек на дюйм длины. Единица измерения разрешающей способности растра.

DXF — векторный графический формат файлов. Разработан Autodesk.

EMF (Enhanced MetaFile) — векторный графический формат для программ в среде Windows.

FIF (Fractal Image Format) — формат файлов для изображений, которые сжаты методом фрактального сжатия.

FPS (frames per second) — количество кадров в секунду. Единица измерения скорости видеосистемы для мультимедиа.

GIF — растровый графический формат, который широко используется в Internet. Разработан CompuServe.

GLide — программный интерфейс API, разработанный в 3Dfx Interactive для графических программ, использующих графические видеоадаптеры семейства Voodoo.

HTML — формат файлов для документов, в которых присутствуют текст, графика и другие элементы. Широко используется в Internet.

JPEG (Joint Photographic Experts Group) — стандарт формата файлов для растровых изображений с эффективным сжатием информации.

LPI – единица измерения линиатуры полиграфического растра — линий в дюйме (lines per inch).

LZW — метод сжатия информации, который используется, например, в файлах формата GIF.

MIP mapping — сохранение нескольких вариантов текстуры для различных ракурсов, масштабов показа, что обеспечивает улучшение вывода текстурированных поверхностей.

MPEG (Moving Pictures Expert Group) — стандарт для цифрового кодирования компьютерных видеофильмов.

OpenGL (Open Graphic Library) — библиотека графических функций, интерфейс для графических прикладных программ. Разработана Silicon Graphics.

PCX — популярный растровый графический формат. Разработан ZSoft.

PDF (Portable Document Format) — формат файлов электронных документов. Файл может включать текст, графику (растровую и векторную) и прочие данные. Разработан Adobe.

RAM (Random Access Memory) — оперативная память компьютера.

VRAM (Video RAM) — видеопамять, кадровый буфер в видеоадаптерах

RGB (Red Green Blue) — аддитивная цветовая модель, согласно которой цвет кодируется тремя компонентами — красным, зеленым и синим.

RGBA (Red Green Blue Alpha) — компоненты описания цвета и прозрачности для элементов изображения.

RLE (Run Length Encoding) — метод сжатия информации. Используется, например, в файлах PCX

TIFF (Tag Image File Format) — растровый графический формат, который используется для обмена графическими данными

VRML (Virtual Reality Modeling Language) — компьютерный язык описание трехмерных объектов и сцен. Используется в Internet

WMF (Windows MetaFile) — векторный графический формат для программ в среде Windows.

XYZ — название цветовой модели, принятой Международной Комиссией по Освещению.

Z-буфер — массив, в котором сохраняются значения расстояния до точки наблюдения (глубина) для каждого пиксела растрового изображения.

Оглавление

ВВЕДЕНИЕ ... 3
ОБЩИЕ ПОЛОЖЕНИЯ ... 5
Модуль № 1 Алгоритмы растровой графики и преобразования точки на плоскости ... 6
Комплексная цель ... 6
Содержание модуля ... 6
1. АЛГОРИТМЫ РАСТРОВОЙ ГРАФИКИ ... 6
1.1. Особенности растровой графики ... 6
1.2. Растровая развертка отрезков ... 8
1.3. Растровая развертка окружности ... 15
1.4. Заполнение сплошных областей ... 21
1.5. Обработка фрагментов изображений. Растровые шрифты ... 32
2. ГЕОМЕТРИЧЕСКИЕ ПРЕОБРАЗОВАНИЯ ... 36
2.1. Элементарные преобразования точки ... 36
2.2. Композиции элементарных преобразований ... 38
2.3. Операции с точками и прямыми ... 40
3. ДВУМЕРНЫЕ ОТСЕЧЕНИЯ ... 43
3.1. Отсечение отрезков регулярным окном ... 43
3.2. Задача отсечения отрезков многоугольником ... 50
3.3. Отсечение многоугольников ... 61
3.4. Вычисление нормали к ребрам, скалярного произведения двух векторов и определение выпуклости многоугольников ... 66
4. ВИДЕОСИСТЕМЫ КОМПЬЮТЕРОВ ... 69
4.1.Типы графических устройств ... 69
4.2. Графические дисплеи на запоминающей трубке ... 69
4.3. Векторные графические дисплеи с регенерацией изображений ... 70
4.4. Растровые графические дисплеи ... 71
4.5. Видеосистемы компьютера ... 72
4.6. Архитектура видеоадаптеров ... 73
4.6.1.Монохромный адаптер Hercules ... 73
4.6.2.Видеоадаптер CGA ... 76
4.6.3. Видеоадаптеры EGA и VGA ... 78
4.6.4. Архитектура видеоадаптера SVGA ... 86
4.6.5. Программирование видеоадаптеров ... 90
4.6.6. Чтение и запись для видеоадаптеров EGA, VGA и SVGA ... 94
4.7. Функции BIOS по прерыванию 10h ... 112
4.8. Стандарт VESA ... 114
Проектное задание ... 115
ТЕСТ РУБЕЖНОГО КОНТРОЛЯ №1 ... 116
Другие виды тестирования по модулю ... 118
ВВЕДЕНИЕ ... 119
ОБЩИЕ ПОЛОЖЕНИЯ ... 120

Модуль № 2 121
ОСНОВЫ ТРЕХМЕРНОЙ ГРАФИКИ 121
Комплексная цель 121
Содержание модуля 121
ГЕОМЕТРИЯ ПРОСТРАНСТВА 121
1.1. Трехмерные однородные координаты и преобразования точки в пространстве 121
1.2. Уравнение прямой и плоскости в пространстве 126
1.3. Изображения трехмерных объектов 127
1.4. Параллельные проекции 129
1.5. Центральные проекции 134
2. ТРЕХМЕРНОЕ ОТСЕЧЕНИЕ 136
2.1. Формы отсекающих объемов 136
2.2. Условия полной видимости/невидимости отрезков 137
2.3. Трехмерное отсечение 139
2.4. Определение выпуклости и разбиение невыпуклых объемов 140
3. УДАЛЕНИЕ НЕВИДИМЫХ ЛИНИЙ И ПОВЕРХНОСТЕЙ 140
3.1. Постановка задачи удаления невидимых линий и поверхностей 140
3.2. Алгоритм плавающего горизонта 142
3.3. Алгоритм Варнока 144
3.4. Алгоритм Робертса 148
3.4.1. Постановка задачи 148
3.4.2. Положение точки относительно тела 149
3.4.3. Проверка корректности описания тела 150
3.4.4. Видовые преобразования 151
3.4.5. Определение нелицевых граней 152
3.4.6. Определение интенсивности закраски 153
3.4.7. Нелицевые отрезки 153
3.4.8. Экранизация отрезков другими телами 154
3.4.10. Определение полностью видимых отрезков 161
3.5. Алгоритм Вейлера-Азертона 162
3.6. Алгоритм разбиения криволинейных поверхностей 164
3.7. Алгоритм, использующий z-буфер 164
3.8. Алгоритм, использующий список приоритетов 166
3.9. Алгоритмы построчного сканирования 168
4. РЕАЛИСТИЧЕСКОЕ ИЗОБРАЖЕНИЕ ОБЪЕКТОВ 170
4.1. Понятие реалистического изображения 170
4.2. Простая модель освещенности 171
4.3. Закраска методами Гуро и Фонга 174
Проектное задание 177
ТЕСТ РУБЕЖНОГО КОНТРОЛЯ №1 178
Бланк ответов 179
ДРУГИЕ ВИДЫ ТЕСТИРОВАНИЯ ПО МОДУЛЮ 180
Выполнение лабораторных работ 180

Выполнение курсовой работы 180
СПИСОК ЛИТЕРАТУРЫ 181
ТОЛКОВЫЙ СЛОВАРЬ 183
СПИСОК СОКРАЩЕНИЙ 187