12.02.14-----------------------------------------------------------------------

Куров Андрей Владимирович, 517л

Компьютерная графика – совокупность методов и средств преобразования информации в графическую форму и из графической с помощью ЭВМ. Методы – математические и алгоритмы. Средства – технические и программные. Компьютерная (вычислительная) геометрия.

Любой геометрический объект может быть задан несколькими способами. Окружность – три точки; координаты центра и радиус. Параметрическое число геометрического объекта – минимальное количество параметров, задающих этот объект.

Прямая – 3 (Ax+By+C=0)

Прямоугольник – 5 (концы диагонали и угол)

Эллипс – 5 (центр, a,b, угол; или вписывая в прямоугольник)

**Об аппроксимации кривых**

Рассмотрим окружность, возьмём две точки. Минимальное расстояние между ними после округления должно быть >=1, иначе точки начнут сливаться. Пусть угол между радиусами этих точек равен , . При достаточно большом радиусе кривизны кривой, две соседние точки кривой должны выбираться таким образом, чтобы величина угла (выраженная в радианах) между радиусами этих точек был не меньше чем 1/радиус. Радиус кривизны . Если прямая задана параметрически, х(т) и у(т), то . В полярных координатах   
Если взять две точки на кривой, провести через М1 касательную, то радиус кривизны - , где w – угол между касательной и осью абсцисс.

Для эллипса можно выделить два участка; граничной является точка, в которой . На участке 0..х1 Х изменяется с шагом 1 и вычисляется У, на участке х1..х2 У изменяется с шагом 1 и вычисляется Х.

**Преобразование изображений на плоскости.**

Отличие фрагментов: 1. Местоположение и радиус, 2. Пропорции, 3. Ориентация.

При линейном преобразовании координаты преобразованной точки линейно зависят от координат исходной: x1=Ax+By+C, y1=Dx+Ey+F. А..Ф – параметры, однозначно определяющие преобразование.

Аффинные преобразования – при которых  
плоскость не вырождается в линию или точку,  
сохраняется параллельность прямых,  
всегда есть обратное преобразование.

Для описания преобразований часто используется матричная форма. Однако в ней нельзя описать, например, перенос: приходится использовать однородные координаты [X,Y,W], где w – масштабный коэффициент. В двумерной графике W=1. Декартовы координаты .  
Линейное преобразование в матричной форме: . Любое сколь угодно сложное преобразование может быть описано совокупностью трёх – переноса, масштабирования и поворота.

1. **Перенос на плоскости**: задается смещениями по Х и У. X1=x+dx, y1=y+dy. Матрица переноса .

19.02.14-----------------------------------------------------------------------

2. **Масштабирование** – изменение пропорций\размеров изображения. Задается четырьмя параметрами: координаты центра масштабирования, коэффициент по Х, коэффициент по У. Все параметры имеют вещественный тип. Центр может располагаться в любой плоскости, проходящей через плоскость экрана.

Однородное масштабирование – коэффициенты одинаковые, неоднородные – с разными коэффициентами.

Центр Xm,Ym, точка X,Y, новая точка X1,Y1, коэффициенты Kx,Ky,

X1 – Xm = Kx (X-Xm)

Y1 – Ym = Ky (Y-Ym)

Таки образом, К.Новая = К.Центра + Коэффициент\*(К.Старая – К.Центра)

В матричной форме, для случая Xm=Ym=0:

. В общем случае – вначале мы проводим ПЕРЕНОС, чтобы центр масштабирования совпал с центром коодинат, затем провести масштабирование, затем обратный перенос: .

Если коэффициент меньше 1 – изображение уменьшается, точки перемещаются ближе к центру масштабирования. Если больше – увеличиваются, точки перемещаются дальше. Если коэффициент меньше 0 – изображение заодно отзеркаливается.

3. **Поворот** – задается тремя параметрами: центр поворота, и углом поворота.

Центр Xc,Yc, точка X,Y, новая X1,Y1, О – угол на который поворачиваем, Ф – исходный угол относительно оси абсцисс. Х1 =

. Аналогичными преобразованиями,

.

В матричной форме: поворот относительно начала координат . Определитель =1 – для любого поворота существует обратный поворот.

**Основные свойства преобразований**

Коммутативность – независимость результата преобразований от порядка, в котором они происходят. (X1 Y1 1) = (X Y 1)\*M1\*M2 – если М1 можно умножить на М2, то в ОБЩЕМ случае, М2 умножить на М1 нельзя – в общем случае, преобразование коммутативностью не обладает.

Коммутативные преобразования:

|  |  |
| --- | --- |
| перенос | Перенос |
| масштаб | Масштаб |
| поворот | Поворот |
| Однородное масштабирование | Поворот |

Доказательство коммутативности производится через перемножение матриц.

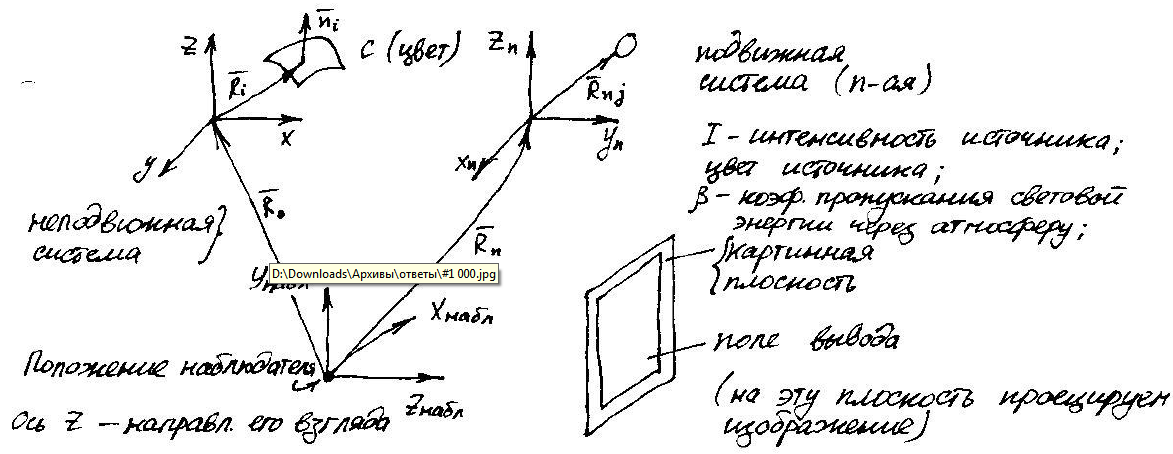
Аддитивные и мультипликативные операции – типа сложение и умножение.

Аддитивные: перенос-перенос и поворот-поворот. Чтобы найти итоговую матрицу такого поворота, достаточно сложить аргументы.

Мультипликативные: перемножить аргументы,

**Синтез сложного динамического реалистичного изображения**

Постановка задачи.



Зададим *подвижную* систему координат (У вверх). В ней зададим поверхность, задав вектор Ri некой центральной точки поверхности, и ориентацию поверхности, задав вектор нормали Ni этой точки. Поверхность обладает цветом Ci.

Поскольку изображение является реалистичным, необходимо передать различные оптические эффекты – отражение света (диффузное и зеркальное), его пропуск, затенение. Задаётся коэффициент зеркального Кзi и диффузного Кдиффi отражения, коэффициент пропускания Кпрi.

Далее задаётся *подвижная* система координат, связанная с неким объектом (радиусвектор Подi). Вектор Rпод идёт из центра неподвижной в центр подвижной системы.

Третья система координат связывается с наблюдателем. Начало её совмещается с положением точки наблюдения, Z направлена по линии взгляда, и изображение строится на картинной плоскости Хнабл:Унабл.

Также присутствуют источники освещения Iистi. Они задаются координатами Xистi, Yистi, Zистi, цветом (интенсивностью каждой из трех составляющих)

Также необходимы условия среды – коэффициент пропускания.

Для каждого динамического объекта требуется указать управляющие операторы, на основе которых может быть вычислено его положение для дискретных моментов времени, разделенных интервалами .

**Этапы синтеза**

1. *Разработка (трехмерной) математической модели синтезируемой визуальной обстановки. Представление поверхности: Аналитическое описание или полигональная модель или т.п.*
2. *Определение положения наблюдателя в 3мерном пространстве, направления линии взгляда и положения экрана, размеров окна обзора, значения управляющих сигналов (для динамических объектов).*
3. *Формирование операторов, осуществляющих пространственное перемещение динамических моделируемых объектов*
4. *Преобразование координат объектов в неподвижной системе в координаты в системе наблюдателя.*
5. *Отсечение объектов сцены (в том числе ФРАГМЕНТОВ объектов) в пределах пирамиды видимости (удаление того, что находится за пределами видимой области).*
6. *Нахождение двумерных перспективных проекций объектов сцены на картинную плоскость.*
7. *Исключение невидимых объектов сцены при заданном положении наблюдателя. Закрашивание и затенение видимых участков сцены.*
8. *Вывод полученного полутонового изображения на экран растрового дисплея.*

26.02.14-----------------------------------------------------------------------

Алгоритмы нижнего уровня -> Алгоритмы среднего уровня (построение плоских изображений) -> Алгоритмы верхнего уровня (удаление невидимого, построение реалистичного изображения)

**Растровая графика**

**Растровая развертка отрезка (разложение в растр)**

Процесс определения пикселей, наилучшим образом аппроксимирующих заданный отрезок. Простейшие случаи – горизонтальный, вертикальный и под 45°. В большинстве случаев проявляется лестничный (ступенчатый) эффект.

*Общие требования:*

1. Отрезок должен выглядеть как отрезок прямой, начинаться и заканчиваться в заданных точках
2. Интенсивность (яркость) вдоль отрезка должна быть постоянной. Отрезки, имеющие разные углы наклона, должны быть одной интенсивности. Восприятие человека зависит не только от интенсивности свечения объекта, но и от расстояния между светящимися объектами //чтобы удовлетворить этому требованию, надо высвечивать точки с переменной интенсивностью от расстояния – потребует дополнительных вычислений, без особой нужды не используется
3. Алгоритмы (особенно нижнего уровня) должны работать быстро

Все алгоритмы имеют пошаговый характер – на очередном шаге высвечиваем пиксель, и производим вычисления, используемые в следующем шаге.

**Алгоритм цифрового дифференциального анализатора**

Если прямая задана каноническим уравнением , то производная . Если заданы начальная и конечная точки х1у1 и х2у2, то . , а – высчитывается и округляется. Однако следует выбирать не всегда – вначале нужно ответить на вопрос, как наклонён отрезок. Если угол наклона <45, то |dX|=1, если > то |dY|=1

1. Начало
2. Ввод хн, ун, хк, ук
3. Проверка отрезка на вырожденность: если вырожденный то высветить точку и переход на конец
4. Если |dx|>|dy|, то L=|dx| иначе L=|dy|
5. Построение отрезка (цикл по i=1 до L+1)
   1. Высвечивание точки E(xt),E(yt)
   2. Вычисление координат следующей точки xt += dx; yt += dy;
6. Конец

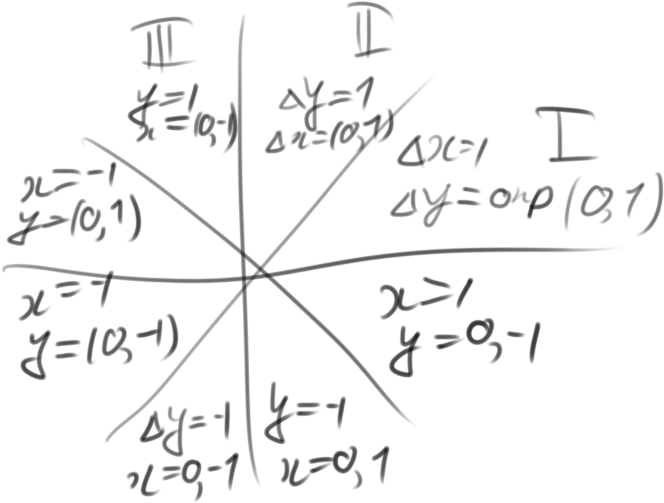
Недостатки – работает с целочисленной арифметикой (координаты текущей точки). Работает медленнее за счет операции округления.

**Алгоритмы Брезенхема**

Ошибка еi – величина изменяющаяся на каждом шаге , расстояние между точкой самого отрезка и точкой, аппроксимирующей его на очередном шаге. , если е<0.5, то ордината пикселя не меняется, Если >0.5, то выбирается верхний пиксель.. Начальное значение ошибки e=m, где м – тангенс угла наклона, а . – здесь ордината идеального отрезка, в вещественных а не целых координатах.

На i+1 шаге . В таком случае вычисляем предварительное . Если на очередном шаге мы корректируем значение у, то мы должны также скорректировать и ошибку.

Допускается начальное e=m-0.5 и дальше сравнивать ошибку с нулем, а не с 0.5



Алгоритм Брезенхема:

1. Начало
2. Ввод хн, ун, хк, ук
3. Проверка отрезка на вырожденность: если вырожденный то высветить точку хн,ун и переход на конец
4. Sx=sign(dx), sy=sign(dy)
5. Dx=|dx|, dy=|dy|
6. Если dx>dy то обмен=0, иначе обмен=1 {t=dy; dy=dx; dx=t}
7. .
8. Цикл построения отрезка (по i=1 to dx+1)
9. Конец

05.03.14-----------------------------------------------------------------------

Алгоритм Брезенхема:

1. Начало
2. Ввод хн, ун, хк, ук
3. Проверка отрезка на вырожденность: если вырожденный то высветить точку хн,ун и переход на конец
4. Sx=sign(dx), sy=sign(dy)
5. dx=|dx|, dy=|dy|
6. Если dx>dy то обмен=0, иначе обмен=1 {t=dy; dy=dx; dx=t}
7. ; **(1)**. //тангенс угла наклона
8. Цикл построения отрезка (по i=1 to dx+1)
   1. Высвечивание точки (xt,yt)
   2. Если (e>=0) {  
       если (обмен=0) {  
       yt=yt+sy  
       } иначе {  
       xt=xt+sx  
       }  
       e=e-1 **(2)**} иначе {  
       если (обмен=0) {  
       xt=xt+sx  
       } иначе {  
       уt=уt+sy  
       }  
       e=e+m **(3)**}
9. Конец

Недостатки: не все переменные являются переменными целого типа (e,m - действительные).

Чтобы перейти к алгоритму, работающему ТОЛЬКО с целыми:

(1). Тогда в цикле в (2) будет e=e-2dx; в (3) e=e+2dy; не нужно вычислять m в 9м шаге.

Брезенхем предложил простейший алгоритм сглаживания; алгоритм Брезенхема с устранением ступенчатости.

Используется при отображении ребёр многоугольника, который закрашивается. Идея состоит в сглаживании резких переходов от ступени к ступени. Сглаживание основывается на том, что каждый пиксель высвечивается со своим уровнем интенсивности. Уровень выбирается пропорционально площади части пикселя. 1 пиксель – квадрат с единичной стороной, а не математическая точка.

Так как интенсивность I~Si площади, то

1) отрезок связан (покрывает) на i шаге с одним пикселем. Обозначим Yi расстояние по вертикали от точки пересечения отрезка с пикселем, до левой нижней границы пикселя. Обозначим тангенс угла наклона отрезка через m, тогда Si=Sпр+Sтр = Yi\*1 + 1\*m/2 = Yi + m/2

2) отрезок покрывает на I шаге два пикселя. Yi – расстояние от нижней границы до пересечения с отрезком. Площадь нижней части S1 = Sпикс – Sтр1 = . Площадь части второго пикселя S2 . Складывая площади, S1+S2 = Yi + m/2

1) На очередном шаге Si = Yi + m/2; S(i+1)=Sпр1 + Sпр2 + Sтр = Si + m

2) S(i+1) = Si + m, но высвечивается верхний пиксель, нижний – не высвечивается, и его площадь не учитывается. S = S+m; если , то выражение корректируется, если , то S=S-1 (вычитаем площадь нижнего пикселя)

В данном случае в качестве ошибки можно рассматривать ei=Si (интенсивность пропорциональна ошибке) – однако её нельзя будет скорректировать через -0.5. За пороговый уровень можно взять другое значение. Обозначим w=1-m, e = e + w = m-0.5 + 1-m = 0.5. I=Imax/2 – начальный пиксель всегда высвечивается в половинной интенсивности.

W – пороговое значение. Если e>=w, то (e=e-w), иначе Y(i+1)=Yi.  
Чтобы внутри цикла не приходилось постоянно умножать на интенсивность, сделать это можно один раз в начале работы алгоритма (m=Imax\*m, e=Imax\*e, w=Imax-m).

При реализации алгоритма без сглаживания исходные данные – начальные и конечные координаты. Здесь же добавится количество уровней интенсивности, либо же максимальный уровень.

**Алгоритм Брезенхема для построения окружности**

Будем считать, что центр находится в начале координат, хс=ус=0. У направлена вверх, Х вправо.

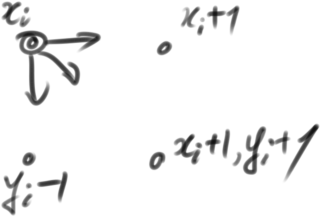
Окружность – симметричная фигура, можно построить половину или четверть и отражать и поворачивать. Y=A(x) (отрисовка четвертькруга от (0,R) до (R,0)) – в первой четверти монотонно убывающая. На очередном шаге возможен переход из (xi,yi) в (xi+1,yi), (xi,yi-1), (xi+1,yi-1).

Критерий – местный аналог ошибки. Разность квадратов (расстояния от центра окружности до диагонального пикселя) и (расстояния от центра окружности до самой окружности).  
Если >0, то диагональный пиксель лежит вне окружности. Выбирается диагональный или вертикальный пиксель для отрисовки.  
Если =0, то диагональный пиксель лежит ровно на окружности. Выбирается диагональный.  
Если <0, то диагональный пиксель лежит внутри окружности. Выбирается диагональный или горизонтальный.

12.03.14-----------------------------------------------------------------------

**Алгоритм Брезенхема для построения окружности**

Будем считать, что центр находится в начале координат, хс=ус=0. У направлена вверх, Х вправо.

Окружность – симметричная фигура, можно построить половину или четверть и отражать и поворачивать. Y=A(x) (отрисовка четвертькруга от (0,R) до (R,0)) – в первой четверти монотонно убывающая. На очередном шаге возможен переход из (xi,yi) в (xi+1,yi), (xi,yi-1), (xi+1,yi-1).

Критерий – местный аналог ошибки. Разность квадратов (расстояния от центра окружности до диагонального пикселя) и (расстояния от центра окружности до самой окружности).

Если , то диагональный пиксель лежит внутри окружности. Выбирается диагональный.

Случай1: первый модуль >=0, второй <0.

Случай2: первый модуль <=0, второй <0, выбор пикселя очевиден, . , где 0<=yi<=R  
Если , то диагональный пиксель лежит ровно на окружности. Выбирается диагональный.   
Если , то диагональный пиксель лежит вне окружности. Выбирается диагональный или вертикальный пиксель для отрисовки.

. Если <0, то выбирается диагональный, если =0 – любой из двух, >0 – вертикальный.

Случай4: бельта2 =0, выбор пикселя (xi,yi-1) очевиден, но надо проверить знак бельта2.

Для случая (случай5) выбор очевиден, но надо проверить знаки бельта1 и бельта2 на предмет отсутствия противоречий.

надо выражать через

Горизонтальный шаг:

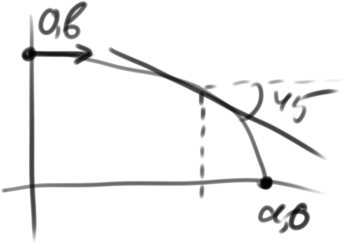
Диагональный шаг:

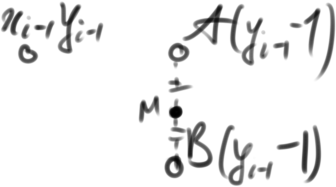
Вертикальный шаг:

Начальное значение

**Алгоритм Брезенхема для построения эллипса**

. Можно использовать операцию переноса центра, тогда . Учесть ненулевые координаты центра можно с помощью

 В этой точке . На первом интервале – единичное приращение по х, приращение по у определяется. На втором – наоборот.

1 интервал: А и В – альтернативные пиксели, М – точка между ними. Зная, как проходит дуга эллипса относительно средней точки, можно выбрать пиксель А или В.

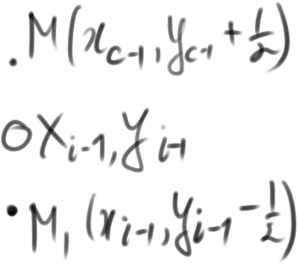
Случай1: средняя точка внутри эллипса. Дуга эллипса проходит между точкой А и М => выбрать для отрисовки нужно пиксель А.

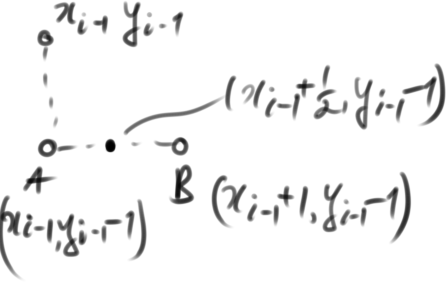
Случай2: средняя точка вне эллипса. Дуга эллипса проходит между точкой М и В => выбрать нужно пиксель В.

Таким образом, осталось определить положение точки М относительно эллипса. Значения ХУ обращающие в равенство – принадлежат эллипсу. Пробная функция: Вычислим через предыдущее значение. Изменение

В итоге можно отказаться не только от возведения в квадрат, но и от умножения. В итоге , где b2=b\*b, bd=2b2

Если на очередном шаге была выбрана нижняя точка В, то необходимо скорректировать значение пробной функции, поскольку на последующем шаге вычисления должны проводиться для точки М1, лежащей ниже этой В.

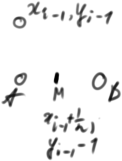


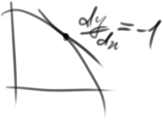
2 интервал: 

Случай 1: точка М лежит вне эллипса (эллипс проходит между А и М), выбираем А.

Случай2: М лежит внутри эллипса (эллипс проходит между М и В), выбираем В.

19.03.14-----------------------------------------------------------------------



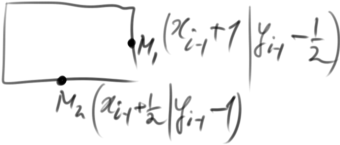


*.*

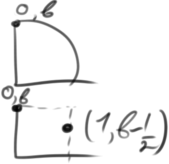
. A4y2=b4x2 (1), отсюда

. Отсюда имеем , т.е.

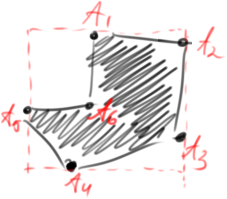
Необходимо также сделать вторую коррекцию на границе интервалов. Сделаем её однократно, в отличие от предыдущей, которая выполнялась двукратно.



Наконец надо вычислить функцию в начальной точке.



**Растровая развёртка сплошных областей (заливка).**

Генерация сплошных областей на базе простых описаний вершин или рёбер многоугольника.

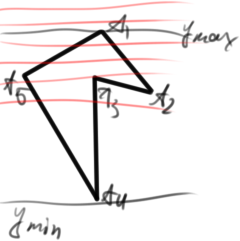
1 границы можно не отрисовывать; принадлежат внутренней области

2 Алгоритм, справляющийся с произвольным многоугольником (выпуклые/невыпуклые; внутри могут содержаться границы

Фактически, задача сводится к определению принадлежности каждого пикселя внутренней области, ограниченной прямоугольник. Это определяется путём «запуска вектора». Если он пересекает прямоугольник дважды – это закрашенная область, единожды – вершина.

Условия: xmin <= x <= xmax && ymin <= y <= ymax.

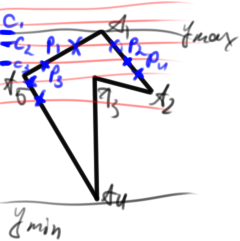
Если область, которую нужно закрасить, значительно меньше прямоугольника граничных координат, то КПД простейшего алгоритма чрезвычайно мал.

Все алгоритмы заполнения можно разделить на две группы – растровые и затравочные.

В растровом алгоритме для каждой строки каждый раз решается задача определения, лежит пиксель внутри многоугольника или нет. Рассматриваются сканирующие строки, пересекающие многоугольник.

Для пикселей каждой сканирующей строки определяется их принадлежность многоугольнику.

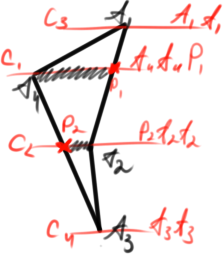
Уменьшить трудоёмкость задачи можно если принять во внимание факт того, что пиксели, расположенные друг рядом с другом, сохраняют неизменным «свойство принадлежности».



С1: [A1..A1]

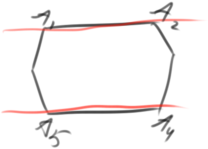
C2: [P1..P2]

C4: [P3..A3] [A3..P4]

Используя подход, связанный с учетом когерентности пикселей (одинаковые свойства рядом расположенных пикселей), проблема возникает только для строк, проходящих ровно через вершины многоугольника. Проблема: для одних строк количество пересечений надо принимать ==2, а для других ==1

Если в примере выше взять ребра А1А5 и А1А2, то у1>у5 && у1>у2. Аналогично, y4<y5, y4<y3. А вот для пятой вершины, y5<y1 и y5>y4 – точка А5 не является точкой экстремума; в то время как А1 и А

26.03.14-----------------------------------------------------------------------

Вторая проблема – с горизонтальными рёбрами. Ребро лежит на сканирующей строке – огромное число «пересечений». Решение – выкинуть ребро. Тогда сканирующая строка всёравно будет пересекаться с А1.. но уже как с концами других рёбер – это не значит, что они не будут закрашены.

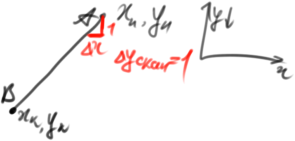
Два тонких момента – горизонтальные ребра (можно не рассматривать) и учет сканирующих строк, проходящих ровно через вершины многоугольника.

Рассмотрим конкретные алгоритмы.

**Алгоритм заполнения с упорядоченным списком ребер**

Активное ребро – ребро, которое в данный момент рассматривается. Ребро, пересекаемое текущей сканирующей строкой. Список активных ребер – список, хранящий информацию об активных рёбрах.

Алгоритм для нахождения точек персечения (прошлая лекция) использовать нельзя – отрезок представлен в виде совокупности ступенек, а значит вместо одной точки пересечения может получиться несколько, и результат заполнения будет определяться четностью\нечетностью количества точек. Необходимо сочетать два подхода – разлагая отрезок в растр надо получать одну точку пересечения для сканирующей строки. – информация об отрезке.



dX – фактически котангенс угла наклона. . . Когда dY <0 – отрезок обработан.

Простой алгоритм:

1. найти пересечение всех отрезков со всеми сканирующими строками (х1,у1), ... (хк,ук).

2. найденные точки персечения упорядочиваем по убыванию координаты У

3. сортировка точек пересечения, принадлежащих одной (каждой) сканирующей строке по возрастанию Х. (х1,у)...(хн,у), х1<=х2<=...хн

4. упорядоченный список для каждой сканирующей строки разбить на пары и закрасить все пиксели, расположенные в интервале, ограниченном очередной парой.

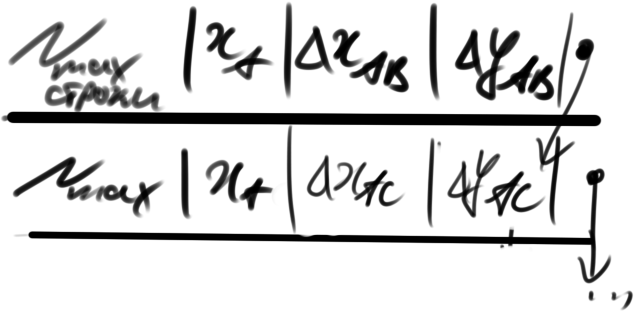
Более эффективный вариант – необходимо избавиться от сортировки всего множества точек пересечения. Для хранения точек в простейшем случае можно использовать массив, и разбить его на n участков, по количеству сканирующих строк.

1. находимые точки пересечения сразу заносятся в нужную группу

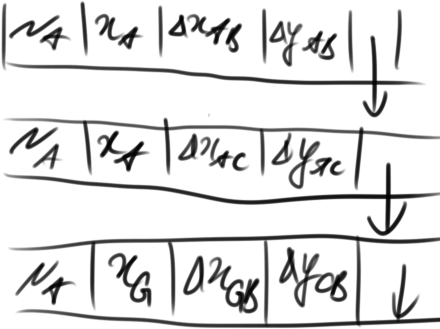
2. упорядочить содержимое каждой у-группе по возрастанию Х

3. образовать пары внутри каждой у-группы и закрасить все пиксели в интервалах.

При этом использовать массивы крайне невыгодно – при статике будет перерасход памяти, при динамике требуется копирование элементов при перевыделении. Можно использовать однонаправленные списки.



Информацию о ребрах в список надо заносить в порядке убывания максимального значения координаты У ребра.

AB, AC, GB, GF, CD, EF, ED. 

Если N текущее > N первого элемента списка, то список активных ребер пуст, многоугольник «ещё не начался». Если N текущее <= N первого элемента списка, то строка пересекает многоугольник (САР не пуст).

Список просматривается до тех пор, пока выполняется условие N тек. <= N очередного элемента списка. При просмотре списка  
1) выбираем Xij абсциссы точек с текущей строкой.  
2) корректируем поля элемента списка: . Если dy<0, то элемент списка удаляется из списка. Удалив обработанное ребро, избавляемся от необходимости просмотра уже просмотренных ребер.  
3) обработка ведется пока список не опустеет

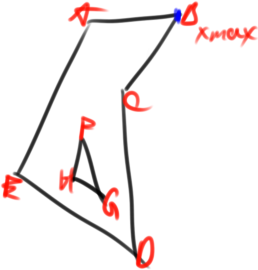
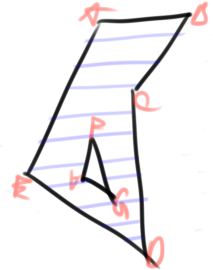
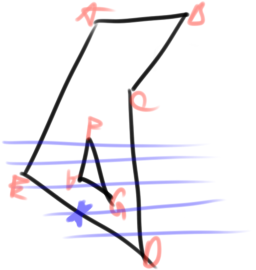
Сформированный список абсцисс точек пересечения ребер с очередной сканирующей строкой надо опять отсортировать по возрастанию и закрасить пиксели, расположенные внутри каждого интервала.

Алгоритм заполнения принято оценивать количеством операций, связанных с изменением цвета пикселя (и получения информации о его цвете), и самим количеством пикселей. В данном алгоритме, обрабатываются только те пиксели, которые лежат внутри области, И каждый пиксель обрабатывается по одному разу. Следовательно, с точки зрения отрисовки данный алгоритм является одним из наиболее быстродействующих. Тем не менее, для некоторых фигур алгоритм становится значительно более накладным («расческа»).

**Алгоритм заполнения по ребрам**

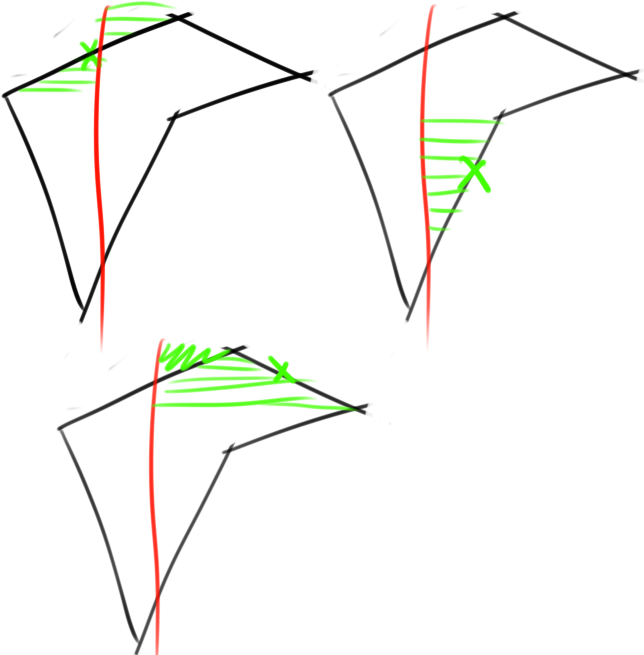
Ребра никак не упорядочиваются, сортировка не происходит. В этом алгоритме ребра могут обрабатываться в произвольном порядке. Алгоритм прост, но требует бОльших временных затрат, поскольку некоторые пиксели могут обрабатываться несколько раз.

Задается цвет фона и цвет закраски; цвет закраски = !(цвет фона), цвет фона = !(цвет закраски) (инверсия/дополнение). Суть алгоритма: для каждой сканирующей строки, пересекающей ребро многоугольника, дополнить (проинвертировать) все пиксели, расположенные правее точки пересечения.



Количество обработок пикселя определяется количеством ребер, расположенных левее этого пикселя.

Алгоритм заполнения с перегородкой: для каждой сканирующей строки пересекающей ребро, дополнить все пиксели, расположенные правее точки пересечения, но левее перегородки (если х < х\_пер). Затем дополнить все пиксели, расположенные левее точки пересечения, но правее перегородки (если х > х\_пер).



**Алгоритм со списком ребер и флагом**

Флаг – признак принадлежности точки внутренней или внешней области многоугольника. =true , если точка расположена внутри. Алгоритм двушаговый:

1) очертить границы закрашиваемого многоугольника

2) закрасить ограниченный многоугольник

02.04.14-----------------------------------------------------------------------

**Алгоритм со списком ребер и флагом**

Флаг – признак принадлежности точки внутренней или внешней области многоугольника. =true , если точка расположена внутри. Алгоритм двушаговый:

1) очертить границы закрашиваемого многоугольника

2) закрасить ограниченный многоугольник

Цикл по всем сканирующим строкам (по у от Ymax до Ymin)

Flag = false

Цикл по всем пикселям сканирующей строки (по х от 0 до Xmax)

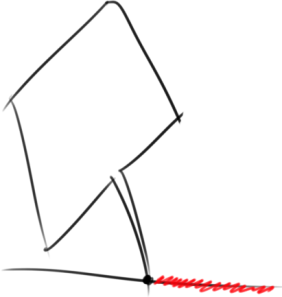
Если цвет(х,у) = цвет границ, то flag=!flag; продолжаем

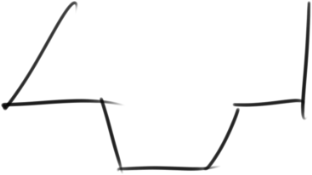
Если флаг=истина, то цвет(х,у) = цвет закраски; иначе цвет(х,у) = цвет фона

Конец цикла по Х

Конец цикла по У

Возникающие проблемы:

На стадии отрисовки границ, два ребра могут сливаться в одно. Как вариант – отрисовывать границу с использованием инверсного цвета пера. Либо же, первое ребро отрисовывается, при отрисовке последующих – если точку надо поставить в ту точку, которая уже подсвечена, то поставить её рядом.



Также извечный вопрос строки, проходящей через вершины.

Также, использовать обычный способ отрисовки отрезков нельзя – может возникнуть разбиение; точек пересечения со сканирующей строкой может оказаться несколько.

Скорость работы алгоритма закраски определяется количеством операций обращений к пикселю. И первый и последний алгоритмы обрабатывают каждый пиксель единожды, НО – нужно и смотреть, сколько обрабатывается пикселей. В некоторых алгоритмах анализируются пиксели, лежащие не только внутри обрабатываемой области.

**Алгоритмы затравочного заполнения**

1.должна быть задана область, подлежащая заполнению (гранично-определённая, или внутренне-определённая)

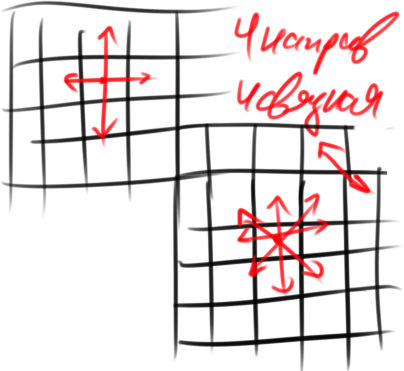
2.должен быть задан затравочный пиксель – точка, заведомо лежащая внутри области

Способы задания области.

1. Гранично-определённая => задана граница области, т.е. известен цвет границы



2. Внутренне-определённая область => все пиксели, принадлежащие области, имеют один и тот же цвет – этот цвет должен быть известен. Закрашиваемые области могут быть четырёхсвязными и восьмисвязными.Связность можно определить количеством направлений движения для достижения точки области.

Граница является 8связной; у 8связной граница является 4связной.

Алгоритмы можно разделить на алгоритмы для гранично-определённых областей, и внутренне-определённых областей. Также для 4связных областей и для 8связных областей, причем 8связный алгоритм справляется и с 4связной.

**Простой алгоритм заполнения с затравкой**

В алгоритмах заполнения очень удобно и эффективно использовать стек для хранения затравочных пикселей.



3(x,y); (x,y+1); (x+1,y); (x,y-1); (x-1; y)

1.Задаются граница, цвет границы, координаты затравочного пикселя

2.занесение затравочного пикселя в стек

3.пока стек не пуст, выполнить следующие действия:

- извлечь пиксель из стека (координаты)

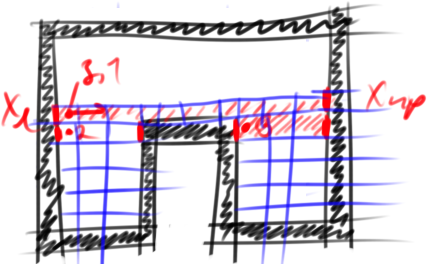
- закрасить извлеченный пиксель (возможно, с проверкой на цвет – нужно ли закрашивать или он уже)

- проанализировать цвет 4-х соседних пикселей ^. Если пиксель не является граничным, то поместить его в стек

Алгоритм достаточно прост в реализации и понимании. Недостатки: перерасход памяти – в стек заносится любой незакрашенный неграничный пиксель, т.е. в стеке побывает каждый пиксель области. В наихудшем случае, пиксель может быть занесён в стек аж четыре раза.

**Построчный алгоритм заполнения с затравкой**

Для улучшения характеристик алгоритма, в стек надо заносить не каждый незакрашенный пиксель, а один на группу незакрашенных. Под группой будем понимать пиксели в одной строке, образующие непрерывный интервал. Это группа примыкающих друг к другу пикселей, ещё не закрашенных, ограниченных уже закрашенными\граничными пикселями. В стек заносится самый левый или самый правый пиксель.

Анализируем пиксель – закрашиваем – анализируем пиксели сверху и снизу. Каждый пиксель анализируется, а затем может быть рассмотрен ещё как и верхняя и нижняя граница – рассматривается три раза.

1. Если стек пуст
2. то конец
3. иначе: извлечение пикселя (х,у) из стека
4. цвет(х,у) = цвет закраски
5. tx = x, запоминаем абсицссу  
   //два однотипных куска, обеспечивают закраску пикселей слева и справа, с запоминанием границы
6. двигаемся влево, x=x-1
7. если цвет(х,у) != цвету границы
8. цвет(х,у) = цвет закраски; гото 6
9. иначе: Xleft = X+1
10. x=tx
11. двигаемся вправо, x=x+1
12. если цвет(х,у) != цвету границы
13. цвет(х,у) = цвет закраски; гото 11
14. иначе: Xright = X-1  
    //два однотипных куска, обеспечивают закраску пикселей слева и справа, с запоминанием границы
15. x = Xleft
16. y = y+1 рассматриваем строку сверху
17. если x <= Xright (иначе гото
18. то Fl=0 //признак нахождения
19. Если цвет(х,у) != цвету закраски (иначе гото 25)
20. То если цвет(х,у) != цвету границы (иначе гото 25)
21. То если х<=хпр (иначе гото 25) (иначе гото 25)
22. То если Fl=0
23. То FL=1
24. Х=Х+1; гото 19

№ по журналу mod 4:

1 -> с упорядоченным списком

2 -> по ребрам

3 -> с перегородкой

0 -> с флагом

Реализовать алгоритм: нарисовать область; закрасить