Министерство образования Республики Беларусь

Учреждение образования
“Белорусский государственный университет информатики и радиоэлектроники”

Кафедра ЭВМ

Садыхов Р.Х., Старовойтов В.В., Конопелько В.Г., Юрас А.В.

Лабораторный практикум
по курсу «Машинная графика»
для студентов специальности Т10.03.00

Минск 2002

УДК 681.3:51 (076)

ББК 32.973-018я7
Л12

Садыхов Р.Х.

Л12 Лабораторный практикум по курсу “Машинная графика” для студентов специальности Т10.03.00 / Р.Х.Садыхов, В.В.Старовойтов, В.Г. Конопелько, А.В.Юрас. – Мн.: БГУИР, 2002. – 52 с.: ил., табл.
ISBN 985-444-

В настоящем учебном пособии рассматривается методы и алгоритмы машинной графики. Особое внимание уделяется оптимизации вычислений. Рассматриваются алгоритмы построения отрезков и окружностей, алгоритмы отсечения невидимых линий и плоскостей, а также подробно рассмотрены алгоритмы заполнения областей.

УДК 681.3:51 (076)
ББК 32.973-018я7

ISBN 985-444-

Введение

Машинная графика – это направление информатики, связанное с визуализацией изображений. Она используется во многих научных и инженерных дисциплинах, в бизнесе и кинематографии, в рекламном и издательском деле, в проектировании. Современная вычислительная техника активно использует элементы машинной графики для более комфортной работы пользователя, существует множество программных продуктов, использующих алгоритмы машинной графики. При этом некоторые алгоритмы были разработаны десятки лет назад, но не потеряли своей актуальности.

Формирование изображения выполняется в три этапа:

- построение модели объекта (описание его элементов и их связей в рамках евклидовой геометрии),
- подготовка модели к визуализации (выполнение геометрических преобразований, удаление невидимых линий),
- визуализация (отсечение окном, формирование растрового представления, вывод на терминал).

Изображение можно представить в виде множества точек, линий, строк текста и закрашенных областей. Для описания многих объектов используют векторные модели. При этом изображение трактуют в виде набора многоугольников, заданных своими ребрами, а они, в свою очередь, задаются координатами вершин. Например, квадрат можно описать четырьмя ребрами E1, E2, E3, E4. Каждое ребро задается своими вершинами E1=P1P2, E2=P2P3, E3=P3P4, E4=P4P1. Каждая вершина задается своими координатами Pi=(xi,yi). При описании трехмерных объектов используются три координаты точки - Pi=(xi,yi,zi). Следует отметить, что визуализация изображений, как правило, выполняется на плоскости, т.е. она двумерна.

В зависимости от типов терминалов, на которые изображения выводятся, они делятся на растровые и векторные. Понятие «векторное изображение»

можно интерпретировать как векторную модель, либо как изображение, формируемое устройством векторного типа, например, графопостроителем, который чертит изображение с помощью перьев разной толщины и цвета.

В растровых устройствах отображения точку заменяет пиксель (от английского picture element). Пиксель – это микрообласть изображения, чаще всего имеющая прямоугольную форму. Растровые изображения имеют ряд особенностей. При визуализации таких изображений следует иметь в виду, что многие понятия и результаты евклидовой геометрии требуют переосмысления. Например, отрезок, заданный двумя вершинами и имеющий бесконечное множество точек в евклидовой геометрии, пикселей в растровом представлении всегда состоит из конечного множества.

Линия – это связное множество точек. На растровом представлении используют два типа связности – четырехсвязность (соседними считаются только смежные по вертикали и горизонтали пиксели) и восьмисвязность (дополнительно соседями считаются диагонально смежные пиксели). Растровое представление линии – это связное множество пикселей, центры которых минимально отклоняются от непрерывного представления этой линии. При этом длина линии, как правило, измеряется количеством пикселей ее растрового представления.

Основные типы растровых изображений - это бинарные (используются только черный и белый цвета, 1 бит/пиксель), двухцветные (например, желтый и темно-серый, 1 бит/пиксель), полутоновые (до 256 оттенков от черного до белого, 8 бит/пиксель), цветные (совокупность трех базовых цветов, каждый из которых трактуется как полутоновое изображение, 24 бит/пиксель). Каждый пиксель задается своими пространственными координатами и значением яркости согласно используемому типу изображения.

Лабораторная работа 1.
Алгоритм Бризенхема для построения отрезка

Цель работы: освоить построение оценочных функций для формирования растрового представления произвольного отрезка прямой.

1.1. Постановка задачи

Дано: описание отрезка прямой, заданное координатами вершин (x_1,y_1) и (x_2,y_2).

Требуется: Сформировать растровое представление отрезка.

1.2. Теоретические основы

Рассмотрим уравнение идеальной прямой, проходящей через две точки,

$$y = y_1 + (x-x_1) *(y_2-y_1)/(x_2-x_1) = y_1 + (x-x_1)*\Delta y/\Delta x,$$

где Δy – разность y-координат концов отрезка, Δx – разность x-координат. Если x и y изменяются вдоль прямой дискретно на δx и δy, тогда $y_i+1=y_i+\delta y = y_i + \delta x*\Delta y/\Delta x$. Пусть δx и/или δy равняется величине пикселя, т.е. 1.

Растровое представление отрезка прямой – это связное множество пикселей, имеющих наименьшее отклонение от идеальной прямой. При этом может использоваться 4- и 8-связность. Точное растровое представление (4- и 8-связное) можно построить только для вертикального и горизонтального отрезка. Для них $\delta x=1$, $\delta y=0$, либо $\delta x=0$, $\delta y=1$. Для отрезка, расположенного под углом 45° (135°) к горизонтальной оси, можно построить точное 8-связное растровое представление. Для него $\delta x=\delta y=1$. Растровое представление всех остальных отрезков выглядит в виде ступенчатой последовательности пикселей (рис.1.1).

В 1965г. Бризенхем (Bresenham) опубликовал целочисленный алгоритм формирования растрового представления произвольного отрезка прямой. Пусть для растрового представления отрезка выбран некоторый пиксель, например имеющий координаты (x_1,y_1). Как выбрать следующий соседний пиксель? Алгоритм использует понятие функции, оценивающей величину отклонения выбираемого пикселя от идеальной прямой. Пусть $0<\Delta y<\Delta x$, тогда фиксируем $\delta x=1$ (т. е. $x_i=x_i+1$) и оцениваем пиксель с какой у-координатой ($y_i+1=y_1$ или $y_i+1=y_1+1$) расположен ближе к идеальной прямой. Выбирается тот пиксель, который имеет меньшее (из двух) значение отклонения, т.е. оценочной функции. При этом отклонение ближайшего пикселя не превышает 0.5 пикселя.

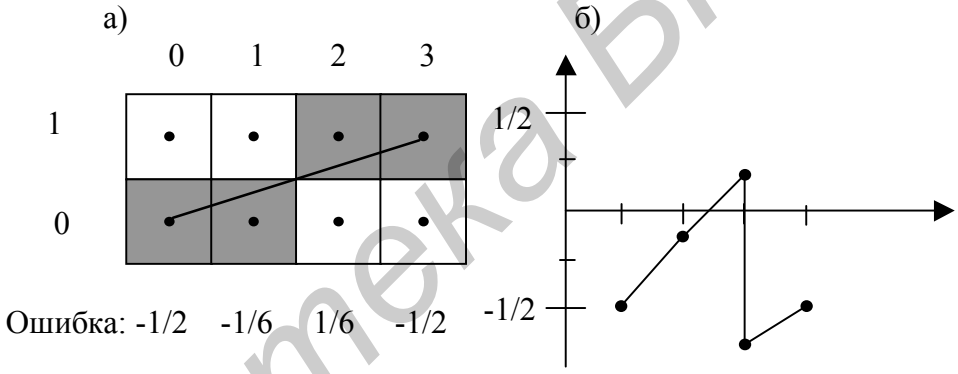

Рис.1.1. а) – идеальный отрезок с вершинами (0,0) и (3,1), его растровое представление. б) – график функции отклонения растрового представления от идеальной прямой

Основа алгоритма – это движение вдоль основной оси на один пиксель и поддержание текущего отклонения от идеальной прямой в заданных пределах. Если текущее отклонение превышает норму, то делается шаг по неосновной оси, чтобы уменьшить отклонение. Идея алгоритма основывается на использовании понятия «ошибки». «Ошибкой» здесь называется расстояние между действительным положением идеального отрезка и «центром» ближайшего пикселя, который аппроксимирует отрезок на очередном шаге. Алгоритм реализован

так, что одна координата изменяется на единицу, а другая – либо не изменяется, либо изменяется на единицу (рис. 1.1).

Кроме того, для вычисления второй координаты требуется только определять знак этой «ошибки», и в зависимости от полученного значения выбирается пиксель, ближе распложенный к идеальному отрезку. Для этого значение «ошибки» смещается на – 0,5.

Пусть f – значение «ошибки» на очередном i-м шаге, $y_и$ – ордината идеального отрезка, а y_i – ордината пикселя, выбранного для аппроксимации отрезка на том же i-м шаге, $m = \Delta y/\Delta x$ – это тангенс угла наклона отрезка.

Поскольку на первом шаге высвечивается пиксель с начальными координатами, то для него f = 0, поэтому задаваемое предварительно значение

$$f = m - 0{,}5 \qquad (1.1)$$

является фактически ошибкой для следующего шага. Приведем основные расчетные соотношения, используемые в алгоритме.

Основной выбирается та ось, по которой приращение по модулю большее. Приращение принимается равным шагу растра, то есть единице, причем знак приращения совпадает со знаком разности начальной и конечной координат отрезка. Значение другой координаты для следующего шага определяется как y = y + m, поскольку приращение ординаты совпадает с величиной одного катета прямоугольного треугольника, а другой катет равен шагу сетки растра, то есть единице.

Ошибка на очередном шаге вычисляется как

$$f_{i+1} = f_i + m, \text{ если } y_{i+1} = y_i,$$

а так как в алгоритме нет необходимости хранить значения ошибок для всех шагов, то вычисление ошибки можно записать $f = f + m$.

В зависимости от полученного значения ошибки выбирается пиксель с той же ординатой (при $f < 0$) или с ординатой на единицу большей, чем у предыдущего пикселя (при $f > 0$). Во втором случае еще надо скорректировать значение ошибки: $f = f – 1$, для того, чтобы значение ошибки снова стало отрицательным.

Очевидно, что такой алгоритм использует арифметические операции с вещественными числами (для определения углового коэффициента и оценки ошибки). Быстродействие алгоритма можно увеличить, если использовать целочисленную арифметику и исключить деление. Для этого выражение (1) надо записать в виде

$$f = \Delta y/\Delta x – 0{,}5$$

и умножить обе части этого равенства на $2\Delta x$, тогда

$$2\Delta x*f = 2\Delta y – \Delta x$$

Обозначив $F = 2\Delta x*f$, окончательно получим

$$F = 2\Delta y – \Delta x$$

Теперь по этой формуле должно вычисляться начальное значение ошибки.

А значение ошибки в цикле прорисовки отрезка: $F = F + 2\Delta y$ или $F = F + 2\Delta x$.

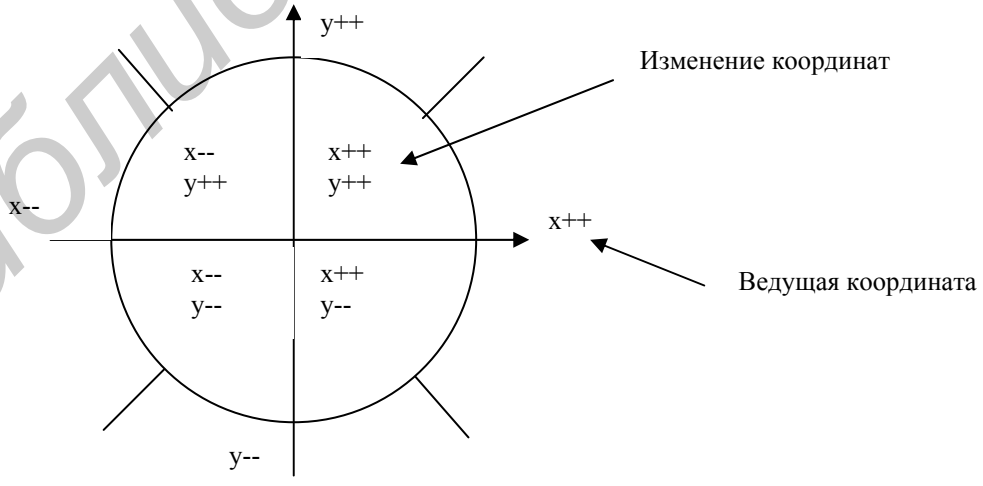

Рис. 1.2. Ведущие координаты и их изменения для всех октантов

Алгоритм Бризенхема, построенный с учетом целочисленных операций, приведенных выше, можно записать в следующем виде:

1. Начало алгоритма прорисовки линии.

2. Ввод исходный данных.

x_1, y_1 – координаты начала отрезка;

x_2, y_2 – координаты конца отрезка.

3. Вычисление приращений $\Delta x = x_2 - x_1$ и $\Delta y = y_2 - y_1$.

4. Вычисление шага изменения каждой координаты пикселя.

если $\Delta x = 0$, то $sx = 0$;

если $\Delta y = 0$, то $sy = 0$;

если $\Delta x < 0$, то $sx = -1$;

иначе $sx = 1$;

если $\Delta y < 0$, то $sy = -1$;

иначе $sy = 1$;

(знаки полученных значений шага сохраняются, это позволяет не ограничиваться первым октаном, когда $\Delta x > 0$ и $\Delta y > 0$)

5. Вычисление модулей приращения координат

$\Delta x = |\Delta x|$, $\Delta y = |\Delta y|$;

6. Вычисление $m = \Delta y - \Delta x$.

7. Анализ вычисленного значения m и изменение ведущей оси с x на y при $m>0$:

если $m > 0$, то

$w = \Delta x$; $\Delta x = \Delta y$; $\Delta y = w$; $flag = 1$;

иначе $flag = 0$;

(flag – флаг, определяющий факт обмена местами координат).

8. Инициализация начального значения ошибки $F = 2\Delta y - \Delta x$.

9. Инициализация начальных значений координат текущего пикселя

$x = x_1$; $y = y_1$.

10. Цикл прорисовки линии.

пока $x \le x_2$ выполнять:

10.1. Нарисовать пиксель с координатами (x , y);

10.2. Вычисление координат следующего пикселя и значения ошибки (F) для него:

если $F \geq 0$, то

$x = x + sx$; $y = y + sy$;

корректировка ошибки $F = F - 2\Delta x$;

иначе

если flag = 0 (основная ось x), то $x = x + sx$;

иначе $y = y + sy$;

корректировка ошибки $F = F + 2\Delta y$

10.3. Конец цикла прорисовки линии

11. Конец алгоритма прорисовки линии.

1.3. Задания к лабораторной работе

1.3.1. Разработать программу отображения многоугольника, заданного своими вершинами с помощью мыши (датчика случайных чисел).

1.3.2. Разработать программу отображения движущейся в заданном окне многоугольной фигуры (автомобиля, “прыгающего квадрата“ и т.п.).

1.3.3. Разработать программу отображения правильного N-угольника, моделирующего растровое представление окружности радиуса R с точностью до одного пикселя.

1.3.4. Разработать программу отображения правильной звезды плавно изменяющегося размера.

1.3.5. Разработать программу изображения ливня (падающие под углом струи), наполняющего водоем.

1.4. Контрольные вопросы

1.4.1. В чем состоит идея алгоритма Бризенхема для построения растрового представления отрезка?

1.4.2. Каковы достоинства алгоритма Бризенхема? Как можно улучшить алгоритм?

1.4.3. Всегда ли совпадают растровые представления отрезков, заданных координатами $(x_1, y_1) – (x_2, y_2)$ и $(x_2, y_2) – (x_1, y_1)$?

1.4.4. Будут ли отличаться растровые представления отрезков $(x_1, y_1) – (x_2, y_2)$ и

а) $(x_1+e, y_1) – (x_2+e, y_2)$,

б) $(x_1+e, y_1+e) – (x_2+e, y_2+e)$,

в) $(x_1, y_1-e)- (x_2, y_2-e)$,

где e – константа $0<e<1$?

1.4.5. Накапливается ли ошибка в процессе выполнения алгоритма Бризенхема?

1.4.6. С какой точностью можно восстановить уравнение непрерывной прямой по растровому представлению отрезка?

1.4.7. Сколько дискретных прямых может проходить через два пикселя?

Литература

Д. Роджерс, Алгоритмы машинной графики.– М.: Мир, 1989, §2.1-2.5.

Дж. Фоли, А. вэн Дэм, Основы интерактивной машинной графики. – М.: Мир, 1985, том 2, §11.2.

Лабораторная работа 2.
Алгоритм Бризенхема для построения окружности

Цель работы: освоить построение оценочных функций для формирования растрового представления окружности произвольного радиуса.

2.1. Постановка задачи

Дано: описание окружности, заданное координатами центра (x_1, y_1) и радиусом R.

Требуется: Сформировать растровое представление окружности.

2.2. Теоретические основы

Окружность задается уравнением:

$$X^2 + Y^2 = R^2 \tag{2.1}$$

$$Y = \pm\sqrt{R^2 - X^2} \tag{2.2}$$

Использование выражения (2. 2) для вычисления координаты Y неэффективно, так как приходится использовать арифметику с плавающей запятой и извлечение квадратного корня. Для повышения эффективности следует использовать только целочисленной арифметику и устранить трудоемкие вычислительные операции. Это достигается в алгоритме Бризенхема.

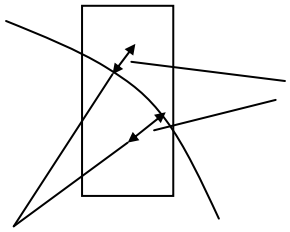

Рис. 2.1. «Ошибка» при формировании растрового представления окружности

Для прорисовки окружности достаточно вычислить только 1/8 ее часть. Остальные получаются зеркальным отражением.

Алгоритм Бризенхема вычисляет часть окружности, лежащую в первом квадранте. В этом алгоритме используется понятие «ошибки» – расстояния от центра пикселя до действительного положения окружности (рис. 2.1).

Вычисление координат очередного пикселя базируется на координатах ранее вычисленного пикселя. Пусть вычисленный пиксель имел координаты (x_i, y_i). Тогда «кандидатами» на прорисовку будут (рис. 2.2).

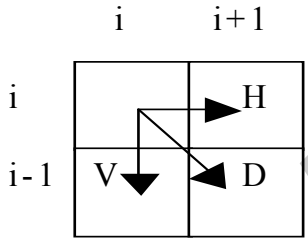

Рис. 2.2. «Кандидаты» на прорисовку на следующей итерации

Необходимо рассчитать ошибку для пикселей H, D, V и выбрать тот, у которого ошибка меньше. При вычислениях используется ошибка для ранее прорисованного пикселя. Она вычисляется:

$$\Delta i = X_i^2 + Y_i^2 - R^2 \qquad (2.3.)$$

Рассчитаем значение ошибки для H.

$$\Delta_H = \Delta_i + (\Delta_H - \Delta_i)$$

Аналогично для D и V:

$$\Delta_V - \Delta_i = y_{i+1}^2 - y_i^2 = -2 \times y_i + 1$$

$$\Delta_D - \Delta_i = 2 \times (x_i - y_i + 1)$$

Имеется четыре варианта прохождения окружности (рис. 2.3).

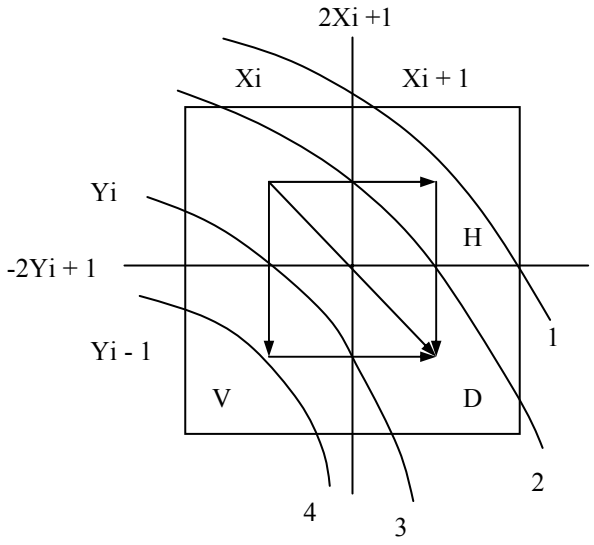

Рис. 2.3. Варианты прорисовки окружности

Проанализируем знак у Δi. Если Δi > 0, то центр пикселя вне окружности и рассматривается случай 3 и 4. Если Δi < 0, то – случаи 1 и 2.

Пусть Δi < 0, следовательно, анализируем ΔH.

Если ΔH < 0, то это соответствует варианту прохождения окружности №1. Необходимо прорисовывать пиксель H. В противном случае положение окружности соответствует варианту №2.

Нужно решить, какой из пикселей: H или D прорисовывать. Для этого вычисляется ΔD и сравнивается с ΔH.

Алгоритм Бризенхема, построенный с учетом целочисленных операций можно записать в следующем виде:

1. Начало алгоритма прорисовки окружности.

2. Ввод исходный данных.

x_1, y_1 – координаты центра окружности;

R – радиус окружности.

3. Инициализация начальных значений координат текущего пикселя, текущей ошибки (d):

x = 0; y = R; d = 0.

4. Цикл прорисовки окружности .

Пока y ≥ x выполнять:

4.1. Нарисовать пиксели с координатами:

(x_1+x, y_1+y); (x_1+y, y_1+x);

(x_1-x, y_1+y); (x_1-y, y_1+x);

(x_1+x, y_1-y); (x_1+y, y_1-x);

(x_1-x, y_1-y); (x_1-y, y_1-x).

4.2. Вычисление координат следующего пикселя и значения ошибки:

если $d \le 0$, то окружность проходит по варианту 1 или 2 (рис. 2.3.):

Вычисляем ошибку для H: d = d + (x << 1) + 1;

если d < 0, то окружность проходит по варианту 1:

x = x + 1.

иначе окружность проходит по варианту 2:

Вычисляем ошибку для D: dw = d + 1 – (y << 1);

x = x + 1;

Если dw < –d, то рисуем D:

d = dw; y = y – 1.

иначе окружность проходит по варианту 3 или 4:

Вычисляем ошибку для V: d = d + 1 – (y << 1);

если d > 0, то окружность проходит по варианту 4:

y = y – 1.

иначе окружность проходит по варианту 3:

Вычисляем ошибку для D: dw = d + 1 + (x << 1);

y = y – 1;

Если dw > –d, то рисуем D:

d = dw; x = x + 1.

4.3. Конец цикла прорисовки линии

11. Конец алгоритма прорисовки линии.

Как видно из вышеприведенного алгоритма, вычисляется положение пикселей 1/8 части окружности (для $y \geq x \geq 0$), координаты остальных пикселей находятся зеркальным отражением полученных пикселей.

2.3. Задания к лабораторной работе

2.3.1. Разработать программу отображения набора окружностей, заданных центрами и радиусами с помощью мыши, датчика случайных чисел.

2.3.2. Разработать программу отображения движущейся в заданном окне окружности (“прыгающего мяча“).

2.3.3. Разработать программу отображения эллипса (перекатывающегося эллипса) с помощью оценочной функции.

2.3.4. Разработать программу отображения параболы с помощью оценочной функции.

2.3.5. Разработать программу отображения движущегося автомобиля (паровоза) с крутящимися колесами (путем изображения “спиц” на колесах).

2.3.6. Разработать программу изображения ливня (падающие под углом капли разных размеров в форме перевернутых парашютов), наполняющего водоем.

2.3.7. Разработать программу отображения дуги окружности, задав необходимые параметры с помощью мыши.

2.4. Контрольные вопросы

2.4.1. В чем состоит идея алгоритма Бризенхема для построения растрового представления окружности?

2.4.2. Каковы достоинства алгоритма Бризенхема? Как можно улучшить алгоритм?

2.4.3. Всегда ли совпадают растровые представления окружностей, заданных координатами $(x_1, y_1) – (x_2, y_2)$ и $(x_2, y_2) – (x_1, y_1)$?

2.4.4. Что проще в вычислительном плане? Сформировать растровое представление окружности с помощью алгоритма Бризенхема для вычерчивания отрезка путем отображения правильного N-угольника, аппроксимирующего окружность с нужной точностью, или алгоритма Бризенхема для построения растрового представления окружности?

2.4.5. Будут ли отличаться растровые представления окружностей, если

а) их радиусы отличаются на e,

б) координаты их центров отличаются на e,

где e – константа $0<e<1$?

2.4.6. Накапливается ли ошибка в процессе выполнения алгоритма Бризенхема?

2.4.7. С какой точностью можно восстановить уравнение непрерывной окружности по ее растровому представлению?

2.4.8. Сколько пикселей на растровом представлении окружности требуется для вычисления ее радиуса? Какова точность вычисления радиуса?

Литература

Д. Роджерс, Алгоритмы машинной графики. – М.: Мир, 1989, §2.6.

Дж. Фоли, А. вэн Дэм, Основы интерактивной машинной графики. – М.: Мир, 1985, том 2, §11.4.

Лабораторная работа 3.
Алгоритмы заполнения области

Цель работы: освоить алгоритмы формирования растрового представления многоугольных областей.

3.1. Постановка задачи

Дано: описание многоугольной области, заданное координатами последовательности вершин.

Требуется: сформировать растровое представление области.

3.2. Теоретические основы

Алгоритмы заполнения (закраски) областей делятся на группы: растровой развертки и затравочного заполнения. В методах растровой развертки область представляет собой многоугольник с заданными координатами концов, ограничивающих его отрезков. В этом методе находится пересечение строк с многоугольником, определяются пары точек, ограничивающих закрашиваемый отрезок, и на основе этого осуществляется закраска.

В методах затравочного заполнения область задается граничными пикселями, которые окрашены в определенный цвет. Кроме того, задается некоторый пиксель внутри области – затравочный. Затем ищутся пиксели, соседние с затравочным, и закрашиваются.

По способу задания области делятся на два типа:

- гранично-определенные, задаваемые своей (замкнутой) границей такой, что коды пикселей границы отличны от кодов внутренней, перекрашиваемой части области. На коды пиксели внутренней части области налагаются два условия - они должны быть отличны от кода пикселей границы и кода пикселя перекраски. Если внутри гранично-определенной области имеется еще одна граница, нарисованная пиксе-

лями с тем же кодом, что и внешняя граница, то соответствующая часть области не должна перекрашиваться;

- внутренне-определенные, нарисованные одним определенным кодом пикселя. При заполнении этот код заменяется на новый код закраски.

Заполняемая область или ее граница - некоторое связное множество пикселей. По способам доступа к соседним пикселям области делятся на 4-х и 8-ми связные. В 4-х связных областях доступ к соседним пикселям осуществляется в четырех направлениях - горизонтально влево и вправо и вертикально вверх и вниз. В 8-ми связных областях к этим направлениям добавляются еще 4 диагональных. Используя тот или иной тип связности, мы может, двигаясь от точки затравки, закрасить все пиксели области.

Важно отметить, что для 4-х связной прямоугольной области граница 8-ми связна (рис. 3.1.а) и наоборот у 8-ми связной области граница 4-х связна (рис. 3.1.б). Поэтому заполнение 4-х связной области 8-ми связным алгоритмом может привести к "просачиванию" через границу и закраске пикселей в примыкающей области.

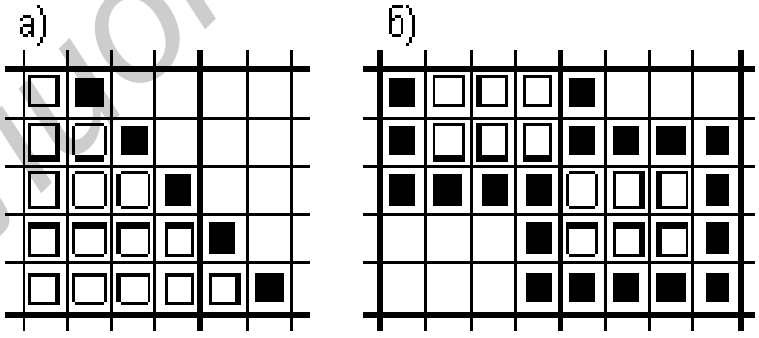

Рис. 3.1. Связность областей и их границ(а) 4-х б) 8-ми внутренне - определенные связные области)

В общем, 4-х связную область мы можем заполнить как 4-х, так и 8-ми связным алгоритмом. Обратное же неверно. Так область на рис. 3. 1. а мы можем заполнить любым алгоритмом, а область на рис. 3.1. б, состоящую из двух

примыкающих 4-х связных областей можно заполнить только 8-ми связным алгоритмом.

3.1.1. Тривиальный алгоритм заполнения с затравкой

В алгоритме используется стек. В него заносится затравочный пиксель. Логика работы алгоритма закраски 4-х связной гранично-определенной области следующая:

1. Начало алгоритма заполнения с затравкой.
2. Поместить координаты затравочного пикселя в стек.
3. Начало цикла заполнения области:

   Пока стек не пуст:

   3. 1. Извлечь координаты пикселя из стека.

   3. 2. Перекрасить пиксель.

   3. Для всех четырех соседних пикселей проверить: является ли он граничным или уже перекрашен. Если не перекрашен, то занести его координаты в стек.
4. Конец цикла заполнения.
5. Конец алгоритма заполнения.

Недостатки: Требуется очень большой стек. В стеке запоминаются пиксели, дублирующие друг друга.

Пример работы затравочного алгоритма. Координаты затравочного пикселя – (2, 2):

Стек. Шаг 1. (2, 2)

Шаг 2. (2, 3) (3, 2) (2, 1) (1, 2)

Шаг 3. (2, 3) (3, 2) (2, 1) (1, 3) (1, 1)

Шаг 4. (2, 3) (3, 2) (2, 1) (1, 3) (2, 1)

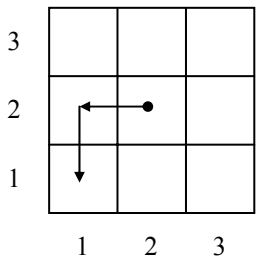

Рис. 3. 2. Пример работы алгоритма заполнения затравкой.

Как уже отмечалось, очевидный недостаток алгоритмов непосредственно использующих связность закрашиваемой области - большие затраты памяти на стек, так как на каждый закрашенный пиксель в стеке по максимуму будет занесена информация о еще трех соседних. Кроме того, информация о некоторых пикселях может записываться в стек многократно. Это приведет не только к перерасходу памяти, но и потере быстродействия за счет многократной раскраски одного и того же пикселя. Значительно более экономен далее рассмотренный построчный алгоритм заполнения.

3.1.2. Построчный алгоритм заполнения с затравкой

Использует пространственную когерентность:

- пиксели в строке меняются только на границах;
- при перемещении к следующей строке размер заполняемой строки скорее всего или неизменен или меняется на 1 пиксель.

Таким образом, в этом алгоритме стек минимизируется за счет хранения только одного затравочного пикселя для непрерывного интервала на строке.

Последовательность работы алгоритма для гранично-определенной области следующая:

1. Начало алгоритма заполнения с затравкой.
2. Координаты затравочной точки помещается в стек.
3. Начало цикла заполнения области:

   Пока стек не пуст:

   3. 1. Координата очередной затравки извлекается из стека и выполняется максимально возможное закрашивание вправо и влево по строке с затравкой, т.е. пока не попадется граничный пиксель. Пусть это $X_{лев}$ и $X_{прав}$, соответственно.

   3. 2. Анализируется строка ниже закрашиваемой в пределах от $X_{лев}$ до $X_{прав}$ и в ней находятся крайние правые пиксели всех незакрашенных фрагментов. Их координаты заносятся в стек.

3. 3. То же самое проделывается для строки выше закрашиваемой.

4. Конец цикла заполнения.
5. Конец алгоритма заполнения.

Рассмотрим пример работы алгоритма. Координаты затравочного пикселя (5,5):

Стек: Шаг 1. (5,5)

Шаг 2. (7,6) (3,4) (7,4)

Шаг 3. (7,6) (3,4) (7,3)

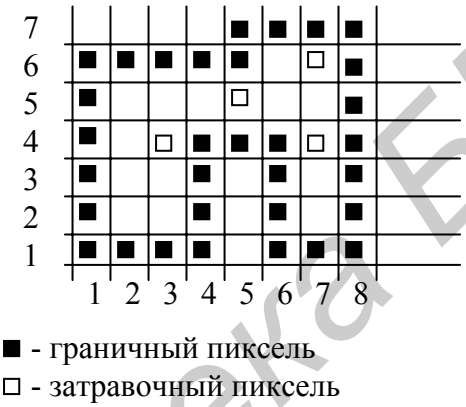

Рис. 3.3. Пример работы построчного алгоритма заполнения с затравкой

3.3. Задания к лабораторной работе

3.3.1. Разработать программу закрашивания выпуклого N-угольника алгоритмом заполнения с затравкой .

3.3.2. Изобразить вращающийся закрашенный прямоугольник построчным алгоритмом заполнения с затравкой.

3.3.3. Закрасить область, описанную самопересекающимся контуром с 20 вершинами и более одной точек самопересечения обоими алгоритмами. (Дать рисунок контура).

3.4. Контрольные вопросы

3.4.1. В чем состоит идея тривиального алгоритма заполнения с затравкой для построения растрового представления области? Каковы достоинства и недостатки алгоритма?

3.4.2. В чем состоит идея построчного алгоритма заполнения с затравкой для построения растрового представления области? Каковы достоинства и недостатки алгоритма?

3.4.3. Какие проблемы возникают при заполнении самопересекающегося контура? Как они разрешаются?

3.4.4. Можно ли закрасить незамкнутый контур? Если да, то как?

3.4.5. Как закрасить круг радиуса R (кольцо радиусов R_1, R_2)?

3.4.6. Каким методом лучше закрашивать треугольник с координатами (x,y), (x + c, y + c), (x + 2c, y). где c>1, x, y – любые числа?

3.4.7. Требуется закрасить n- угольник. Какой метод а) проще, б) быстрее, в) выполнит корректное заполнение независимо от формы и особенностей контура?

3.4.8. Каким алгоритмом проще заполнить многоугольник, имеющий только горизонтальные и вертикальные стороны?

Литература

Д. Роджерс, Алгоритмы машинной графики. – М.: Мир, 1989, §2.15–2.24.

Дж. Фоли, А. вэн Дэм, Основы интерактивной машинной графики. – М.: Мир, 1985, том 2, §11.5-11.7.

Т. Павлидис, Алгоритмы машинной графики и обработки изображений. – М.: Радио и связь, 1990, глава 8.

Лабораторная работа 4.
Алгоритмы отсечения областей

Цель работы: освоить алгоритмы формирования растрового представления многоугольных областей, отсекаемых прямоугольным окном и окном в форме произвольного многоугольника.

4.1. Постановка задачи

Дано: описание множества отрезков, заданных координатами вершин $[(xi_1,yi_1), (xi_2,yi_2)]$ и окно видимости, заданное в виде многоугольника.

Требуется: Сформировать растровое представление тех отрезков или их частей, которые видны в заданном окне.

4.2. Теоретические основы

4.1.1. Двумерный алгоритм Коэна - Сазерленда

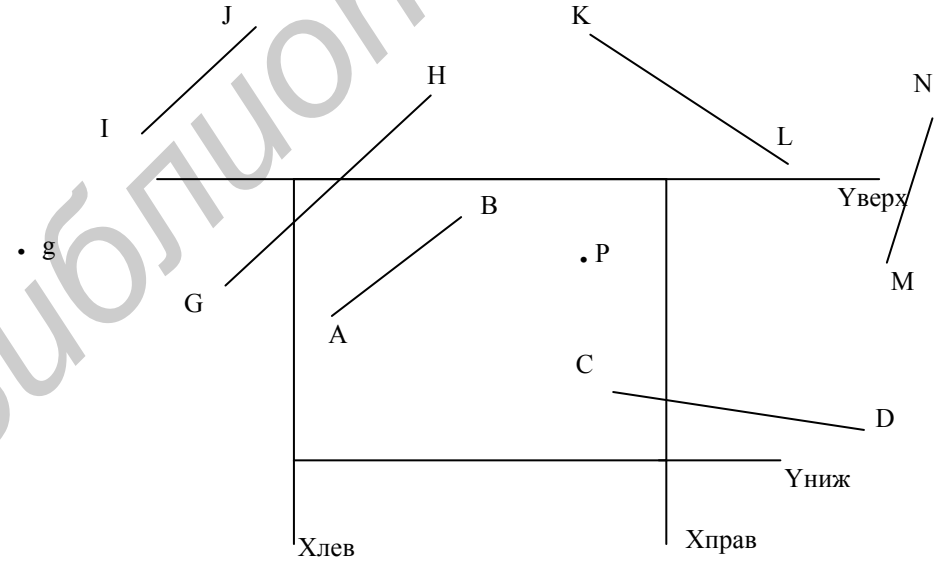

Рис. 4.1. Примеры взаимного расположения отрезков и прямоугольного окна

Целью отсечения является нахождение тех областей, отрезков, точек, которые лежат внутри окна (рис. 4.1).

В реальных сценах приходится отсекать большое количество отрезков и точек. Поэтому основное требование к алгоритму – эффективность. Проверкой того, что точка находится внутри окна, является:

$$X_{прав} \le X \le X_{лев}$$

$$Y_{ниж} \le Y \le Y_{верх}$$

Если оба конца отрезка внутри окна, то и отрезок внутри окна. Для отрезков KL, GH и CD необходимо искать координаты пересечения отрезками, ограничивающими окно.

Тесты полной видимости или невидимости можно хорошо адаптировать для аппаратной реализации. Для этого используется 4-х битовая кодировка точки (табл. 4.1).

Таблица 4.1. Четырехбитовая кодировка точки

отрезок	P1	P2	P1&P2
[AB]	0001	0000	0000
[CD]	0100	0110	0100
[GH]	1000	0010	0000
	внпл	внпл	внпл

1) Если коды обоих концов отрезков нулевые, то он целиком лежит в окне.

2) Если побитовое произведение кодов ≠ 0, то отрезок полностью невидим и его можно отбросить.

3) Если побитовое произведение равно 0, то нужны дополнительные проверки.

Пересечение двух отрезков можно считать следующим образом:

1) рассмотрение частных случаев.

2) наклонные отрезки.

$P_1(x_1, y_1)$ $P_2(x_2, y_2)$

Отрезок лежит на прямой, уравнение которой имеет вид:

$y = m (x – x_1) + y_1$, где m – коэффициент наклона.

Точки пересечения этой прямой со сторонами окна будут следующими (табл. 4.2):

Таблица 4.2. Точки пересечения прямой с ребрами окна.

Ребро	X	Y	Условие
с левой	$x_л$	$m(x_л-x_1)+y_1$	$m \neq p$
с правой	$x_п$	$m(x_п-x_1)+y_1$	$m \neq p$
с верхней	$X_1+(1/m)(y_в-y_1)$	$y_в$	$m \neq 0$
с нижней	$X_1+(1/m)(y_н-y_1)$	$y_н$	$m \neq 0$

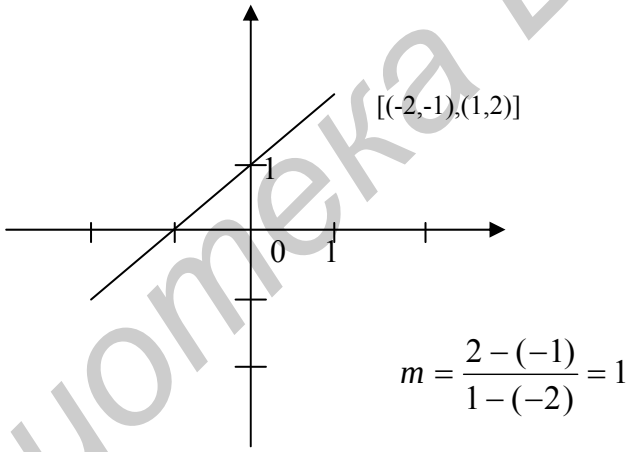

Рис. 4.2. Результат отсечения

Таблица 4.3. Точки пересечения прямой с ребрами окна.

Ребро	X	Y
Левая	-1	0
Правая	1	2
Верхняя	0	1
Нижняя	-2	-1

Результат отсечения [(-1,0),(0,1)]

4.1.2. Алгоритм Кируса-Бека

В алгоритме Кируса-Бека используется вектор нормали для определения местоположения точки относительно окна, т. е. (внутри, на границе, вне окна) для точки, принадлежащей отрезку (рис. 4.3.). Внутренняя нормаль n (все вектора в тексте без стрелок сверху) в произвольной точке a, лежащей на границе окна, удовлетворяет следующему условию: $n\,(b - a) \geq 0$, где a, b – любые другие точки, лежащие на границе области. Поскольку θ лежит в интервале от $[-\pi/2;\pi/2]$, то скалярное произведение векторов всегда ≥ 0.

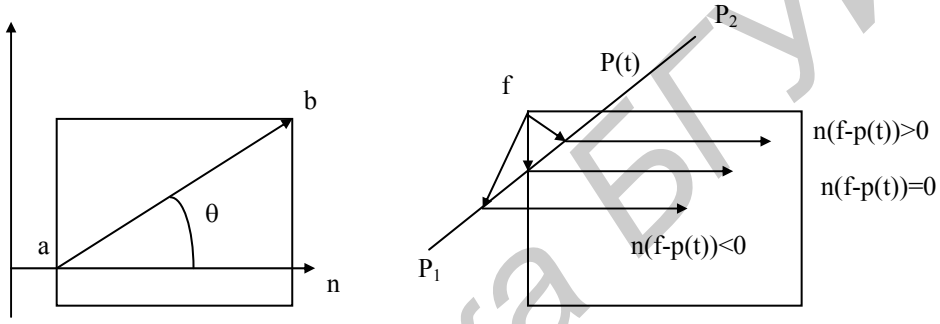

Рис. 4. 3. Отсечение по алгоритму Кируса-Бека.

Пусть задан отрезок P_1P_2. Его параметрическое представление имеет следующий вид:

$$P(t) = P_1 + (P_2 - P_1)\ t,\ 0 \leq t \leq 1$$

Пусть f – граничная точка окна, n – нормаль к одной из двух сторон, примыкающих к f.

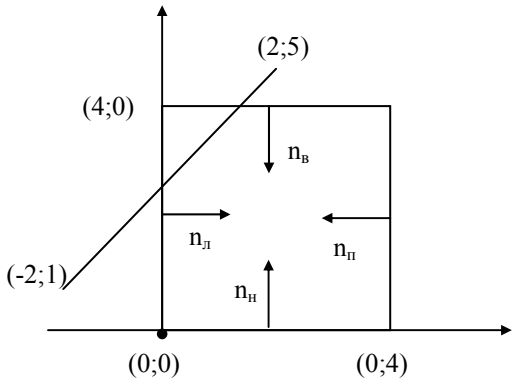

Рис. 4.4. Пример пересечения отрезка и окна

Возможны три случая расположения точки f:

1) P(t) – f лежит вне окна.

2) P(t) – f лежит на ребре

3) P(t) – f лежит внутри области.

Если решить уравнение относительно условия n (f – P(t)) = 0, то получим точку пересечения отрезка с ребром.

$P(t) = p_1 + (p_2 - p_1) \cdot t = [-2; 1] + ([2; 5] - [-2; 1])\, t = [-2; 1] + [4; 4]\, t = (-2 + 4t)i + (1 + 4t)j$,

где i и j – единичные векторы $0 \le t \le 1$.

$n_л = i$; $n_п = -i$; $n_н = j$; $n_в = -j$;

Выбираем точку f с координатами (0; 0), тогда $P(t) - f = (-2 + 4t)i + (1 + 4t)j$

$n_л(P(t) - f) = (-2 - 4t) = 0$, значит t=1/2

$n_н(P(t) - f) = (1 + 4t) = 0$, $t = -1/4$ – отбрасывается

Для верхней и правой границы – f (4, 4), тогда $P(t) - f = (-6 + 4t)i + (-3 + + 4t)j$

$n_в(P(t) - f) = -(-3 + 4t) = 0$, t=3/4

$n_п \cdot (P(t) - f) = -(-6 + 4t) = 0$, t=3/2 – отбрасывается, т.к. $0 \le t \le 1$

Точки пересечения: t=1/2 и t=3/4.

t = 1/2: P(t) = [0; 3] t = 3/4: P(t) = [1; 4]

Рассмотрим другой пример:

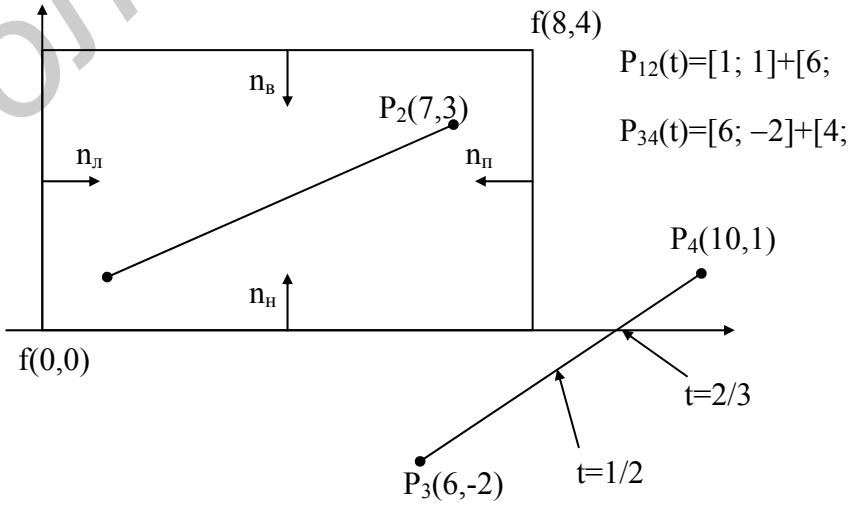

Рис. 4.5. Пример взаимного расположения отрезков и окна

Таблица 4.4. Точки пересечения прямой с ребрами окна.

Ребро	N	f	P(t)-f	n·(P(t)-f)=0	t
Левое	i	(0;0)	(1+6t)·i+(1+2t)·j	1+6t	-1/6
Правое	-i	(8;4)	(-7+6t)·i+(-3+2t)·j	7-6t	7/6
Нижнее	j	(0;0)	(1+6t)·i+(1+2t)·j	1+2t	-1/2
Верхнее	-j	(8;4)	(-1+6t)·i+(-3+2t)j	3-2t	3/2

t не лежит в интервале [0;1], а значит пересечений отрезка с ребрами нет.

Но с другой стороны при t=0 и t=1 все значения выражения $\lambda\cdot(P(t)-f)$ больше нуля, а это означает, что все точки находятся внутри, т.е. отрезок целиком виден.

Рассмотрим отрезок P_3P_4: $P_3(6;-2)$ $P_4(10;1)$

$P(t)=[6;2]+[4;3]\cdot t$

Таблица 4.5. Точки пересечения прямой с ребрами окна.

Ребро	N (P(t) – f)	t
Левая	6 + 4 t	–3/2
Правая	(–2 + 4) t	1/2
Нижняя	–2 + 3 t	2/3
Верхняя	–(–6 + 3 t)	2

Значение t для пересечения с правой и нижней сторонами допустимы. Но учитывая ориентацию отрезка от P_5 к P_6 (для t от 0 до 1), получаем, что такой отрезок не может пересечь правую сторону при t=1/2 прежде чем он пересечет нижнюю при t=2/3 и при этом должен будет иметь с окном общие точки. Т.е. подставляем t=1/2 или t=2/3 в уравнение, получаем координаты и по ним делаем вывод о попадании точки в окно.

Формальная запись алгоритма Кируса-Бека.

Пусть дано уравнение прямой P(t) в параметрической форме:

$$P(t) = P_1 + (P_2 - P_1)\, t.$$

Также даны нормали к сторонам (n_i) и точки, принадлежащие ребрам (f_i). Чтобы найти пересечение со стороной, надо решить следующее уравнение:

$$n_i\,(P(t) - f_i) = 0.$$

Подставляя в это уравнение выражение P(t), получим:

$$n_i\,(P_1 + (P_2 - P_1)\, t - f_i) = 0$$

$$n_i\,(P_1 - f_i) + n_i\,(P_2 - P_1)\, t = 0; \quad D = (P_2 - P_1); \quad W_i = P_1 - f_i;$$

$$t\,(n_i\, D) + W_i\, n_i = 0;$$

$$t = -\,(W_i\ n_i) / (D\ n_i)$$

$D \neq 0$, иначе это либо точка, либо отрезок, параллельный ребру.

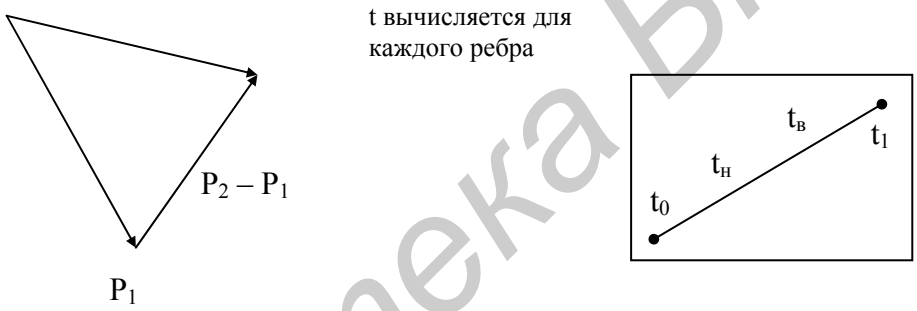

Рис. 4.6. Вычисление точек в алгоритме Кируса-Бека

1. Если t за пределом интервала [0;1], то делается дополнительная проверка: $W_i\ n_i < 0$ и $W_i\ n_i + n_i\ D < 0$; Если это верно, то отрезок считается тривиально.

2. Если t за пределом, то это выражение будет больше 0 и отрезок изображается тривиально.

3. Если t лежит в допустимых пределах, то проверяется t n_i. Если t n_i>0, то найденная t рассматривается в качестве возможного нижнего предела, иначе t рассматривается в качестве возможного верхнего предела.

4. Надо проверить, что $t_н < t_в$. Иначе отрезок невидим. Здесь $t_н$ и $t_в$ – это не значение t пересечения с ребрами, а имеется ввиду, что $t_н$ ближе к t_0, чем $t_в$.

4.3. Задания к лабораторной работе

4.3.1. Разработать программу отсечения выпуклого N-угольника алгоритмом Коэна - Сазерленда.

4.3.2. Разработать программу отсечения невыпуклого N-угольника алгоритмом Кируса - Бека.

4.3.3. Сформировать многоугольный контур некой фигуры (например, автомобиль, самолет и т.п.), показать его серым цветом, передвигать этот контур мимо прямоугольного окна, изображая красным ту его часть, которая попадает в окно. Выполнить это задание при помощи обоих алгоритмов.

4.4. Контрольные вопросы

4.4.1. В чем состоит идея алгоритма Коэна - Сазерленда? Каковы достоинства и недостатки алгоритма?

4.4.2. В чем состоит идея алгоритма Кируса - Бека? Каковы достоинства и недостатки алгоритма?

4.4.3. Какие проблемы могут возникнуть при отсечении и закраске (самопересекающегося) контура? Как они разрешаются?

4.4.4. Каким алгоритмом проще выполнить процедуру отсечения для многоугольника, имеющего только а) горизонтальные и вертикальные стороны, б) стороны, расположенные под углами 30 и 120 градусов?

Литература

Д. Роджерс, Алгоритмы машинной графики. – М.: Мир, 1989, глава3.

Дж. Фоли, А. вэн Дэм, Основы интерактивной машинной графики. – М.: Мир, 1985, том 2, §11.6.

Лабораторная работа 5.
Геометрические преобразования в однородных координатах

Цель работы: научиться выполнять сдвиг, поворот, симметричное отражение, масштабирование и комбинированные преобразования объектов на плоскости.

5.1. Постановка задачи

Дано: Координатное описание объекта.

Требуется выполнить его геометрическое преобразование.

5.2. Теоретические основы

Рассмотрим следующие преобразования: сдвиг, поворот, симметричное отражение и масштабирование на плоскости точки с координатами (x,y). Эти преобразования в матричном представлении имеют следующий вид:

сдвиг:

$$[x',y'] = [x,y] + [T_x,T_y] \text{ или } P' = P + T$$

поворот вокруг начала координат:

$$[x',y'] = [x,y]\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \text{ или } P' = P*R$$

симметричное отражение относительно оси OY:

$$[x',y'] = [x,y]\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \text{ или } P' = P*M$$

масштабирование:

$$[x',y'] = [x,y]\begin{bmatrix} Dx & 0 \\ 0 & Dy \end{bmatrix} \text{ или } P' = P*D$$

Различия в их описании затрудняют формирование единого описания преобразований и их комбинирование.

Пусть M – произвольная точка плоскости с координатами x и y, вычисленными относительно заданной прямолинейной координатной системы. Однородными координатами этой точки называется любая тройка одновременно неравных нулю чисел x1, x2, x3, связанных с заданными числами x и y следующими соотношениями:

$$\frac{x1}{x3} = x\,,\ \frac{x2}{x3} = y\,.$$

При решении задач компьютерной графики однородные координаты обычно вводятся так: произвольной точке M(x,y) плоскости ставится в соответствие точка M(x,y,1) в пространстве (рис. 5.1).

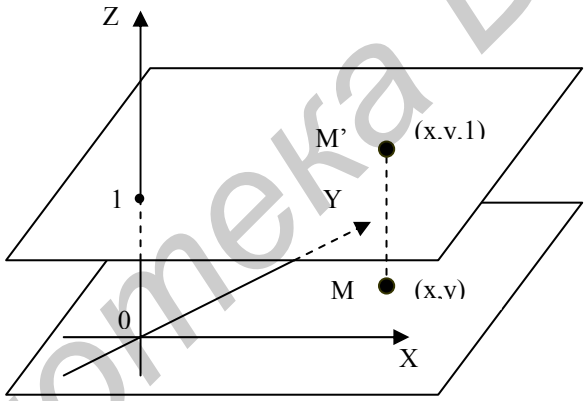

Рис. 5.1. Однородные координаты точки M

Заметим, что произвольная точка на прямой, соединяющей начало координат, точку O(0,0,0), с точкой M(x,y,1), может быть задана тройкой чисел вида (hx,hy,h). Будем считать, что h не равно 0. Вектор с координатами (hx,hy,h) является направляющим вектором прямой, соединяющей точки O(0,0,0) и M(x,y,1).Эта прямая пересекает плоскость Z = 1 в точке (x,y,1), которая однозначно определяет точку (x,y) координатной плоскости xy. Тем самым между произвольной точкой с координатами (x,y) и множеством троек чисел вида (hx,hy,h), h не равно 0, устанавливается взаимно однозначное соответствие, позволяющее считать числа hx,hy,h новыми координатами этой точки.

В проективной геометрии для однородных координат принято обозначение x : y : 1, или , более общее , x1,x2,x3 (непременно требуется, чтобы числа x1,x2,x3 одновременно в нуль не обращались).

Применение однородных координат оказывается удобным уже при решении простейших задач. Рассмотрим, например, вопросы, связанные с изменением масштаба. Если устройство отображения работает только с целыми числами (или если необходимо работать только с целыми числами), то для произвольного значения h (например, h=1) точку с однородными координатами (0.5, 0.1, 2.5) представить нельзя. Однако при разумном выборе h можно добиться того, что бы координаты этой точки были целыми числами. В частности, при h=10 для рассматриваемого примера имеем (5, 1, 25).

Приведенный пример показывает полезность использования однородных координат при проведении расчетов. Однако основной целью введения однородных координат в компьютерной графике является их удобство в применении к геометрическим преобразованиям. При помощи троек однородных координат и матриц третьего порядка можно описать любое аффинное преобразование в плоскости. В самом деле, считая h = 1, рассмотрим следующую матричную запись:

$$(X',Y',1) = (X,Y,1)\begin{bmatrix} \alpha & \gamma & 0 \\ \beta & \delta & 0 \\ \lambda & \mu & 1 \end{bmatrix}$$

Элементы произвольной матрицы аффинного преобразования не несут в себе явно выраженного геометрического смысла. Поэтому что бы реализовать то или иное отображение, т.е. найти элементы соответствующей матрицы по заданному геометрическому описанию, необходимы специальные приемы. Обычно построение этой матрицы в соответствии со сложностью рассматриваемой задачи разбивают на несколько этапов. На каждом этапе ищется матрица, соответствующая тому или иному преобразованию.

Выпишем матрицы, описывающие базовые преобразования на плоскости.

Матрица вращения:

$$[R] = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.1)$$

Матрица масштабирования:

$$[D] = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.2)$$

Матрица отражения:

$$[M] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.3)$$

Матрица сдвига:

$$[T] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \lambda & \mu & 1 \end{bmatrix} \quad (5.4)$$

Рассмотрим пример выполнения более сложного аффинного преобразования плоскости - поворот вокруг точки A(a,b) на угол φ (рис. 5.2). Оно выполняется за три шага с использованием базовых преобразований.

1-й шаг. Перенос на вектор A(-a,-b) для совмещения центра поворота с началом координат :

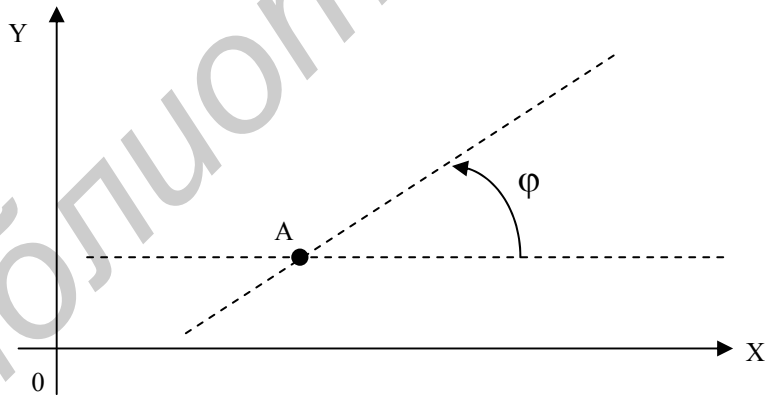

Рис. 5.2. Поворот вокруг точки A’

$[T_{-A}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -a & -b & 1 \end{bmatrix}$ - матрица соответствующего преобразования.

2-й шаг. Поворот на угол φ:

$$[R_\varphi] = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ - матрица соответствующего преобразования.

3-й шаг. Перенос на вектор A(a,b) для возвращения центра поворота в прежнее положение:

$$[T_A] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{bmatrix}$$ - матрица соответствующего преобразования.

Перемножим матрицы в том же порядке, как они выписаны:

$$[T_{-A}][R_\varphi][T_A]. \qquad (5.5)$$

В результате искомое преобразование (в матричной записи) будет выглядеть следующим образом:

$$(X',Y',1) = (X,Y,1)\begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ -a*\cos\varphi + b*\sin\varphi + a & -a*\sin\varphi - b*\cos\varphi + b & 1 \end{bmatrix}$$

Элементы полученной матрицы (особенно в последней строке) не так легко запомнить. В то же время каждая из трех перемножаемых матриц по геометрическому описанию соответствующего отображения легко строится.

Следует иметь в виду, что матрицы в выражении (5.5) в общем случае нельзя переставлять местами или перемножать в произвольном порядке. Описанные преобразования применяются к конечным точкам отрезка, положение промежуточных точек вычисляется путем простейшей линейной аппроксимации. Поэтому при формировании растровых изображений сначала следует вы-

полнить преобразование отрезка, а затем строить его растровое представление с помощью алгоритма Бризенхема.

5.3. Задания к лабораторной работе

5.3.1. Разработать программу поворота многоугольника, заданного своими вершинами с помощью мыши (датчика случайных чисел), вокруг одной из вершин на произвольный угол.

5.3.2. Разработать программу отображения движущейся на экране многоугольной фигуры.

5.3.3. Разработать программу отображения многоугольника, симметрично отражающегося относительно произвольной прямой, проходящей через две точки, задаваемые с помощью мыши.

5.3.4. Разработать программу отображения правильной звезды плавно изменяющегося размера, перемещающейся на экране.

5.3.5. Разработать программу изображения шагающего робота.

5.4. Контрольные вопросы

5.4.1. В чем состоит идея однородных координат?

5.4.2. Каковы достоинства матричного представления преобразований? Как можно ускорить их выполнение?

Литература

Д. Роджерс,Дж. Адамс, Математические основы машинной графики. – М.: Мир, 1980, глава 2.

Дж. Фоли, А. вэн Дэм, Основы интерактивной машинной графики. – М.: Мир, 1985, том 1, глава 7.

Лабораторная работа 6.
Формирование изображений трехмерных объектов.

Цель работы: научиться создавать реалистичные изображения трехмерных объектов, описанных в виде многогранников.

6.1. Постановка задачи

Дано: множество граней трехмерного объекта, заданных координатами вершин и значения интенсивности в каждой вершине.

Требуется сформировать изображение трехмерного объекта.

6.2. Теоретические основы

Для упрощения вычислений модели объектов следует задавать набором выпуклых граней с малым числом вершин. Изображение каждой грани формируется отдельно. Можно использовать три простейших метода рендеринга полигональных моделей, дающих приемлемые результаты, - метод плоского (постоянного) закрашивания, метод Гуро и метод Фонга.

6.2.1. Метод постоянного закрашивания грани

Это самый простой метод. Он заключается в том, что на грани выбирается произвольная точка и вычисляется соответствующая ей освещенность. Считается, что все остальные точки грани имеют такую же освещенность. В качестве модели освещенности обычно используются простейшие модели вида:

$$I = K_a * I_a + K_d(n,l) \qquad (6.1)$$

или

$$I = K_a * I_a + K_d(n,l) + K_s(n,h)^p \qquad (6.2)$$

Получающееся при этом изображение носит явно выраженный полигональный характер – отчетливо видно, что объект состоит из отдельных плоских граней. Это связано с тем, что если рассматривать освещенность как функцию,

определенную на поверхности какого-либо объекта, то она имеет резкие изменения на границах граней (при плоской закраске освещенность является кусочно-постоянной функцией).

Более высокого уровня реалистичности можно добиться, если воспользоваться методом, обеспечивающим непрерывное изменение функции освещенности внутри граней и вдоль их границ.

6.2.2. Метод Гуро

Затенение по Гуро – это метод линейной интерполяции функции освещенности в пределах одной грани. Он был разработан в 1971 и носит имя своего изобретателя. Это простой и эффективный метод придает ощущения изогнутости для ровного полигона. Этот метод также часто используется для сокращения глубины прорисовываемой сцены путем имитации исчезновения удаленных объектов в тумане.

Полигоном в трехмерной графике принято считать участок поверхности, имеющий как минимум три вершины лежащие в одной плоскости. Полигон может иметь произвольное количество вершин, но в трехмерной компьютерной графике в большинстве случаев используют именно треугольные полигоны. Во-первых, три вершины всегда однозначно определяют положение плоскости в пространстве, а четвертая вершина может лежать вне пределов плоскости, и ее положение придется всегда проверять, что вызовет дополнительные трудности. Во-вторых, из треугольников всегда можно получить любой другой многоугольник .

Пусть задана плоская грань v1v2v3 (рис.6.1). Найдем значение освещенности в каждой ее вершине, используя формулы 6.1 или 6.2. Обозначим получившиеся значения через I1,I2,I3. Рисуя грань v1v2v3 построчно, будем находить значения освещенности в концах каждого горизонтального отрезка путем линейной интерполяции значений вдоль ребер. Так освещенность в точке A (рис.6.1) вычисляется по следующей формуле:

$$I(A) = I_1(1\text{-}t) + I_2(t,t) = \frac{|Av1|}{|v1v2|}$$

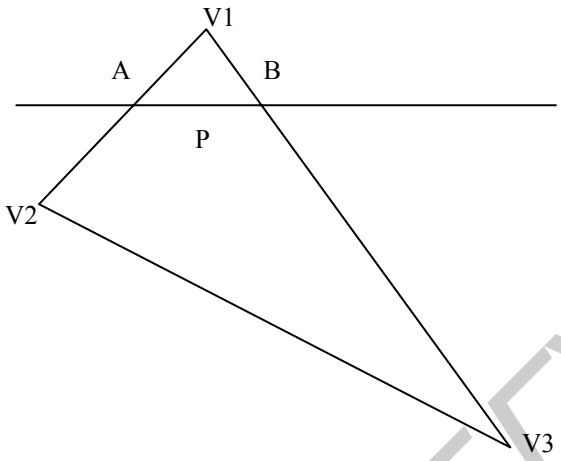

Рис. 6.1. Вычисление освещенности в точке A

При построении растрового представления отрезка AB будем считать, что интенсивность пикселей изменяется линейно от I(A) в точке A до I(B) в точке B. Ниже приводится алгоритм, осуществляющий построение горизонтального отрезка с изменением интенсивности от I_a до I_b.

1. Получить координаты точек A и B (xa и xb).
2. Рассчитать по формуле значения интенсивности в этих точках ia и ib.
3. Ввести промежуточное значение интенсивности i и присвоить ему начальное значение равное ia.
4. По формуле приведенной ниже рассчитать приращение интенсивности при переходе из одной точки к другой вдоль оси X.
   di = (ib - ia)/(xb – xa + 1)
5. Ввести промежуточное значение x для перехода от точки к точке по оси X. Присвоить x в качестве начального значения xa.

6. В цикле отрисовать всю линию на каждом шаге наращивая значение x (пока не дойдем до точки xb) и значение i на приращение di, рассчитанное по формуле.

Этот алгоритм рассчитан на то, что цветам от ia до ib соответствует набор промежуточных цветов с линейно изменяющейся интенсивностью, например, когда вся палитра состоит из оттенков одного или нескольких цветов. Подобный метод билинейной интерполяции используется в ряде графических систем, например в OpenGL.

Метод Гуро гарантирует создание непрерывной функции освещенности вдоль грани объекта, но эта функция не является гладкой (дифференцируемой) на границах граней. Следующий метод строит более гладкую функцию освещенности объекта.

6.2.3. Метод Фонга

Аппроксимация Фонга, или затенение по Фонгу , носят название по имени своего изобретателя - Ву Тонг Фонга. Этот метод дает более точные результаты при затенении многоугольников по сравнению с затенением Гуро. Суть алгоритма Фонга заключается в определении нормали к поверхности в каждой вершине многоугольника и последующей интерполяции вектора по всему полигону поверхности. Далее для каждого пикселя необходимо вычислить значение яркости, основываясь на значениях вектора нормали в этом пикселе. Этот процесс требует больше времени, чем вышеописанные методы, и его программая реализация плохо подходит для использования в играх, рассчитанных на домашний компьютер. Поэтому большинство компьютерных программ используют значительно упрощенный вариант аппроксимаций.

Затенение по Фонгу – это достаточно популярный метод для реализации затенения, особенно в программе 3D Studio. Однако, несмотря на то, что этот

метод значительно точнее затенения Гуро, он по-прежнему не является физически точным. Затенение по Фонгу – это приближенная реализация физической модели света. Каждый одиночный пиксел имеет собственное значение яркости, просчитанное при использовании интерполированного вектора нормали.

Просчет затенения по Фонгу включает вычисление двух квадратных корней, реализацию двух операций деления, четырех операций умножения. Кроме этого используется значительное количество операций сложения для каждого пикселя, а алгоритм не является целочисленным.

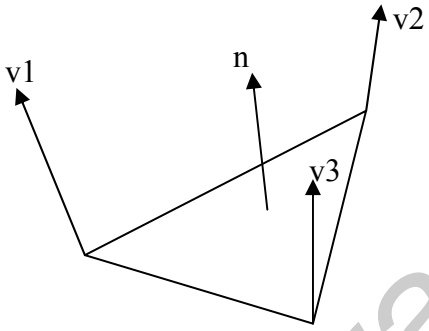

Рис. 6.2. Вычисление вектора нормали во внутренней точке

Затенение по Фонгу основывается на том, что количество света, отраженного от поверхности прямо пропорционально косинусу угла между нормалью к поверхности и направлением света. При этом не имеет значения, под каким углом вы смотрите на поверхность. Реально очень немногие поверхности обладают такими идеальными свойствами. Наиболее подходящими являются поверхности, имеющие микроскопические шероховатости, например, хорошо отшлифованное дерево. Большинство поверхностей также обладают некоторой степенью спектрального отражения. Затенение по Фонгу не разделяет их между собой. Оно также не учитывает тот факт, что яркость обратно пропорциональна квадрату расстояния от источника света. Рассмотрим полигон. В каждой вершине полигона мы имеем вектор нормали к поверхности, частью которой полигон является (v1, v2, v3). Эти векторы интерполированы по всей площади полигона, подобно тому, как это делалось для затенения Гуро. В затенении Гуро, однако, используется только одно значение -shade (это скалярная величина).

Здесь же оперируем вектором с тремя значениями Vx, Vy и Vz (три значения, три координаты – определяющие положение вектора в пространстве).

Возьмем пиксель на этом полигоне (рис. 6.2). Вектор нормали, соответствующий этому пикселю на поверхности равен **n**. Луч падает на полигон вдоль вектора **l**. Количество света, отраженного от данного пикселя, является функцией скалярного произведения векторов, дающих в результате число, определяющее длину результирующего вектора. Скалярное произведение вектора **n**(x,y,z) и вектора **l**(x,y,z) может производиться по любой из двух формул:

$$\mathbf{n}*\mathbf{l} = n_x l_x + n_y l_y + n_z l_z$$

$\mathbf{n}*\mathbf{l} = |\mathbf{n}|*|\mathbf{l}| \cos\theta$, где θ (тетта) - угол между векторами.

Нам больше подходит второй вариант, т.к. он оперирует с длинами (величинами) векторов и углом между ними. Величина нормали к поверхности равна 1. А величина вектора падающего света приводится к 1 и будет иметь значения от 0 до 1. Значение 1 соответствует самому яркому свету.

Далее результатом вычисления данной функции будет значение в диапазоне от -1 до 1, где 1 соответствует максимальной яркости. Понятия отрицательного света не существует, поэтому значения меньше 0 следует заменять на 0. Если затем умножить данное значение на величину яркости света (brightness), то получаем значение яркости рассматриваемого пикселя.

6.3. Задания к лабораторной работе

6.3.1. Разработать программу отображения выпуклого многоугольника, заданного своими вершинами (квадрата, пирамиды, октаэдра и т.п.).

6.3.2. Разработать программу отображения выпуклого многоугольника, заданного своими вершинами (квадрата, пирамиды, октаэдра и т.п.) при плавно меняющихся значениях функции освещенности.

6.3.3. Разработать программу отображения правильного N-гранника, моделирующего растровое представление шара радиуса R

6.4. Контрольные вопросы

6.4.1. Оцените сложность описанных методов рендеринга.

6.4.2. Расскажите алгоритм билинейной интерполяции.

6.4.3. К чему приведет применение в методах Гуро и Фонга более сложных методов интерполяции?

6.4.4. За счет чего метод Фонга сложнее метода Гуро?

6.4.5. Можно ли добиться абсолютной гладкости функции освещенности на границах граней?

6.4.6. Как обрабатывать модели объектов с невыпуклыми гранями?

6.4.7. Как формировать изображения невыпуклых многогранников?

Литература

Д. Роджерс, Алгоритмы машинной графики. – М.: Мир, 1989, глава 5.

Дж. Фоли, А. вэн Дэм, Основы интерактивной машинной графики. – М.: Мир, 1985, том 2, глава 16.

Лабораторная работа 7.
Удаление невидимых линий и поверхностей.

Цель работы: научиться выполнять удаление невидимых линий и граней заданного объекта.

7.1. Постановка задачи

Дано: объект, заданный в трехмерном пространстве списками вершин, ребер и граней.

Требуется отобразить только видимые грани этого объекта.

7.2. Теоретические основы

Удаление невидимых линий и поверхностей - это одна из наиболее сложных задач машинной графики. Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые невидимы для наблюдателя, смотрящего на объект из заданной точки пространства. Даже при отображении простейших моделей возникают сложности (рис. 7.1).

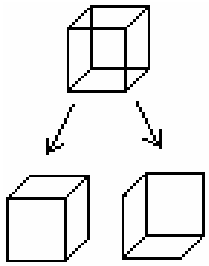

Рис. 7.1 Две возможных интерпретации невидимых граней

Разработано много алгоритмов решения данной задачи. Чем быстрее работает алгоритм, тем хуже результат его работы, и, наоборот, чем более качественные результаты показывает алгоритм, чем медленнее он работает. То есть имеет место обратная зависимость: СКОРОСТЬ ≈ 1/РЕЗУЛЬТАТ.

Все алгоритмы, решающие данную задачу, включают в себя процедуру сортировки: по геометрическому расстоянию от тела, поверхности, ребра или точки до точки наблюдения. Поскольку чем больше это расстояние, тем больше у обрабатываемого объекта вероятность оказаться заслоненным.

7.2.1. Алгоритм Робертса

Алгоритм Робертса – это первое известное решение задачи об удалении невидимых линий. Это математически элегантный метод, работающий в объектном пространстве. Алгоритм прежде всего удаляет из каждого тела те ребра или грани, которые экранируются самим телом. Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части, если таковые есть, экранируются этими телами. Поэтому вычислительная трудоемкость алгоритма Робертса растет теоретически как квадрат числа объектов. Это в сочетании с ростом интереса к растровым дисплеям, работающим в пространстве изображения, привело к снижению интереса к алгоритму Робертса. Однако математические методы, используемые в этом алгоритме, просты и точны. Кроме того, этот алгоритм можно использовать для иллюстрации некоторых важных концепций. Более поздние реализации алгоритма, использующие предварительную приоритетную сортировку вдоль оси z и простые габаритные или минимаксные тесты, демонстрируют почти линейную зависимость от числа объектов.

В алгоритме Робертса требуется, чтобы все изображаемые тела или объекты были выпуклыми. Невыпуклые тела должны быть разбиты на выпуклые части. В этом алгоритме выпуклое многогранное тело с плоскими гранями должно представиться набором пересекающихся плоскостей. Уравнение произвольной плоскости в трехмерном пространстве имеет вид:

$$ax + by + cZ + d = 0$$

В матричной форме этот результат выглядит так :

$$[xyz1][P]^T = 0.$$

где $[P]^T = [abcd]$ представляет собой плоскость. Поэтому любое выпуклое твердое тело можно выразить матрицей тела, состоящей из коэффициентов уравнений плоскостей, т. е.

$$[V] = \begin{bmatrix} a1 & a2 & . & an \\ b1 & b2 & . & bn \\ c1 & c2 & . & cn \\ d1 & d2 & . & dn \end{bmatrix}$$

где каждый столбец содержит коэффициенты одной плоскости.

Напомним, что любая точка пространства представима в однородных координатах вектором [S]=[x y z 1]. Более того, если точка [S] лежит на плоскости, то $[S]\cdot[P]^T = 0$. Если же [S] не лежит на плоскости, то знак этого скалярного произведения показывает, по какую сторону от плоскости расположена точка. В алгоритме Робертса предполагается, что точки, лежащие внутри тела, дают положительное скалярное произведение.

Алгоритм Робертса выполняется в два этапа. На первом этапе из тела удаляются те ребра и грани, которые заслоняются самим телом. На втором этапе вычисляется, какие части тела заслоняются другими телами и, следовательно, не должны быть изображены.

Первый этап алгоритма Робертса. Необходимо определить грани, которые заслоняются самим телом. Если угол между направлением взгляда и вектором нормали больше 90 градусов, грань будет видима. Следовательно, для видимых граней скалярное произведение между внешней нормалью и направлением взгляда должно быть отрицательным. Тест на видимость выполняется следующим образом:

$$[\text{xв,ув,zв,1}]\begin{bmatrix} a1 & a2 & . & an \\ b1 & b2 & . & bn \\ c1 & c2 & . & cn \\ d1 & d2 & . & dn \end{bmatrix} = [\text{t1,t2,…,tn}]$$

где xв,ув,zв – координаты вектора взгляда. Проанализируем каждую компоненту ti (она соответствует скалярному произведению вектора взгляда на вектор нормали). Если ti меньше 0, то i-я грань будет видима, в противном случае невидима. Невидимым является ребро, смежное для двух невидимых граней.

Второй этап алгоритма Робертса. Если задано только одно тело, то алгоритм завершается, и второй этап не требуется. Если же сцена представлена несколькими телами, то необходимо определить, закрывают ли остальные тела анализируемое тело и если да, то удалить невидимые части заданного тела.

Основные шаги при реализации второго этапа алгоритма Робертса:

Шаг 1. Сформировать приоритетный список этих тел. Выполнить сортировку по z. Сортировка производится по максимальным значениям координаты z вершин тел. Первым в упорядоченном списке и обладающим наибольшим приоритетом будет то тело, у которого минимальное среди максимальных значений z. В используемой правой системе координат это тело будет самым удаленным от точки наблюдения, расположенной в бесконечности на оси z.

Шаг 2. Для каждого тела из приоритетного списка. Проверить экранирование всех лицевых ребер всеми другими телами сцены. Тело, ребра которого проверяются, называется пробным объектом, а тело, относительно которого в настоящий момент производится проверка, называется пробным телом. Естественно, что нужно проверять экранирование пробного объекта только теми пробными телами, у которых ниже приоритеты.

Шаг 3. Провести проверки экранирования для прямоугольных объемлющих оболочек пробного объекта и пробного тела. Если

x_{min} (пробное тело) = x_{max} (пробный объект) или

x_{max} (пробное тело) < x_{min} (пробный объект) или

y_{min} (пробное тело) = y_{max} (пробный объект) или

y_{max} (пробное тело) < x_{min} (пробный объект),

то пробное тело не может экранировать ни одного ребра пробного объекта. Перейти к следующему пробному телу. В противном случае:

Шаг 4. Провести предварительные проверки, чтобы увидеть, не протыкается ли пробное тело пробным объектом и существует ли возможность частичного экранирования первого последним.

Сравнить максимальное значение z У пробного объекта с минимальным значением г у пробного тела.

Если z_{max} (пробный объект) < z_{min} (пробное тело), то протыкание невозможно. Перейти к следующему телу. В противном случае:

Проверить видимое протыкание.

Если z_{max} (пробный объект) = z_{min} (пробное тело), то пробный объект может проткнуть переднюю грань пробного тела.

Установить флаг видимого протыкания для последующего использования. Занести проткнутое тело в список протыканий. Если

x_{max} (пробный объект) = x_{min} (пробное тело) или

x_{min} (пробный объект) < x_{max} (пробное тело),

то пробный объект может проткнуть бок пробного тела.

Установить флаг видимого протыкания для последующего использования. Занести тело в список протыканий. Если

y_{max} (пробный объект) = y_{min} (пробное тело) или

y_{min} (пробный объект) < y_{max} (пробное тело),

то пробный объект может проткнуть верх или низ пробного тела.

Установить флаг видимого протыкания для последующего использования. Занести проткнутое тело в список протыканий.

Если список протыканий пуст, устанавливать флаг протыкания не надо.

Шаг 5. Провести проверку экранирования ребер. Проверка полной видмости. Если ребро полностью видимо, то перейти к следующему ребру.

Вычислить видимые участки отрезков и сохранить их для последующей проверки экранирования телами с более низкими приоритетами.

7.3. Задания к лабораторной работе

7.3.1. Постройте модель выпуклого (невыпуклого) многогранника, выполните анализ невидимых граней и постройте его изображение.

7.3.2. Постройте модель шара в виде N-гранника и его изображение.

7.4. Контрольные вопросы

7.4.1. В чем состоит идея алгоритма Робертса?

7.4.2. Каковы достоинства алгоритма Робертса? Можно ли его применять к невыпуклым темам?

7.4.3. Как вычислить вектор нормали к грани? Как убедиться в том, что нормаль является внутренней?

7.4.4. Какова сложность алгоритма Робертса?

Литература

Д. Роджерс, Алгоритмы машинной графики. – М.: Мир, 1989, глава 4.

Дж. Фоли, А. вэн Дэм, Основы интерактивной машинной графики. – М.: Мир, 1985, том 2, глава 16.

Св. план 2002, поз. 56

Учебное издание

Садыхов Рауф Хосровович
Старовойтов Валерий Васильевич
Конопелько Виталий Геннадьевич
Юрас Александр Васильевич

Лабораторный практикум по курсу
“Машинная графика”
для студентов специальности Т10.03.00

Редактор
Корректор

Подписано в печать	.04.2002.	Формат 60x84 1/16.
Бумага офсетная.	Печать ризографическая.	Усл.печ.л.
Уч.-изд.л. 3.	Тираж экз.	Заказ

Издатель и полиграфическое исполнение:
Учреждение образования “Белорусский государственный университет информатики и радиоэлектроники”
Лицензия ЛП № 156 от 05.02.2001
Лицензия ЛВ № 509 от 03.08.2001
220013, Минск, П. Бровки, 6.