ГОСУДАРСТВЕННЫЙ КОМИТЕТ РОССИЙСКОЙ ФЕДЕРАЦИИ ПО
ВЫСШЕМУ ОБРАЗОВАНИЮ

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

С.М.АВДЕЕВА, А.В.КУРОВ

БАЗОВЫЕ АЛГОРИТМЫ МАШИННОЙ ГРАФИКИ

Методические указания по выполнению лабораторных работ по дисциплине "Машинная графика"

Москва
1996

ОГЛАВЛЕНИЕ Стр.

Введение 3
1. Лабораторная работа N1 "Программная реализация основных алгоритмов построения отрезков и исследование их временных и визуальных характеристик" 4
2. Лабораторная работа N2 "Программная реализация алгоритмов генерации окружности и исследование их временных и визуальных характеристик" 16
3. Лабораторная работа N 3 " Программная реализация алгоритмов заполнения сплошных областей с затравкой и исследование их временных характеристик". 24
4. Лабораторная работа N 4 " Программная реализация алгоритмов заполнения сплошных областей с упорядоченным списком ребер и оценка их временных характеристик" 29
5. Лабораторная работа N 5 " Программная реализация алгоритмов заполнения сплошных областей по ребрам и оценка их временных характеристик" 33
Литература 37

ВВЕДЕНИЕ

Дисциплина "Машинная графика" является одной из основных при обучении студентов по специальности "Программное обеспечение вычислительной техники и автоматизированных систем". Учебный план этой дисциплины предусматривает изложение основных алгоритмов двумерной и трехмерной машинной графики. Поскольку при создании трехмерных изображений многократно приходится использовать различные алгоритмы плоской графики (такие, как построение отрезков, кривых, закраска замкнутых областей, отсечение отрезков и многоугольников), то будущий специалист должен хорошо ориентироваться в этих алгоритмах, чтобы выбирать и реализовывать в создаваемых программных продуктах наиболее рациональные из них по тем или иным критериям.

Алгоритмы плоской графики отличаются большей простотой по сравнению с трехмерными алгоритмами. В связи с этим в рамках фиксированного количества лекционных часов имеет смысл больше времени выделить для изложения более сложных и более интересных алгоритмов удаления невидимых линий и поверхностей и построения реалистических изображений.

Такую задачу можно решить в случае самостоятельного изучения базовых алгоритмов самими студентами. Такой подход требует, однако, наличия соответствующей учебной литературы, причем в достаточном количестве. Написание данных методических указаний позволит, по мнению авторов, решить в определенной степени эту проблему. Использование методических материалов является еще более актуальным для слабослышащих студентов (группа учебного центра), а также студентов, обучающихся по специальности "ПРограммное обеспечение вычислительной техники и автоматизированных систем" в рамках второго образования.

Самостоятельное изучение теоретического материала закрепляется в ходе выполнения лабораторных работ, большая часть которых и посвящена реализации базовых алгоритмов. При этом во время лабораторных занятий имеются благоприятные возможности для получения консультаций и ответов со стороны преподавателя на
невыясненные вопросы студентов.

К базовым алгоритмам или алгоритмам нижнего уровня относят алгоритмы, позволяющие вычерчивать основные геометрические фигуры (отрезки, окружности, эллипсы и т.д.), а также выполнять такие операции, как закраска сплошных областей и отсечение отрезков и многоугольников. Обычно часть из этих алгоритмов реализуется в качестве примитивов в графических библиотеках языков высокого уровня.

Эти алгоритмы используются в алгоритмах более высокого уровня (или верхнего), к которым относят построение трехмерных и реалистических изображений. Основные алгоритмы высокого уровня рассмотрены авторами в методическом пособии по выполнению курсовой работы по "Машинной графике".

ЛАБОРАТОРНАЯ РАБОТА N1

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ОСНОВНЫХ АЛГОРИТМОВ ПОСТРОЕНИЯ ОТРЕЗКОВ И ИССЛЕДОВАНИЕ ИХ ВРЕМЕННЫХ И ВИЗУАЛЬНЫХ ХАРАКТЕРИСТИК

Цель работы: реализация алгоритмов построения отрезков по методу цифрового дифференциального анализатора (ЦДА) и алгоритмов Брезенхема (действительного, целочисленного и с устранением ступенчатости) и исследование их характеристик и сравнение полученных результатов.

В ходе выполнения практической части этой лабораторной работы необходимо выполнить следующие пункты задания:

1.Реализовать алгоритмы ЦДА, Брезенхема (действительный, целочисленный, с устранением ступенчатости).

2.Сравнить визуально отрезки, построенные в соответствии с каждым алгоритмом, а также с отрезком, построенным процедурой языка высокого уровня. Проверить попадание отрезка в заданную конечную точку.

3.Определить время, затрачиваемое на построение отрезка по каждому из алгоритмов.

4.Для заданного алгоритма получить зависимость длины максимальной ступеньки от угла наклона отрезка и отобразить ее в виде графика или гистограммы.

Программы, реализующие алгоритмы, должны обеспечивать построение произвольного отрезка, то есть располагающегося в любом из восьми октантов, включая горизонтальный и вертикальный, а также предусматривать высвечивание точки в случае вырожденного отрезка. Визуальное сравнение отрезков, построенных по разным алгоритмам, можно осуществить либо высвечиванием близко расположенных параллельных отрезков, либо рисованием на одном и том же месте одного отрезка разными алгоритмами. При этом каждый раз новый отрезок высвечивается неовым цветом. В случае несовпадения результатов не все старые пикселы высветятся новым цветом, что и будет свидетельствовать о различающихся результатах. Проверку попадания отрезка в требуемую конечную точку можно осуществить путем сравнения цвета конечной точки с цветом рисования отрезка.

Исследование временных характеристик построения отрезков в силу высокой скорости построения одного отрезка следует проводить путем замера времени, затрачиваемого на высвечивание нескольких десятков отрезков (например, пятидесяти или ста отрезков). При этом целесообразно сначала установить текущее время в ноль, а затем снять показания текущего времени. При этом достаточно учитывать значения сотых, десятых долей и целых значений секунд, так как в силу высокого быстродействия процесс рисования не превысит минуты. Если же не устанавливать предварительно время в ноль, то необходимо будет учитывать возможное изменение минут и даже часов.

Для выполнения последнего пункта задания достаточно исследовать отрезки, расположенные в пределах одного октанта. При этом ступеньку будут образовывать те пикселы, у которых значение одной из координат остается неизменным при изменяющейся другой координате. Построенный график (гистограмма) должны имметь обозначение и оцифровку осей координат.

Теоретическая часть

Необходимость разработки специальных алгоритмов построения геометрических объектов объясняется тем, что сами эти объекты имеют непрерывную (аналоговую) природу, а изображение строится, как правило, на экране растрового дисплея, то есть получаемое изображение носит дискретный характер. Это означает, например, что нельзя построить непосредственно отрезок, соединяющий две заданные точки экрана.

Процесс нахождения пикселов, наилучшим образом аппроксимирующих заданный отрезок, называется разложением отрезка в растр.

Решение поставленной задачи очевидно лишь для трех типов отрезков: горизонтальных, вертикальных и наклоненных под углом в 45 .

Перед рассмотрением алгоритмов построения отрезков необходимо сформулировать наиболее общие и простые требования, предъявляемые к таким алгоритмам. Во-первых, отрезки должны выглядеть прямыми; начинаться и заканчиваться в заданных точках. Во-вторых, яркость вдоль отрезка должна быть постоянной и не зависеть от длины и наклона. В-третьих, алгоритмы должны работать быстро.

Первое требование в силу дискретной природы растрового дисплея выполнено всегда быть не может. Можно лишь добиться того, что визуально (человеческим глазом) отрезок будет восприниматься прямым. Решение этой задачи может достигаться путем увеличения разрешающей способности экрана дисплея и применения методов устранения ступенчатости.

Второму требованию удовлетворяют также только горизонтальные, вертикальные и наклоненные под углом в 45 отрезки. Однако вертикальные и горизонтальные отрезки по сравнению с отрезками, расположенными под 45 , будут выглядеть ярче, так как расстояние между соседними пикселами у них меньше, чем у наклонных отрезков. Обеспечение постоянной яркости вдоль отрезка требует высвечивания очередного пиксела яркостью, зависящей от расстояния между пикселами, вычисление которого производится с использованием операций извлечения квадратного корня и умножения (возводить в квадрат лучше путем умножения ячисла самого на себя). Использование этих операций существенно замедляет работу алгоритма, поэтому второе требование остается, как правило, невыполненным.

Удовлетворение третьего требования достигается путем сведения к минимуму вычислительных операций, использования операций над целочисленными данными, а также реализацией алгоритмов на аппаратном или микропрограммном уровне.

Многие алгоритмы вычерчивания отрезков и кривых используют пошаговый принцип, суть которого состоит в том, что координаты высвечиваемого пиксела определяются каждый раз на очередном шаге вычислений, а не вычисляются заранее для всех пикселов. Результат вычислений на текущем шаге зависит от результатов, полученных на предыдущем шаге.

Первый алгоритм вычерчивания отрезков - по методу цифрового дифференциального анализатора - использует достаточно общий принцип, известный в математике: изученгие какого-либо явления на основе дифференциального уравнения или системы таких уравнений, описывающей это явление.

Поскольку прямая линия на плоскости описывается уравнением вида

$$AX+BY+C=0,$$

где A,B,C - коэффициенты этого уравнения, то производная dy/dx
является постоянной.

Заменив дифференциалы конечными разностями, получим

$$\frac{Y}{X} = \frac{Y_к - Y_н}{X_к - X_н}, \qquad (1)$$

где Xн,Yн и Xк,Yк - коррдинаты начальной и конечной точек отрезка.

Ордината очередного пиксела Yi+1 может быть вычислена по известной ординате предыдущего пиксела Yi следующим образом:

$$Y_{i+1}=Y_i+ Y$$

Подставляя Y из (1), получим

$$Y_{i+1}=Y_i + \frac{Y_к - Y_н}{X_к - X_н} X \qquad (2)$$

Остается определить величину приращения X. В рассматриваемых здесь алгоритмах большее из приращений (X или Y) выбирается в качестве единицы растра, а приращение вдоль другой координатной

оси подлежит определению. Если же поступить по-другому (меньшее из приращений взять равным единице), то отрезок на экране может получиться "дырявым", то есть состоящим из отдельных точек, не расположенных вплотную друг к другу.

Алгоритм разложения отрезка в растр по методу ЦДА может быть записан следующим образом:

1.Ввод исходных данных Xн,Yн,Xн,Yк.

2.Проверка вырожденности отрезка. Если отрезок вырожден, то высвечивание точки и переход к п.7.

3.Вычисление l= !Xк-Xн!, если !Xк-Xн!>!Yк-Yн!
!Yк-Yн!, если !Yк-Yн!>!Xк-Xн!

4.Вычисление dX=(Xк-Xн)/l, dY=(Yк-Yн)/l.

5.Задание координатам текущей точки начальных значений: X=Xн, Y=Yн.

6.Цикл от i=1 до i=l+1 с шагом 1:

высвечивание точки с текущими координатами (E(X),E(Y)), где E - операция округления до ближайшего целого);

вычисление координат следующей точки:

X=X+dX, Y=Y+dY.

7.Конец.

Алгоритм Брезенхема

Работа алгоритма Брезенхема основывается на использовании понятия ошибка. Ошибеой здесь называется расстояние между действительным положением отрезка и ближайшим пикселом сетки растра, который аппроксимирует отрезок на очередном шаге.

На каждом шаге вычисляется величина ошибки и в зависимости от полученного значения выбирается пиксел, ближе расположенный к идеальному отрезку. Поскольку при реализации алгоритма на ЭВМ удобнее анализировать не само значение ошибки, а ее знак, то истинное значение ошибки смещается на -0,5.

Пусть f - значение ошибки на очередном i-м шаге, Yи - ордината идеального отрезка, а Yi - ордината пиксела, выбранного для аппроксимации отрезка на том же i-м шаге, m - тангенс угла наклона отрезка.

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

f=m-0,5 (3)

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

Как и в предыдущем алгоритме, большее по модулю из приращений принимается равным шагу растра, то есть единице, причем знак приращения совпадает со знаком разности конечной и начальной координат отрезка:

X=sign(Xк-Xн), если ! X!>! Y!

Y=sign(Yк-Yн), если ! Y!>! X!

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 fi+1=Yиi+1-Yi+1=Yиi+m-Yi+1=fi+m (если Yi+1=Yi)

(4)

(Поскольку в алгоритме и программе не надо сохранять значения ошибок для всех шагов, то последнее выражение можно записать как f=f+m).

В зависимости от полученного значения ошибки выбирается пиксел с той же ординатой (при f<0) или пиксел с ординатой, на единицу большей, чем у предыдущего пиксела (при f>0).

Поскольку предварительное значение ошибки вычисляется заранее, то есть f+m вычислено на предыдущем шаге, то во втором случае останется только вычесть единицу из значения ошибки:

f=f-1.

Алгоритм Брезенхема, работающий с действительными величинами, можно записать в следующем виде:

1.Ввод исходных данных Xн,Yн,Xк,Yк.

2.Проверка вырожденности отрезка. Если отрезок вырожденный, то высвечивается точка и осуществляется переход к п.11.

3.Вычисление приращений dX=Xк-Xн и dY=Yк-Yн.

4.Вычисление шага изменения каждой координаты пиксела: SX=sign(dX), SY=sign(dY).

5.Вычисление модулей приращения координат: dX=!dX!, dY=!dY!

6.Вычисление модуля тангенса угла наклона отрезка: m=dY/dX.

7.Анализ вычисленного значения m и обмен местами dX и dY при m>1:

если m>1, то выполнить

W=dX, dX=dY, dY=W, m=1/m, fl=1; если m<1, то fl=0

(fl - флаг, определяющий факт обмена местами координат).

8.Инициализация начального значения ошибки: f=m-0,5

9.Инициализация начальных значений координат текущего пиксела:

X=Xн, Y=Yн

10.Цикл от i=1 до i=dX+1 с шагом 1:

Высвечивание точки с координатами (X,Y).

Вычисление координат и ошибки для следующего пиксела:

Если f>0, то

если fl=1, то X=X+SX

иначе Y=Y+SY;

корректировка ошибки f=f-1.

Если f<0, то

если fl=1, то Y=Y+SY

иначе X=X+SX.

Вычисление ошибки f=f+m.

11.Конец.

Приведенный алгоритм легко преобразуется к целочисленному варианту. Для этого выражение (3) запишем в виде: f= Y/ X-1/2 и, умножив обе части этого равенства еа 2 X, получим:

2 X f=2 Y- X.

Обозначив f1=2 X f, окончательно получим:

f1=2 Y - X (5)

(В соответствии с этим выражением должно вычисляться теперь начальное значение ошибки в п.8 алгоритма).

Тогда подсчет нового значения ошибки в п.10 "действительного" алгоритма будет производиться по формулам:

f1=f1+2 Y и f1=f1-2 X.

Алгоритм Брезенхема с устранением ступенчатости Принципиально устранить ступенчатость (лестничный эффект) невозможно. Однако применением специальных методов можно добиться того, что визуально ступеньки будут слабо заметны или практически незаметны. Излагаемый здесь способ устранения ступенчатости рассматривает пиксел не как математическую точку, а как некоторую конечную область.

При наличии нескольких оттенков цвета внешний вид отрезка улучшается путем размывания его краев. Для этого интенсивность пиксела устанавливается пропорционально площади части пиксела, находящейся под отрезком. Поэтому необходимо получить выражения для вычисления этой площади.

Отрезок, тангенс угла наклона m которого лежит в диапазоне 0<m<1 (как было указано ранее, все отрезки приводятся к такому виду), может при данном значении абсциссы пересечь один или два пиксела.

В первом случае искомая площадь находится как сумма площадей прямоугольника S1 и треугольника S2:

S=S1+S2=Yi*1+1*m/2=Yi+m/2,

где Yi- расстояние между нижней левой вершиной пиксела и точкой пересечения отрезка с левой границей пиксела.

Если отрезок пересекает два пиксела, то площадь части нижнего пиксела (S4) может быть найдена как разность площади всего пиксела и площади треугольника S3:

$$S4=1-S3=1-\frac{(1-Yi)(1-Yi)}{2m}$$

где 1-Yi - длина одного катета, (1-Yi)/m - длина другого катета.

Площадь S5 части верхнего пиксела - треугольника равна

$$S5=\left(1-\frac{1-Yi}{m}\right)\cdot\frac{2}{m}$$

Практика показывает, что учет площади S5 позволяет более реалистично представить отрезок, поэтому найдем сумму площадей S4 и S5:

$$S=S4+S5=1-\frac{(1-Yi)}{2m}+\left(1-\frac{1-Yi}{m}\right)\cdot\frac{2}{m}=$$

$$=\frac{2m-1+2Yi-Yi+m+1+Yi-2m+2mYi-2Yi}{2m}=Yi+m/2$$

Таким образом, в обоих случаях площадь вычисляется по одной и той же формуле.

В качестве ошибки в данном алгоритме принимается часть площади пиксела, находящаяся под отрезком, то есть Yi+m/2. Вычислим значение ошибки при переходе к соседнему пикселу. Если ордината соседнего пиксела не увеличивается, то площадь, находящаяся под отрезком, увеличивается на величину площади прямоугольника со сторонами 1 и m, то есть f=f+m.

Если же ордината соседнего пиксела увеличивается на единицу, то вычисленная доля площади пиксела будет содержать и площадь пиксела, через который отрезок не проходит, следовательно, необходимо вычесть величину площади пиксела, то есть единицу:

f=f+m-1.

Поскольку доля площади не может быть отрицательной величиной, то по сравнению с ранее рассмотренными алгоритмами Брезенхема необходимо скорректировать величину ошибки, прибавив к ней величину W=1-m. При этом начальное значение ошибки будет равно

f=m-0,5+1-m=0,5, а значение ошибки будет лежать в диапазоне $0<f<1$.

Первый пиксел отрезка будет, таким образом, всегда высвечиваться интенсивностью, равной половине максимальной. Для более реалистичного изображения это значение можно изменить.

Можно сразу получить не дробное значение максимальной интенсивности, а ее непосредственное значение, если умножить на максимальное количество уровней интенсивности следующие величины: тангенс угла наклона m, коэффициент W, ошибку f.

Общий алгоритм Брезенхема с устранением ступенчатости может быть записан следующим образом:

1.Ввод исходных данных Хн,Yн,Хк,Yк (координаты концов отрезка), I - количество уровней интенсивности.

2.проверка вырожденности отрезка. Если отрезок вырожден, то высвечивание отдельного пиксела и переход к п.13.

3.Вычисление приращений dX=Хк-Хн и dY=Yк-Yн.

4.Вычисление шага изменения каждой координаты: SX=sign(dX), SY=sign(dY).

5.Вычисление модулей приращения координат: dX=!dX!, dY=!dY!.

6.Вычисление модуля тангенса угла наклона m=dY/dX.

7.Анализ вычисленного значения m и обмен местами dX и dY при m>1:

если m>1, то выполнить

p=dX; dX=dY; dY=p; m=1/m; fl=1;

если m<1, то fl=0.

(fl - флаг, определяющий факт обмена местами координат).

8.Инициализация начального значения ошибки f=I/2.

9.Инициализация начальных значений координат текущего пиксела:

X=Xн, Y=Yн.

10.Вычисление скорректированного значения тангенса угла наклона m=mI и коэффициента W=I-m.

11.Высвечивание пиксела с координатами (X,Y) интенсивностью E(f).

12.Цикл от i=1 до i=dX с шагом 1:

Если f<W, то

если fl=0, то X=X+SX;

если fl=1, то Y=Y+SY;

f=f+m.

Если f>W, то X=X+SX, Y=Y+SY, f=f-W.

Высвечивание пиксела с координатами (X,Y) интенсивностью E(f).

13.Конец.

Данный алгоритм, как и в предыдущем случае, можно тем же приемом преобразовать к целочисленному виду. Умножив на 2 X, получим измененные выражения для вычисления ошибки f1:

начальное значение ошибки f1= X1,

а текущее значение будет вычисляться как f1=f1+2 Y или

f1=f1-2 X1+2 Y1, где X1= XI и Y1= YI.

При этом, правда, значение интенсивности для высвечивания очередного пиксела должно иметь значение E(f/2 X), причем деление производится целочисленное.

ЛАБОРАТОРНАЯ РАБОТА N2

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМОВ ГЕНЕРАЦИИ ОКРУЖНОСТИ И ИССЛЕДОВАНИЕ ИХ ВРЕМЕННЫХ И ВИЗУАЛЬНЫХ ХАРАКТЕРИСТИК

Цель работы: реализация алгоритмов построения окружности, исследование и сравнение характеристик алгоритмов.

При выполнениии этой лабораторной работы студент должен выполнить следующий объем работ:

1.Реализовать алгоритм построения окружности на основе ее уравнения и алгоритм Брезенхема.

2.Сравнить визуально результаты алгоритмов п.1 и результат, выдаваемый процедурой рисования окружности графической библиотеки.

3.Исследовать временные характеристики всех трех алгоритмов, построив графики зависимости времени построения окружности от ее радиуса.

Программы, реализующие алгоритмы, должны обеспечивать построение окружностей, центр которыхнаходится в произвольной точке экрана, а также учитывать разные разрешающие способности экрана.

Визуальное сравнение результатов рисования окружностей следует выполнить попарным наложением друг на друга окружностей одного радиуса, но вычерченных разными алгоритмами и разными цветами.

Графики следует построить в пределах одного экрана, друг под другом, что легко позволит сравнить скорости роста временных затрат во всех трех вариантах. Чтобы легко можно было воспользоваться полученными результатами, оси координат у графиков должны быть надписаны и оцифрованы.

Теоретическая часть

Самый простой подход к построению окружности может заключаться в использовании ее уравнения (X-Xo) + (Y-Yo) = R или X=Xo+Rcost, Y=Yo+Rsint, где Xo,Yo - координаты центра, R - радиус окружности, t - параметр (0<t<2). При этом, учитывая тот факт, что шаг изменения аргумента должен составлять величину t=1/R, рассчитать для каждого значения параметра t значения координат соответствующих точек окружности и соединить их затем отрезками прямых. Такой путь требует довольно большого количества вычислений, в чем студенты и должны убедиться в ходе выполнения лабораторной работы.

Другим, более простым и эффективным, путем является использование алгоритма Брезенхема для генерации окружности. Чтобы построить полную окружность, достаточно сгенерировать ее одну восьмую часть. Остальные части получаются затем путем симметричного отражения относительно определенной прямой. Так, отражая одну восьмую часть, построенную в первом октанте для углов в диапазоне 0-45 , относительно прямой с уравнением Y=X, получим одну четвертую часть, лежащую в первом квадранте. Отразив эту четверть относительно прямой X=0, получим одну вторую часть, лежащую выше оси абсцисс, наконец, отразив эту полуокружность относительно прямой Y=0, получим полную окружность.

Рассмотрим построение части окружности, лежащей в первом квадранте. Предположим, что центр окружности лежит в начале координат, ее радиус равен R, ось абсцисс направлена вправо, ось ординат - вверх. Пусть алгоритм начинает работу в точке с координатами X=0, Y=R, тогда он заканчивает работу в точке X=R, Y=0.

При таком направлении построения окружности Y как функция аргумента X будет являться монотонно убывающей (Y= R - X) .

Для любого пиксела при выбранном направлении генерации окружности существует только три варианта выбора следующего пиксела, наилучшим образом аппроксимирующего окружность: горизогтальный пиксел вправо, диагональный вниз и вправо, вертикальный вниз.

В качестве критерия выбора очередного пиксела используется минимум модуля разности квадратов расстояний от центра окружности до пиксела и до идеальной окружности. Эти величины имеют следующие значения :

$$L_г = |(X_i+1) + Y_i - R|$$

$$L_д = |(X_i+1) + (Y_i-1) - R|$$

$$L_в = |X_i + (Y_i-1) - R|,$$

где $L_г$, $L_д$, $L_в$ - искомые величины расстояний для горизонтального, диагонального и вертикального направлений соответственно, X_i, Y_i - координаты текущего пиксела.

В окрестности текущей точки возможны пять вариантов прохождения окружности: 1)между горизонтальным (X_i+1,Y_i) и диагональным (X_i+1,Y_i-1) пикселами; 2)между горизонтальным (X_i+1,Y_i) и вторым диагональным (X_i+1,Y_i+1) пикселами; 3)между вертикальным (X_i,Y_i-1) и диагональным (X_i+1,Y_i-1); 4)между вертикальным (X_i,Y_i-1) и третьим диагональным (X_i-1,Y_i-1) пикселами; 5)прохождение точно через диагональный (X_i+1,Y_i-1) пиксел.

По аналогии с ошибкой в алгоритме построения отрезков выберем и здесь некоторую величину, анализ знака которой позволит выбрать нужный пиксел.

Такой величиной D является разность квадратов расстояний от центра окружности до диагонального (X_i+1,Y_i-1) пиксела и до идеальной окружности:

$$D_i = (X_i+1) + (Y_i-1) - R$$

Отрицательное значение D_i может быть только в том случае, когда диагональный пиксел (X_i+1,Y_i-1) лежит внутри окружности, то есть это случаи 1 и 2 и следует выбирать горизонтальный (X_i+1,Y_i) или диагональный (X_i+1,Y_i-1) пиксел.

Рассмотрим случай 1 и вычислим разность D1 расстояний $L_г$ и $L_д$, то есть

$$D1 = L_г - L_д = |(X_i+1) + Y_i - R| - |(X_i+1) + (Y_i-1) - R| \qquad (6)$$

При $L_г < L_д$ $D1<0$ и в этом случае расстояние до горизонтального пиксела меньше, чем до диагонального, поэтому выбирается горизонтальный пиксел. При $L_г > L_д$ $D1>0$ и в этом случае расстояние до диагонального пиксела меньше, чем до горизонтального, поэтому выбирается диагональный пиксел.

При $L_г = L_д$ $D1=0$ и можно выбирать либо горизонтальный, либо диагональный пиксел. Для определенности выберем горизонтальный пиксел.

Для сокращения вычислений упростим выражение (6). При этом следует учесть, что для случая 1 $L_г>0$, а $L_д<0$, так как горизонтальный пиксел лежит вне окружности, а диагональный внутри окружности.

$$D1 = (X_i+1) + Y_i - R + (X_i+1) + (Y_i-1) - R =$$
$$= 2[(X_i+1) + (Y_i-1) - R] + 2Y_i - 1 = 2(D_i+Y_i) - 1$$

во втором случае выбирается горизонтальный пиксел, второй диагональный пиксел (X_i+1,Y_i+1) выбран быть не может в силу монотонного убывания функции. Для этого случая $L_г<0$ и $L_д<0$, поэтому

$$D1 = R - (X_i+1) - Y_i + (X_i+1) + (Y_i-1) - R = 1 - 2Y_i$$

Для первого квадранта $1<Y_i<R$, поэтому $D1=1-2Y_i<0$. Таким образом, и в случае 2, если $D1<0$, то выбирается горизонтальный пиксел.

Рассмотрим случай $D_i>0$. Это будет справедливо только в том случае, если диагональный пиксел лежит вне окружности, то есть это случаи 3 и 4 и надо выбирать либо диагональный (X_i+1,Y_i-1), либо вертикальный (X_i,Y_i-1) пиксел.

Рассмотрим случай 3 и вычислим разность расстояний $L_д$ и $L_в$:

$$D2 = L_д - L_в = |(X_i+1) + (Y_i-1) - R| - |X_i + (Y_i-1) - R| \qquad (7)$$

При $L_д < L_в$ $D2<0$ и расстояние до диагонального пиксела меньше, чем до внртикального, поэтому выбирается диагональный пиксел.

При $L_д > L_в$ $D2>0$ - расстояние до вертикального пиксела меньше, чем до диагонального, поэтому выбирается вертикальный пиксел.

При $L_д = L_в$ $D2=0$ и можно выбирать либо диагональный, либо вертикальный пиксел. Для определенности выберем диагональный пиксел.

Для упрощения вычисления выражения (7) учтем, что для случая 3 Lд>0, а Lв<0, так как диагональный пиксел лежит вне окружности, а вертикальный внутри окружности.

D2=(Xi+1) + (Yi-1) - R+Xi + (Yi-1) - R=
=2[(Xi+1) + (Yi-1) - R-2Xi]-1=2(Di-Xi)-1

В четвертом случае выбирается вертикальный пиксел, третий диагональный пиксел (Xi-1,Yi-1) выбран быть не может в силу монотонного убывания функции. Для этого случая Lд>0 и Lв>0 и

D2=(Xi+1) + (Yi-1) - R - Xi - (Yi-1) + R=2Xi+1

Для первого квадранта 0<Xi<R, поэтому D2=2Xi+1>0. Таким образом, и в случае 4. если D2>0, то выбирается вертикальный пиксел.

Остается проверить случай 5, для которого Di=0, то есть диагональный пиксел лежит точно на окружности. вычислим значение D1, учитывая, что Lг>0, а Lд=0:

D1=(Xi+1) + Yi - R > 0

Вычислим значение D2, учитывая, что Lд=0, Lв<0:

D2=Xi - (Yi-1) - R < 0

Таким образом, при Di=0 выполняются те же условия для D1 и D2, как и в ранее рассмотренных случаях.

Теперь остается записать формулы для вычисления координат очередного пиксела и критерия для каждого из трех возможных
направлений движения.

При переходе к горизонтальному пикселу:

Xi+1=Xi+1
Yi+1=Yi
Di+1=(Xi+1+1) + (Yi+1-1) - R =(Xi+1) +(Yi-1) - R +2Xi+3=
=Di+2Xi+1+1

При переходе к диагональному пикселу:

Xi+1=Xi+1
Yi+1=Yi-1
Di+1=(Xi+2) + (Yi-2) - R =Di+2Xi+3-2Yi+3=
=Di+2Xi+1-2Yi+1+2

При переходе к вертикальному пикселу:

Xi+1=Xi
Yi+1+Yi-1
Di+1=(Xi+1) + (Yi-2) - R =Di-2Yi+3=Di-2Yi+1+1

Начальное значение для X=0, Y=R будет равно:
D=1+(R-1) - R =2(1-R).

Алгоритм Брезенхема для генерации окружности в первом квадранте может быть записан в следующем виде:

1.Ввод исходных данных R (радиус окружности) и при необходимости Xc,Yc (координаты центра окружности).

2.Задание начальных значений текущих координат пиксела X=0, Y=R, параметра D=2(1-R), установка конечного значения орди-
наты пиксела Yk=0.

3.Высвечивание текущего пиксела (X,Y).

4.Проверка окончания работы: если Y<Yk, то переход к п.11.

5.Анализ значения параметра D: если D<0, то переход к п.6; если D=0, то
переход к п.9; если D>0, то переход к п.7.

6.Вычисление параметра D1=2D+2Y-1 и анализ полученного значения: если D1<0, то переход к п.8;
если D1>0, то переход к п.9.

7.Вычисление параметра D2=2D-2X-1 и анализ полученного значения: если D2<0, то переход к п.9;
если D2>0, то переход к п.10.

8.Вычисление новых значений X и D (горизонтальный шаг): X=X+1; D=D+2X+1. Переход к п.3.

9.Вычисление новых значений X,Y и D (диагональный шаг): X=X+1; Y=Y-1; D=D+2(X-Y+1).
Переход к п.3.
10.Вычисление новых значений Y и (вертикальный шаг):
Y=Y-1; D=D-2Y+1. Переход к п.3.
11.Конец.

При выполнении лабораторной работы данный алгоритм необходимо дополнить с тем, чтобы учитывались вводимые координаты центра окружности и разные разрешающие способности экрана, а также обеспечивалось бы рисование всей окружности.

Учет ненулевых координат центра окружности означает фактически выполнение операции переноса, поэтому координаты точек окружности (X1,Y1) будут вычисляться по формулам: X1=Xc+X; Y1=Yc+Y. Неодинаковые разрешающие способности можно учесть, скорректировав ординату точки окружности путем умножения ее на величину xa/ya (xa, ya - два числа, выдаваемые процедурой
GetAspectRatio модуля Graph, их отношение обратно пропорционально отношению разрешающих способностей экрана).

Лабораторная работа N 3

" Программная реализация алгоритмов заполнения
с затравкой и исследование их временных характеристик" Цель работы: реализовать построчный алгоритм заполнения
гранично-определенной области с затравкой.

Область должна быть с внутренними дырами и зубцами на границе. Предусмотреть возможность ввода координат произвольной
области и любой затравочной точки. Оценить время реализации алгоритма для областей различной сложности.

В алгоритмах затравочного заполнения сплошных областей предполагается, что известна определенная точка (затравка)
внутри области и необходимо определить точки, соседние с затравочной и расположенные внутри области. Если соседняя точка расположена не внутри, значит обнаружена граница области, если же точка находится внутри, то она становится новой затравочной точкой и поиск продолжается рекурсивно.

Известны два алгоритма заполнения с затравкой : простой алгоритм заполнения с затравкой; построчный алгоритм заполнения с затравкой. Данные алгоритмы применимы к гранично-определенным областям, то есть таким, что все пикселы на границе данных областей имеют выделенное значение или цвет , но не один пиксел из внутренней части таких областей не может иметь это выделенное значение (рис.). Гранично-определенные области могут быть 4- или 8-связными (рис.). Любой пиксел в 4-связной области становится доступен с помощью движений только в четырех направлениях: вверх, вниз, направо, налево; для 8-связной же области к любому пикселу можно подойти. используя комбинацию движений в 2-х горизонтальных, в 2-х вертикальных и 4-х диагональных направлениях. Алгоритм заполнения 8-связной области заполнит и 4-связную область (обратное неверно).

Простой алгоритм заполнения с затравкой (базовый алгоритм разработан Смитом, см. [5]) легко реализовать, используя понятие стека с порядком обслуживания " первым пришел, последним обслужен", то есть, когда новое значение помещается в стек, то все остальные опускаются вниз на один уровень, а когда значение извлекается из стека, все остальные поднимаются на один уровень. Такой стек еще называется стеком прямого действия.

Процесс реализации простого алгоритма с затравкой для 4-связанной области состоит в следующем: 1. Затравка выдает затравочный пиксел (x,y), который затем помещается в стек. Стек инициилизируется. 2. Осуществляется проверка на наличие пикселов в стеке. Если стек не пуст, то: 2.1. Пиксел извлекается из стека и ему присваивается требуемое значение:

если пиксел (x,y) < > требуемое значение, тогда пиксел (x,y) = требуемое значение.

2.2. Проводится анализ: надо ли помещать соседние пикселы в стек. Для этого, каждый из соседних к данному 4-связанных пиксела проверяется на два условия: не является ли он граничным или не присвоено ли уже ему требуемое значение: если (пиксел (x+1,y) < > граничное зн-е и пиксел(x+1,y) < > требуемое зн-е),

тогда пиксел (x+1,y) помещается в стек; если (пиксел (x,y+1) < > граничное зн-е и пиксел(x,y+1) < > требуемое зн-е),

тогда пиксел (x,y+1) помещается в стек; если (пиксел (x-1,y) < > граничное зн-е и пиксел(x-1,y) < > требуемое зн-е),

тогда пиксел (x-1,y) помещается в стек; если (пиксел (x,y-1) < > граничное зн-е и пиксел(x,y-1) < > требуемое зн-е),

тогда пиксел (x,y-1) помещается в стек.

То есть, если проверка какого-либо из двух случаев дает положительный результат, то пиксел игнорируется, в противном случае этот пиксел помещается в стек. В приведенном алгоритме 4-связанные пикселы проверяются, начиная с правого от текущего, в направлении против часовой стрелки. Данный алгоритм может заполнять и области, содержащие дыры.

На лабораторную работу выносится реализация построчного алгоритма с затравкой, поэтому расмотрим подробнее построчный алгоритм заполнения с затравкой для 4-связной области (для самостоятельной работы предлагается переделать его для 8-связанной области и заполнение проводить не в 4-х , а в 8-ми направлениях). Гранично-определенная 4-связанная область может быть выпуклой, не выпуклой, а также содержать внутри себя дыры; но во внешней, примыкающей к данной гранично-определенной области, не должно быть пикселов с цветом заполнения.

При инициализации алгоритма в стек прямого действия помещаются координаты введенного затравочного пиксела (x,y). При реализации алгоритм можно разделить на следующие этапы:

Пока стек не пуст:

1. Затравочный пиксел на интервале извлекается из стека, содержащего затравочные пикселы (в построчном алгоритме с затравкой размер стека минимизируется за счет хранения только одного затравочного пиксела для любого непрерывного интервала на сканирующей строке) и ему присваивается требуемое значение:

пиксел (x,y) = требуемое зн-е.

2. Интервал с затравочным пикселом заполняется влево и вправо от затравочной точки вдоль сканирующей строки до тех пор, пока не будет найдена граница:

2.1. Сохраняем координаты абсциссы затравочного пиксела: xt=x.

2.2. Заполняем интервал вправо от затравки: x=x+1, пока (пиксел (x,y) < > граничное зн-е) делать:(пиксел (x,y)=требуемое зн-е; x=x+1);

2.3. В переменной xr запоминаем крайний правый пиксел: xr=x-1.

2.4. Восстанавливаем координату абсцисс затравки: x =xt.

2.5. Заполняем интервал слева от затравки: x=x-1, пока (пиксел (x,y) < > граничное зн-е) делать:(пиксел (x,y)=требуемое зн-е; x=x-1);

2.6. В переменной xl запоминаем крайний левый пиксел: xl= x+1.

2.7. Восстанавливаем координату абсциссы затравки: x=xt.

3. В диапазоне xl <= x <= xr проверяются строки , расположенные непосредственно над и под текущей строкой и на них определяются еще не заполненные пикселы. При обнаружении таких пикселов (если не все пикселы уже заполнены или граничны), крайний правый пиксел в каждом интервале данного диапазона заносится как затравочный в стек. То есть:

3.1. Осуществляем проверку строки, расположенной выше текущей:

x=xl. y=y+1. Пока (x<=xr) ищем затравочный пиксел на строке , выше текущей:

f=0 (промежуточная переменная); пока (пиксел (x,y)< > граничное зн-е и пиксел(x,y) < > требуемое зн-е и x<xr) делать: (если f=0 , тогда f=1; и абсцисса увеличивается на 1: x=x+1); помещаем в стек крайний справа пиксел : если f=1, тогда (если (пиксел(x,y)< >граничное зн-е и пиксел(x,y)< >требуемое зн-е и x=xr), тогда пиксел (x,y) помещается в стек); в противном случае в стек заносится пиксел с координатами (x-1,y);

затем промежуточной переменной присваивается нулевое значение: f=0. Далее проверка не является ли строка выше текущей границей многоуника, или уже полностью заполненной продолжается в том случае, если интервал был прерван: xn=x; пока (пиксел (x,y)=граничное зн-е или пиксел(x,y)=требуемое зн-е и x<xr) делать : x=x+1;

убедимся, что координата абсциссы пиксела увеличилась: если x=xn, тогда x=x+1.

3.2. Аналогично пункту 3.1. осуществляем проверку строки ниже текущей, изменения лишь касаются координаты y, ей присваивается значение : y=y-1. И так до тех пор, пока стек не становится пустым, область заполненной , а работа алгоритма завершенной. Если гранично-определенная область примыкает к краю экрана, то при реализации алгоритмов заполнения с затравкой возможен выход за пределы области, чтобы этого не случилось рекомендуется: - либо на каждом шаге производить проверку, не превзойдены ли

пределы памяти; - либо сразу же увеличивать пределы памяти на краях, для исключения возможности появления ошибки.

В следущих лабораторных работах рассматриваются алгоритмы заполнения сплошных областей в порядке сканирования (растровой развертки) дисплея. В этих алгоритмах в порядке сканирования строк определяется, лежит ли точка внутри области, ограниченной ребрами многоугольника или нет , поэтому процесс реализации таких алгоритмов начинается от верхней вершины верхнего ребра многоугольника и идет к нижнему пикселу нижнего ребра. Информация о нахождении текущей точки внутри или снаружи заполняемой точки легко получается с помощью " контроля четности" , то есть подсчета числа пересечений ребер многоугольника, ограничивающих закрашиваемую область, со строкой сканирования, на которой находится эта точка (рис. 3.3.), тт. V1, V2 - лежат внутри области, так как число пересечений нечетное, а тт. V3,V4 - расположены вне закрашиваемой области, ибо число пересечений четное.

Лабораторная работа N 4

" Программная реализация алгоритмов заполнения
сплошной области с упорядоченным списком ребер"

Цель работы: заполнить произвольно введенную область, используя алгоритм с упорядоченным списком ребер. Определить время заполнения области и сравнить с построчным затравочным алгоритмом.

Это быстродействующий алгоритм, суть которого состоит в сортировке в порядке сканирования точек пересечений ребер многоугольника, ограничивающего заданную область, со сканирующими строками. Процесс реализации алгоритма можно разбить на два этапа: подготовка (сортировка) данных и преобразование отсортированных данных в растровую форму.

При реализации данного алгоритма большое время уходит на подготовку данных, для этого необходимо определить для каждого ребра многоугольника точки пересечений со строками сканирования, проведенными через середины интервалов, используя уже реализованные алгоритм Брезенхема или цифровой дифференциальный анализатор. Горизонтальные ребра не могут пересекать сканирующую строку и , следовательно, игнорируются. Но при синтезе изображения они, естественно, присутствуют , ибо формируются верхней и нижней строками пикселов. Каждое пересечение (x, y+1/2) заносится в список. Затем список отсортировывается по строкам и по возрастанию абсциссы в строке (то есть точка с координатами (x1,y1) будет предшествовать точке с координатами (x2,y2) в том случае, если y1> y2 или при равных ординатах точек - x1 <=x 2.

После подготовки данных, они преобразуются в растровую форму, для этого из отсортированного списка выделяются пары точек (x1,y1) и (x2,y2) (они соответствуют условию предыдущего абзаца) и на сканирующей строке инициализируются точки с целыми значениями x, которые соответствуют неравенству: x1 <= x+1/2 <= x2. При реализации данного алгоритма дополнительные трудности возникают при пересечении сканирующей строки и многоугольника точно по вершине (рис.4.1). При использовании соглашений о середине интервала между строками сканирования возможен случай, когда получится нечетное количество пересечений. и, следовательно, разбиение пикселов на пары даст неверный результат.

Правильный результат в этом случае получается, если учитывать точку пересечения в вершине два раза в том случае, если она является точкой локального минимума или максимума, и один раз в противном случае. Локальный максимум или минимум многоугольника в иссследуемой вершине определяется с помошью проверки концевых точек ребер, соединенных в данной вершине. Если у обоих конечных точек ординаты у больше, чем у вершины, значит вершина является точкой локального минимума. Если одна больше, а другая меньше, значит вершина - точка локального максимума. Если же одна ордината больше, а другая меньше, значит вершина не является ни точкой локального минимума, ни точкой локального максимума (на рис. 4.1. т. V1 - локальный максимум, V3 - локальный минимум, тт. V2,V4 не являются ни локальным минимумом , ни локальным максимумом; то есть при пересечении в тт. V1
, V3 учитывается два пересечения со строками сканирования, а в тт. V2 и V4 - одно).

Вышеописанный алгоритм с упорядоченным списком ребер требует формирования большого списка точек пересечений, который должен быть полностью отсортирован. Поэтому разработан более эффективный алгоритм с упорядоченным списком ребер, в котором процесс сортировки разделяется по строкам в направлении y и в строке в направлении x с помощью групповой сортировки по y .

При подготовке данных аналогичным образом определяются точки пересечений ребер многоугольника со строками сканирования, затем координата x точки пересечения помещается в группу, соответствующую y. Для каждой такой y-группы отсортировываются точки пересечений в порядке возрастания координаты абсцисс, то есть x1 предшествует в отсортированном списке x2 , если x1 <= x2. Затем отсортированные данные преобразовываются в растровую форму, то есть для каждой строки сканирования выделяются из

списка абсцисс точек пересечений пары точек пересечений и на соответствующей строке сканирования - у активизируются пикселы для их целых значений (как и в предыдущем случае).

Таким образом, модифицируя простой алгоритм с упорядоченным списком ребер мы сначала с помощью групповой сортировки по оси ординат проводим сортировку в порядке сканирования строк, а затем сортируем в строке по оси абсцисс, то есть начинаем процесс развертки до окончания сортировки. Как видно из алгоритма, в нем легче добавлять или удалять информацию из сортировочного списка, а следовательно, и дисплейного файла: необходимо только добавить или удалить информацию из соответствующей у-группы и пересортировать только измененные строки.

Несмотря на то, что при использовании у-групп задача сортировки упрощается, при реализации модифицированного алгоритма с упорядоченным списком ребер требуется резервировать большое количество памяти (значительная часть которой не будет использована) или ограничивать число пересечений с данной сканирующей строкой. Эти недостатки легко преодолеваются благодаря введению добавочной структуры данных - связанного списка активных ребер (САР).

В этом случае значительно сокращается потребность в памяти, а точки пересечения со строками сканирования вычисляются в пошаговом режиме. В процессе подготовки данных в алгоритме с упорядоченным списком ребер, использующем список активных ребер необходимо определить для каждого ребра многоугольника наивысшую сканирующую строку, пересекаемую данным ребром. Это удобно сделать анализируя строки сканирования, проведенные через середины отрезков (через у+1/2). Затем ребро многоугольника сохраняется в у-группе, соответствующей этой сканирующей строке и формируется связанный список, в который заносятся следующие значения: - начальное значение абсцисс точек пересечения -х; - число сканирующих строк, пересекаемых ребром многоугольника - у; - шаг приращения по оси абсцисс при переходе от одной строки сканирования к другой - х.

В процесе преобразования подготовленных данных в растровую форму для каждой строки сканирования осуществляется проверка соответствующей у-группы на наличие новых ребер, в случае их обнаружения, соответствующие значения заносятся в САР. После чего координаты абсцисс точек пересечения из связанного списка отсортировываются в порядке возрастания (то есть x1 <= x2) и из него выделяются пары точек, с помощью которых активизируются пикселы на строке сканирования для целых значений х аналогично двум предыдущим случаям.

Затем для каждого ребра из САР число сканирующих строк, пересекаемых данным многоугольником - у, уменьшается на 1. Если в результате, у становится меньше 0, то данное ребро исключается из САР и вычисляется новое начальное значение координаты абсцисс точек пересечения xn = x + x. Затем переходят к новой строке сканирования и этапы реализации алгоритма повторяются. Так как в данном алгоритме минимизированы операции ввода/вывода, то при реализации можно сделать его независящим от устройств.

Лабораторная работа N 5

" Программная реализация алгоритмов заполнения сплошных областей по ребрам и оценка их временных характеристик"

Цель работы: Заполнить произвольно введенную область , используя алгоритм заполнения по ребрам. Определить время заполнения области и сравнить с временем реализации алгоритма с упорядоченным списком ребер и построчным затравочным алгоритмом.

В алгоритме заполнения по ребрам в отличие от предыдущего не треуется создавать и сортировать списки различных данных. Он очень прост: при его реализации необходимо для каждой строки сканирования, пересекающей ребро многоугольника, ограничивающего закрашиваемую область, в точке с координатами (x1,y1), активизировать все пикселы, которые лежат справа от (x1,y1) и для которых справедливо неравенство : x+1/2 > x1.

К каждому ребру многоугольника алгоритм применяется индивидуально. Пересечения строк сканирования осуществляется аналогично алгоритму с упорядоченным списком ребер, то есть определяются точки пересечений со сканирующими строками, проведенными через середины интервалов (y+1/2). Рекомендуется использовать этот алгоритм совместно с дисплейным файлом, что позволяет выбирать и обрабатывать ребра в любом порядке. Ибо тогда при обработке ребра многоугольника обрабатываются пикселы из дисплейного файла, соответствующие точкам пересечения ребра со строкой сканирования.

После обработки всех ребер многоугольника, ограничивающего закрашиваемую область, дисплейный файл выводится в порядке сканирования на экран графического дисплея. Данный алгоритм характеризуется существенными недостатками: - многократной обработкой одного и того же пиксела в случае областей закрашивания сложной формы; - зависимостью алгоритма от операций ввода/вывода.

Для сокращения числа обрабатываемых пикселов используется "перегородка". В этом случае алгоритм модифицируется следующим образом: анализируется каждая точка пересечения каждого ребра многоугольника со строкой сканирования.

Если текущая точка пересечения находится слева от перегородки, то активизируются все пикселы, центры которых лежат справа от точки пересечения ребра со строкой сканирования и слева от перегородки. Если же точка пересечения находится справа от перегородки, то активизируются пикселы, центры которых расположены слева или на пересечении строки сканирования с ребром многоугольника и справа от перегородки (рис. 5.1.)

Рекомендуется проводить перегородку через одну из вершин многоугольника и реализовывать данный алгоритм совместно с использованием дисплейного файла. Недостатком алгоритма заполнения с перегородкой все же остается неоднократная обработка части пикселов.

Для того , чтобы избавиться от этого, разработан модифицированный алгоритм заполнения сплошной области со списком ребер и флагом. Алгоритм со списком ребер и флагом - двухшаговый алгоритм. Во-первых, обрисовывается контур, ограничивающий область, в результате чего на каждой строке сканирования определяются пары ограничивающих пикселов; а во-вторых, активизируются пикселы, расположенные между вычисленными на предыдущем шаге ограничивающими пикселами.

Процесс реализации алгоритма со списком ребер и флагом состоит из следующих этапов:

1. Обработка ребер многоугольника, ограничивающего заполняемую область. Также, как в предыдущих алгоритмах, считаем, что строки сканирования проходят через центр строк пикселов, то есть через середину интервала y+1/2. После определения пересечений вычисляем самый левый пиксел, центр которого лежит справа от точки пересечения, у которого x+1/2 больше абсциссы точки пересечения.
2. Заполнение области. Для каждой строки сканирования, имеющей точки пересечения с многоугольником, ограничивающем область выполняем следующую последовательность действий: f=0, если пиксел находится вне области (f-промежуточная переменная); x=xl (xl - левая граница); пока (x<=xr) (xr - правая граница) делать (если пиксел (x,y) =граничное зн-е , тогда присвоить f=1; если f=1, тогда присвоить пикселу (x,y) цвет многоугольника, в противном случае, присвоить пикселу (x,y) цвет фона; x=x+1).

Иллюстрация алгоритма заполнения по ребрам и флагу приведена на рис. 5.2. Расссмотрим лишь выделенную строку сканирования. Для обрисовки контура на этой строке инициилизируются пикселы с абсциссами 2,5,6,9.

При заполнении получаем следующие результаты: f=0; x=0, x=1 - пикселы не граничные и f=0, следовательно, ничего не происходит; x=2 - пиксел граничный, следовательно f=1 и пикселу присваивается интенсивность многоугольника; x=3, x=4 - пикселы не граничные и f=1, следовательно, им присваивается цвет многоугольника; x=5 - пиксел граничный, следовательно, f=0 и пикселу присваивается фоновый цвет; x=6 - пиксел граничный, следовательно, f=1 и пикселу присваивается цвет многоугольника; x=7, x=8 - пикселы не граничные и f=1, следовательно им присваивается цвет многоугольника; x=9 - пиксел граничный, следовательно, f=0 и пикселу присваивается цвет фона.

При реализации данного алгоритма каждый пиксел обрабатывается только один раз, следовательно временные затраты на выполнение операций ввода/вывода значительно меньше, чем в простом алгоритме заполнения по ребрам или в алгоритме с перегородкой. Существует аппаратная реализация алгоритма со списком ребер и флагом, в этом случае его скорость выполнения увеличивается на несколько порядков по сравнению с упорядоченным списком ребер (при программной реализации скорость выполнения алгоритмов отличается незначительно, в чем предлагается убедиться при выполнении Лабораторной работы).

При реализации данного алгоритма возможен даже синтез несложных изображений в режиме реального времени. Ни один из алгоритмов, описанных в данной Лабораторной работе не требует создания и сортировки списков в том случае, если используется дисплейный файл.