LOBACHEVSKY STATE UNIVERSITY
of NIZHNI NOVGOROD
National Research University

Computing Mathematics and Cybernetics faculty

Software department

Bill Gates:
«Оценивать эффективность программ по числу строк кода – это все равно, что оценивать прогресс авиации взвешиванием самолета»

CS255. Computer Graphics Introduction Course

Базовые растровые алгоритмы: Растеризация линий. Алгоритмы закрашивания

Турлапов, Вадим Евгеньевич
проф. каф. МО ЭВМ

Растеризация линий.
Прямое вычисление. Цифровой дифференциальный анализатор (ЦДА,DDA)

Растеризация прямой линии (x_1,y_1), (x_2,y_2)

Один из методов разложения отрезка в растр состоит в решении дифференциального уравнения, описывающего этот процесс аналогично устройству ЦДА. Для прямой линии имеем:

dy/dx = m=const или $\Delta y/\Delta x$ = (y2 – y1) / x2 – x1=m , где m – наклон (slope) прямой.

В зависимости от угла наклона выполняется цикл по x (m<=1) или по y (иначе). При этом используются соответственно выражения:

y= f(x)=y1+m(x-x1) и y(i+1)-y(i)=m ·(x(i+1)-x(i)), $\Delta x \neq 0$, x(0)=x1, y(0)=y1; (1) или

x=g(y)=x1+(y-y1)/m и x(i+1)-x(i)=(y(i+1)-y(i))/m , $\Delta y \neq 0$, x(0)=x1 , y(0)=y1. (2)

Фрагмент алгоритма прямого вычисления по ЦДА для случая: |x2-x1|>|y2-y1|, x2>x1, x-целое.

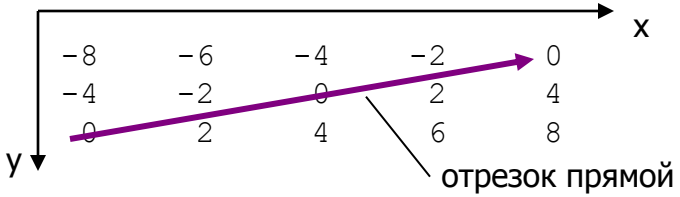
```
float y=y1;
for (int x=x1; x<=x2; x++, y+=m) setPixel(x,round(y)); // y -округляется до ближайшего целого
```

Наиболее интересен цифровой дифференциальный анализатор (digital differential analyzer - DDA) в более широком смысле - на основе неявного уравнения прямой:

$$\Phi(x,y) = \Delta x\cdot(y-y1) + \Delta y\cdot(x-x1) = 0 \quad (3)$$

Подстановка (x,y) в уравнение (3) позволяет анализировать близость точки к ориентированной прямой в величинах кратных Δx, Δy со знаком: «+» - справа; «-» - слева от прямой.

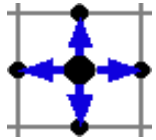
Пример применения анализатора (3) в окрестности отрезка (3,7),(7,5), т.е. $\Delta x=4$, $\Delta y=-2$:



Растеризация линий.
Инкрементные алгоритмы

Инкрементные алгоритмы (incremental algorithms): алгоритмы построенные только на основе целочисленных сложений и вычитаний, т.е. без умножения и деления

Пусть **(x0,y0)** - наша точка , а **(x,y)** - некоторая другая точка. Для определения "близких" пикселей введем **понятия 4- и 8-смежности (8-adjacent)** и, затем, основанные на них **понятия 4- и и 8-связности (8-connected)**: когда существует непрерывный путь из 4- или 8-смежных пикселов из (x0,y0) в (x,y).

1. **4-смежность** $\|x-x_0\|+\|y-y_0\|\le 1$	
2. **8-смежность** $\|x-x_0\|\le 1, \|y-y_0\|\le 1$	

Типичными представителями инкрементных алгоритмов являются **алгоритмы Брезенхэма (Bresenham J.E)**:
- Алгоритм для отрезков, 1965 год (для плоттеров)
- Алгоритм для окружностей, 1977 год

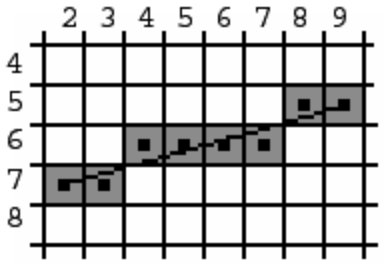
Во всех алгоритмах наличие приращения по x или y рассчитывается на основе анализа функции погрешностей, построенной с различными вариантами на основе цифрового дифференциального анализатора.

Растеризация линий.
Алгоритмы Брезенхэма (Bresenham J.E.)

Восьмисвязный алгоритм Брезенхэма для отрезка (x1,y1),(x2,y2):

```
//Bresenham's line-drawing algorithm (число шагов=d+1)
void line_bresenham8(int x1,int y1,int x2,int y2){
int x_err=0, y_err=0;           // функции погрешностей по x и y
int dx=x2-x1; int dy=y2-y1; // приращения по x и y на всем отрезке
int incX=0; int incY=0;                  // шаги приращения по x и y =0
if (dx>0) {incX=1; else if (dx!=0) incX=-1;}  // шаг приращения по x
if (dy>0) {incY=1; else if (dy!=0) incY=-1;}  // шаг приращения по y
dx=abs(dx); dy=abs(dy);
dx>dy? d=dx: d=dy;  // выбор max(dx,dy) в качестве тестовой величины
int x=x1, y=y1; setPixel(x,y); // установка пикселя в начале отрезка
for (int i=1; i<=d; i++)
 {x_err+=dx; y_err+=dy;  // приращение ошибки при перемещении на пиксель
 if (x_err>=d) {x+=incX; x_err-=d;} // условие шага по x
 if (y_err>=d) {y+=incY; y_err-=d;} // условие шага по y (+0.5d,-0.5d)!!!
 setPixel(x,y);}} // err = Δx·(y-y1) + Δy·(x-x1) =0
```

Пример для отрезка (2,7),(9,5):м

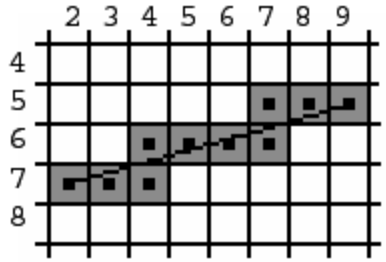


Результаты ЦДА для него:

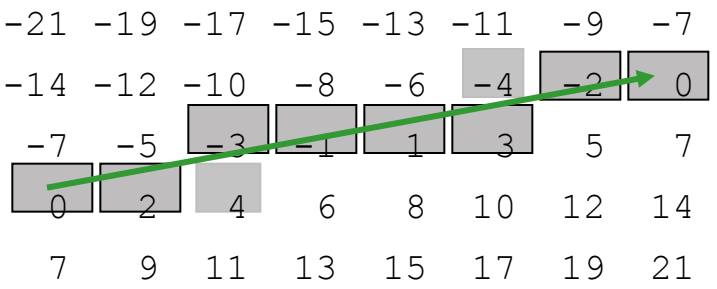
-21	-19	-17	-15	-13	-11	-9	-7
-14	-12	-10	-8	-6	-4	-2	0
-7	-5	-3	-1	1	3	5	7
0	2	4	6	8	10	12	14
7	9	11	13	15	17	19	21

Растеризация линий.
Алгоритмы Брезенхэма (Bresenham J.E.)

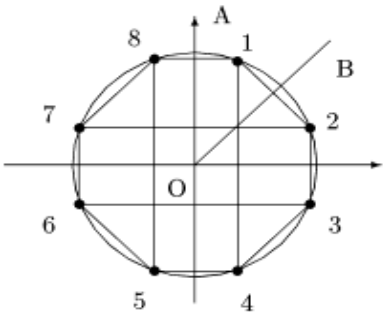
Результат работы 4-связного алгоритма (число шагов=dx+dy+1=10):



На том же множестве значений ЦДА:



Растеризация линий.
Окружность. Алгоритм Брезенхэма (1977)



Функция $\mathbf{F(x,y)=x^2+y^2-R^2}$ определяет лежит ли точка: на окружности (**F=0**), внутри окружности (**F<0**) или вне ее (**F>0**). Рассмотрим растеризацию окружности во втором октанте правостронней системы координат. Для получения полной окружности будем строить еще 7 пикселов симметричных рассчитанному (метод circle8Pixel).

Пусть x_i,y_i принадлежит окружности: $F(x_i,y_i)=0$. Применим в алгоритме метод средней точки (midpoint algorithm, модификация Мичнера), заключающийся в оценке ошибки для наиболее вероятного прогноза движения вдоль среднего (по интервалу) значения касательной (=-1/2).

$Err_i=F(x_i+1,y_i-1/2)=2x_i - y_i + 5/4$ (1)

Если $Err_i<0$, то $y_{i+1} = y_i$ и

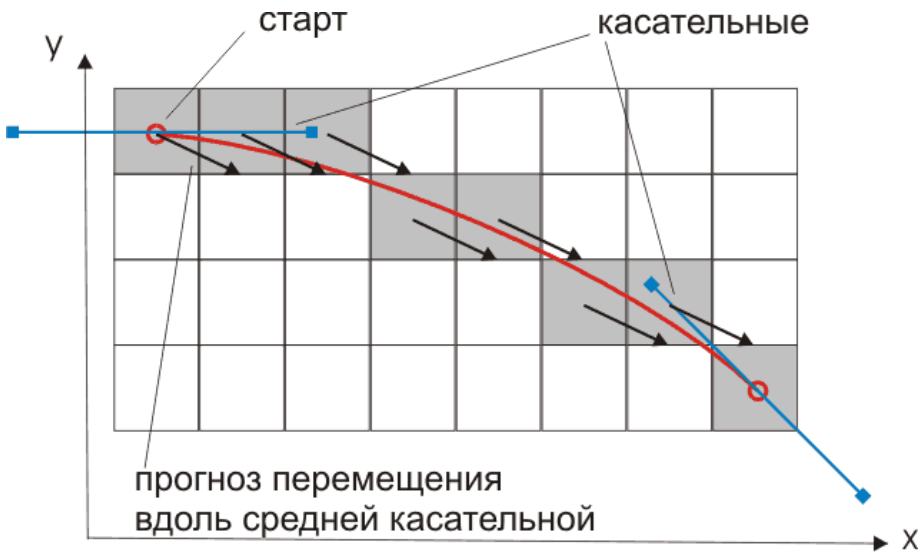
$Err_{i+1}=F(x_i+2,y_i-1/2)=4x_i - y_i + 17/4$ (2)

$dErr_i= Err_{i+1}- Err_i= 2x_i + 3$ (3)

Если же $Err_i>0$, то $y_{i+1} = y_i - 1$ и

$Err_{i+1}=F(x_i+2,y_i-3/2)=4x_i - 3y_i + 25/4$ (4)

$dErr_i= Err_{i+1}- Err_i= 2(x_i - y_i)+ 5$ (5)



Пример программной реализации метода с приращениями ошибки по формулам (3,5).

Растеризация линий.
Окружность. Алгоритм Брезенхэма (1977)

Первоначальный алгоритм Брезенхэма

Для возможных точек вычислим и сравним абсолютные значения разностей квадратов расстояний от центра окружности до точки и окружности:

- $|Dg| = |(X+1)^2 + Y^2 - R^2|$
- $|Dd| = |(X+1)^2 + (Y-1)^2 - R^2|$
- $|Dv| = |X^2 + (Y-1)^2 - R^2|$

Выбирается и заносится та точка, для которой это значение минимально.

Выбор способа расчета определяется по значению Dd. Если $Dd < 0$, то диагональная точка внутри окружности – приращение по x. Если $Dd > 0$, то диагональная точка вне окружности – в зависимости от величин приращения ошибки, приращение либо по y, либо по x и y. И, наконец, если $Dd = 0$, то диагональная точка лежит точно на окружности – приращение по x и y одновременно.

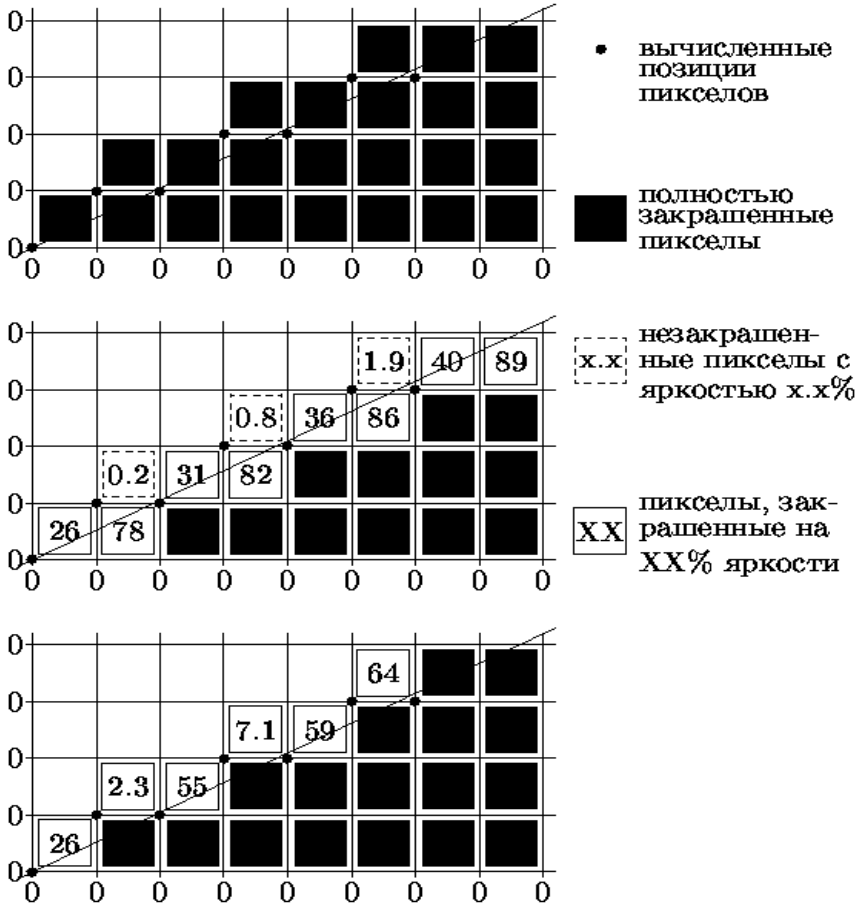
Пример алгоритма Брезенхэма с сайта algolist.manual.ru

Растеризация линий.

Модификация алгоритма Брезенхэма со сглаживанием границы

Модификация растеризации линии по Брезенхэму с целью сглаживания границы:

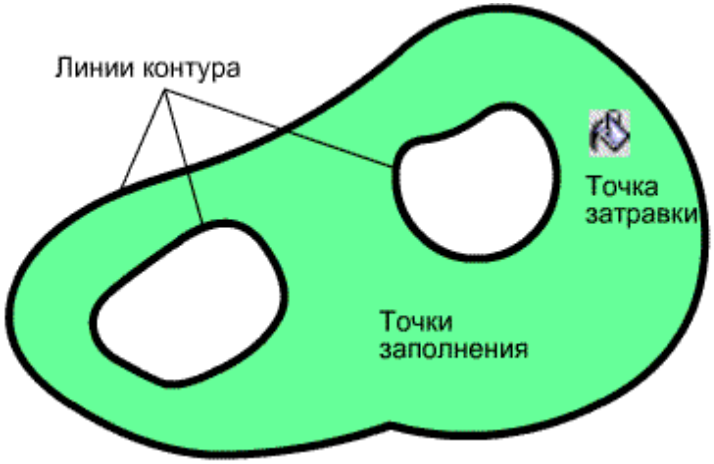
- Полная закраска
- Неполная закраска 8-связный вариант
- Неполная закраска, 4-связный вариант



Для оценки процента закраски (p) может быть использована величина функции ошибки:
p=100·(d-err)/d

Алгоритмы закрашивания.
Алгоритмы вывода фигур

- **Фигура (region)** -плоский геометрический объект, состоящий из **линий контура** и **точек заполнения**, содержащихся внутри контура.
- В графическом выводе различают, в связи с этим: 1)вывод контура; **2)вывод точек заполнения**.
- В выводе точек заполнения
- различают:



1) закрашивание (от внутренней точки - затравки) до границы заданного цвета, либо закрашивание пикселов, принадлежащих заданному цветовому диапазону; ~pixel-defined regions

2) закрашивание контура, заданного математически ~ symbolic-defined regions.

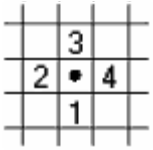
Затравочный пиксел для фигур заданных цветовой границей или цветовым диапазоном, как правило, вносится пользователем.

Внутренний пиксел для фигур заданных математически, при необходимости, может быть определен либо на основе решения задачи пересечения фигуры с тестовой горизонталью (см. далее), либо вычислением суммарного угла при обходе всех вершин фигуры лучем из тестируемой точки (=360 градусов).

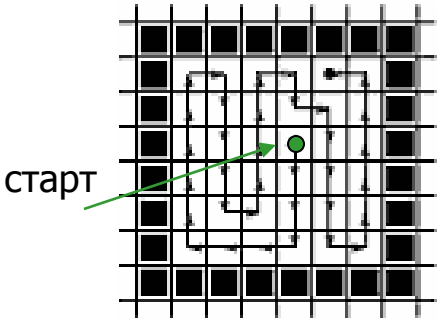
Алгоритмы закрашивания до цвета границы.
Простейший рекурсивный алгоритм

Для 8-связной границы нельзя применять 8-связный алгоритм закрашивания, но можно 4-связный.

```
void FillPixel(int x, int y, int borderC, int fillC)
{ // 4-связный вариант
int c=getPixel(x,y); // цвет пиксела
if ((c!=borderC)&&(c!=fillC)) // не граничный и не закрашен?
                {setPixel(x,y,fillC); // закрасить и «осмотреться»:
                FillPixel(x+1,y,borderC,fillC);   //Рекурсия
                FillPixel(x,y+1,borderC,fillC);   //
                FillPixel(x-1,y,borderC,fillC);   //
                FillPixel(x,y-1,borderC,fillC); }  //
}
```



Порядок перебора соседних пикселов

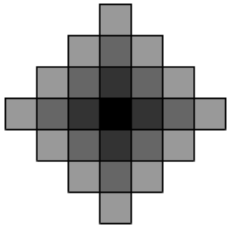


Пример обхода области

Алгоритм работает пока стек рекурсии не пуст. Имеет низкую эффективность, т.к.: 1)требует большой глубины рекурсии ~ 4n pix (при стеке в 64K рекурсивным алгоритмом можно закрасить квадрат 57×57 пикселов);

2)каждый уже отрисованный пиксел тестируется еще 3 раза. ➔ Модификация - итеративный алгоритм: в стек заносятся пиксели с неполностью закрашенной окрестностью (вместо рекурсии - проверка окрестности).

Более рационален **волновой** алгоритм (см. рис). Для хранения координат пикселов текущего ромбовидного фронта волны использует динамичекие массивы до 10000 элементов.

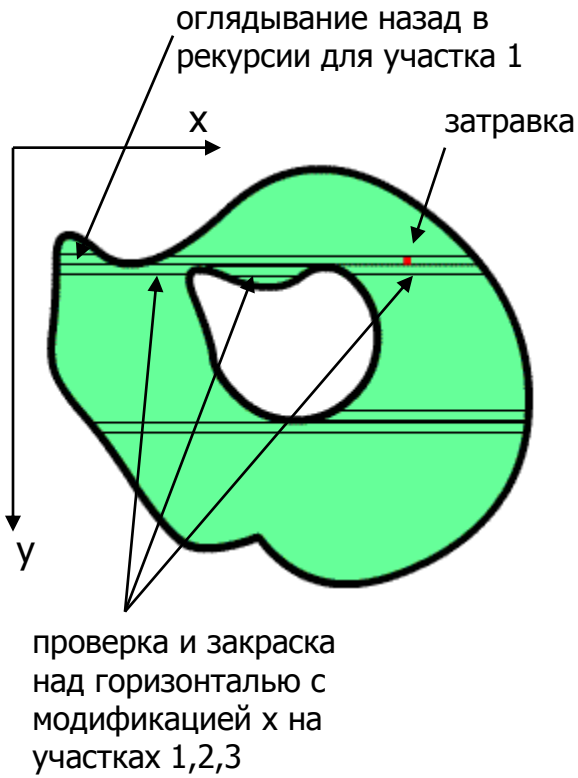


Волновой алгоритм

Алгоритмы закрашивания до цвета границы.
Закрашивание линиями

Еще более быстрый и получивший распространение в КГ алгоритм. На каждом шаге закрашивается горизональная линия между пикселами контура и обнаруживаются незакрашенные пикселы над и под линией. Рекурсивный, но глубина рекурсии пропорциональна лишь числу пикселов в линии. В случае создания стека достаточно хранить 1 пиксел на каждый закрашиваемый участок.

```
int lineFill(int x, int y,int dy,int preXL, int preXR)
{int xl=x, xr=x; int c;
//левая и правая граница текущей горизонтали
do {xl--; c=getPixel(xl,y):} while (c!=BORDER);
do {xr++; c=getPixel(xr,y):} while (c!=BORDER);
xl++; xr--;
Line(xl,y,xr,y,fillC); //закрашиваем горизонталь
//Рекурсии: 1)проверка и закраска над горизонталью
for (x=xl; x<=xr; x++) {c=getPixel(x,y+dy);
    if (c!=BORDER) x=lineFill(x,y+dy,dy,xl,xr);}
// 2)оглянулись и закрасили левее предыдущей левой
// и правее предыдущей правой границ
for (x=xl; x<=preXL; x++) {c=getPixel(x,y-dy);
    if (c!=BORDER) x=lineFill(x,y-dy,-dy,xl,xr);}
for (x=preXR; x<=xr; x++) {c=getPixel(x,y-dy);
    if (c!=BORDER) x=lineFill(x,y-dy,-dy,xl,xr);}
return xr; }
Обращение в затравке: lineFill(xSt,ySt,1,xSt,xSt);
```



Построчное заполнение контура полигона, заданного списком вершин (polygon-defined region)

Алгоритм "XY"

Свойство топологии контура: ***любая прямая пересекает контур четное число раз.***

1.Найти min{y_n} и max{y_n} по всем N вершинам P_n

2.Выполнить цикл по y от ymin до ymax:

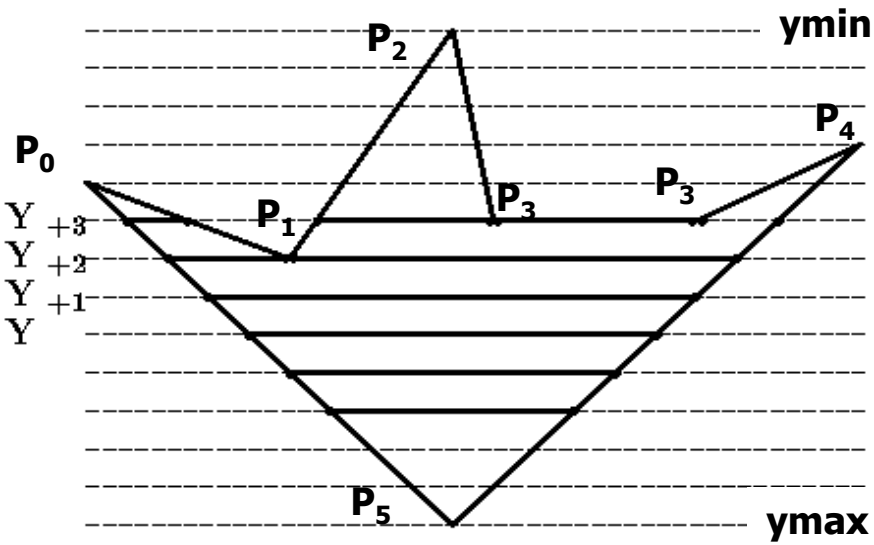
{ 2.1. Расчет множества {x_k} x-координат пересечения всех отрезков границы с горизонталью y

$x_k = x_n + [(x_{n+1} - x_n)\cdot(y - y_n)]/(y_{n+1} - y_n)$

2.2. Сортировка {x_k} по возрастанию

2.3. Вывод отрезков {(x_0,y)-(x_1,y), (x_2,y)-(x_3,y), …, (x_{2m},y)-(x_{2m+1},y)} цветом заливки

}



Число тактов работы алгоритма: $N_{тактов} = (y_{max} - y_{min}) \cdot N_{горизонтали} = (y_{max} - y_{min}) \cdot (N+1)$.

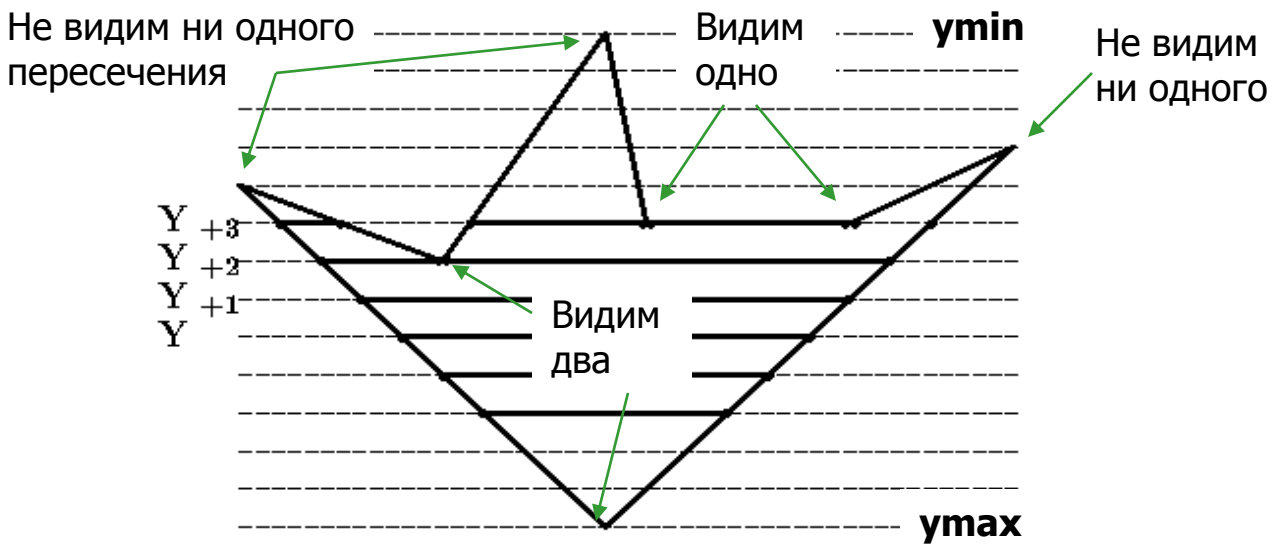
Свойство топологии контура можно также использовать **для поиска полигона по указанной внутренней точке**.

Правило подсчета числа пересечений ребер с горизонталью

Данный простой алгоритм работоспособен, но генерирует двух и трехкратное занесение части пикселов. Это мало приемлемо для устройств вывода типа матричных или струйных принтеров. Возможны также ошибки из-за попадания пересечения точно в место соединения двух ребер. Проблему решает следующее правило:

при подсчете пересечений игнорировать пересечения строки развертки с верхней конечной точкой ребра, а также полностью игнорировать горизонтальные ребра (см. рис. и Ф.Хилл,с.661).

Правило при желании можно усложнить.



Построчное заполнение полигона, заданного списком вершин. Возможности оптимизации

Алгоритм можно ускорить уменьшением числа ребер, перебираемых для каждой горизонтали. Возможность уменьшения основана на следующих обстоятельствах и принципах:

1.Принцип когерентности.

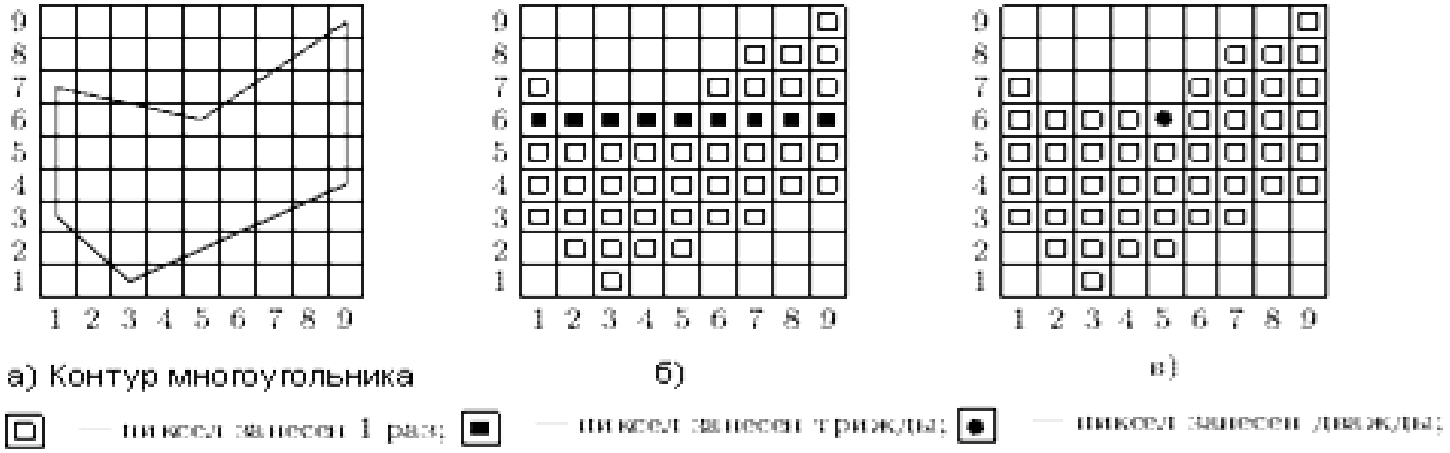
Пикселы соседних строк скорее всего одинаковы по числу и цвету и меняются только там, где строка пересекается с ребром многоугольника (когерентность строк).

Если ребро пересекалось i-й строкой, то оно скорее всего будет пересекаться и i+1 строкой (когерентность ребер) – возможно создание мало изменяющихся списков активных ребер пересекаемых горизонталями.

2.Смена количества интервалов заливки происходит только тогда, когда в строке сканирования появляется вершина.

3.При переходе к новой строке скорее всего $X_{i+1} = X_i + 1/k$ (k - тангенс угла наклона ребра dy/dx, так как dy = 1, то 1/k = dx).

Пример меньшения дублирования правилом подсчета пересечений: случай (в)



Растеризация линий.
Стиль линии. Перо

Для описания различных типов изображений, построенных на основе линии, используются стиль линии или перо. Перо, принимая различную форму (горизонтальный, вертикальный, наклонный отрезки, квадрат, эллипс) способно при скольжении под разными углами изменять форму следа.

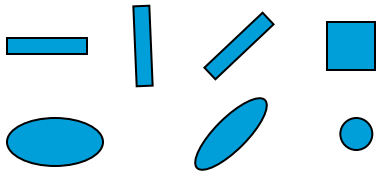
Алгоритм вывода толстой линии

Основа – вывод тонкой линии:

….

Вывод пиксела (x,y) ➔ Вывод фигуры пера с центром в (x,y): Pen(x,y)

….

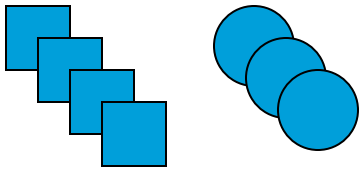


Преимущество:

•Можно использовать эффективные алгоритмы вывода линий

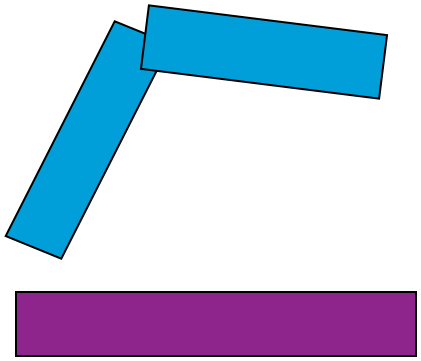
Недостаток:

•Неэффективность (многократное перекрытие пикселов) для ряда форм (квадрат, эллипс) и положений пера



Растеризация линий.
Стиль линии. Перо

Перья в виде отрезков линий
Более эффективны.
Проблемы: разрывы в углах полилинии и острые концы
Варианты решения проблемы:
1)Вывод линии как полигона с заполнением.
Недостаток: меньшая скорость.
2)Воспроизведение толстой линии последовательностью эквидистантных полилиний



Алгоритмы вывода пунктирной линии

Тонкая пунктирная линия
Сводится к делению по модулю S (период пунктира) значения счетчика Count:
Если (Count mod S) < S/2, то вывод setPixel(x,y); Count++;

На стыке фрагментов ломаной (кривой) необходимо предотвратить обнуление счетчика.
В алгоритмах вывода, использующих симметрию, применение пунктира затруднено.

Для **толстой** линии простейший алгоритм получается заменой Pixel на Pen.

Справочный материал. Лабораторная работа Demo

Растеризация.
Стиль. Кисть

Кисть (Brush) используется для плоских площадных объектов. Общий вид алгоритма задания стиля:
C=f(x,y) // где функция f(x,y) определяет цвет в зависимости от координат и стиля
setPixel(x,y,C);
Иногда аргументы не нужны - случайное заполнение:
C=random();

Штриховка (Hatch)
f(x,y,hatch)={Сш, если ((x+y)modS)<T (T-толщина штриховки, S – период); Сфона – иначе} – для штриховки под 45 градусов

Растровый образец (Pattern). Копирование образца кисти (m x n):
xt=(x mod m); yt=(y mod n);
C=f(xt,yt,pattern);
setPixel(x,y,C);

Если размеры шаблона кисти равны степени двойки: м. использовать поразрядный сдвиг в mod.
Справочный материал Лабораторная работа Demo

Темы самостоятельной работы

1. Реализация алгоритма Брезенхэма для прямой линии с антиалиасингом и без в форме приложения для тестирования на полигонах
2. Реализация алгоритма Брезенхэма для окружности с антиалиасингом и без в форме приложения для тестирования
3. Реализация алгоритма Брезенхэма для эллипса с антиалиасингом и без в форме приложения для тестирования
4. Реализация алгоритма Брезенхэма для произвольной дуги с антиалиасингом и без в форме приложения для тестирования
5. Реализация алгоритма закрашивания линиями для фигуры с цветовой границей в форме приложения для тестирования
6. Реализация алгоритма закрашивания линиями для математического описания контура (в форме приложения для тестирования)
7. Реализация редактора для рисования пером и кистью (в форме приложения для тестирования)

Источники

1. Порев В.Н. Компьютерная графика. –СПб.: БХВ-Петербург, 2002. –432с.
2. Хилл Ф. OpenGL. Программирование компьютерной графики. – С.Пб: Питер, 2002. 1088с.
3. Роджерс Д., Адамс Дж. Математические основы машинной графики: Пер. с англ. - М.: Мир, 2001. - 604с.
4. Шикин Е.В., Боресков А.В. Компьютерная графика. Полигональные модели. –М.: ДИАЛОГ-МИФИ, 2001.-464с.
5. Майкл Ласло. Вычислительная геометрия и компьютерная графика на С++: Пер. с англ. –М.: «БИНОМ», 1997. –304с.
6. The home page of the Graphics Research Group at Harvard University. Computer Graphics courses http://www.eecs.harvard.edu/graphics/#classes
7. Stanford Computer Graphics Laboratory. Courses in Graphics http://graphics.stanford.edu/courses/
8. Фень Юань. Программирование графики для Windows. -М.: 2004.
9. Юджин Олафсен, Кенн Скрайбнер, К.Дэвид Уайт. MFC и Visual C++ 6.0. Энциклопедия программиста. – М.: .2003.