22. Понятие о векторной и растровой графике. Основные алгоритмы двумерной растровой графики. Построение отрезков, кругов и эллипсов. Заполнение прямоугольников, многоугольников и эллиптических секторов. Отсечение в растровой графике. Отсечение линий, кругов, эллипсов и многоугольников.

Понятие о векторной и растровой графике. Основные алгоритмы двумерной растровой графики. Построение отрезков, кругов и эллипсов. Заполнение прямоугольников, многоугольников и эллиптических секторов. Отсечение в растровой графике. Отсечение линий, кругов, эллипсов и многоугольников.

1) Понятие о векторной и растровой графике.

Растровые изображения состоят из множества точек. Все вместе они образуют картинку: Как вы можете себе представить, для хранения информации обо всех этих точках требуется файл немалого размера. Другая проблема при использовании растровой графики заключается в том, что ее нельзя пропорционально увеличить. Если вы попытаетесь увеличить растровую картинку, она приобретет зернистый вид, потому что вы увидите все эти точки.

Векторные изображения задаются с помощью уравнений, которые определяют линии, формы и их расположение. Чистые фрагменты не нуждаются в задании, а уравнения довольно эффективно используются для сохранения информации. В результате, размер файлов оказывается значительно меньше. Масштаб векторных изображений можно увеличивать или уменьшать, практически не изменяя качества изображения. При этом не имеет значения, какой размер имеет ваш рисунок, он выглядит великолепно. На самом деле, векторное изображение может при увеличении выглядеть даже лучше, потому что кривые становятся более плавными.

2) Основные алгоритмы двумерной растровой графики. Построение отрезков, кругов и эллипсов.

Вывод линии. Алгоритм Брезенхема

При ограниченном разрешении точка приближаемой линии может находиться либо на самой линии, либо слева от нее либо справа. Расстояние на котором фактически установленная точка находиться от рисуемой линии, -- ошибка в рисовании линии. Переходя к каждой следующей точке, ошибка скажет какая точка будет хуже приближать кривую а какая лучше.

Принцип Брезенхема состоит в том, чтобы с каждой итерацией двигаться на одну точку по той оси проекция на которую больше. По другой оси смешение на один пиксель происходит лишь тогда, когда линия отклонилась от текущей оси более чем на полпикселя (Рисунок 1).

```
inline void Image::Octant0(int X0,int Y0,int DeltaX,int DeltaY,int XDirection) {

  /* DeltaX -- поэкция на ось X

     DeltaY -- проэкция на ось Y

     XDirection -- направление движения
```

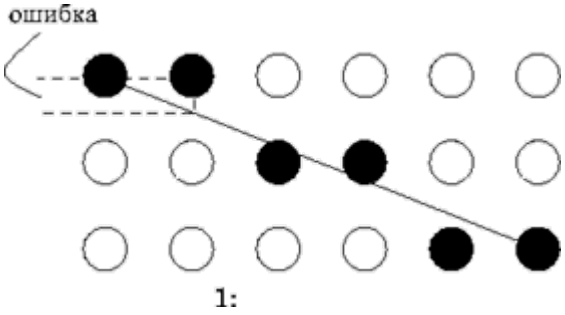
```
для данной функции |DeltaX|>DeltaY
следовательно основная ось X, а
вспомогательная Y
*/
int DYx2;
int DYx2MinusDXx2;
int ErrorTerm;
/* Установим начальную ошибку накопления и значения,
   используемые во внутреннем цикле */

DYx2=DY*2;
DYx2MinusDXx2= DYx2 - DX*2;
ErrorTerm = DYx2 - DX;// ошибка

/* ставим первую точку */
setZPixel(X0,Y0,getZPixel(DeltaX),Color);

while(DeltaX--) {
  if(ErrorTerm >=0) {
/* шагаем по вспомогательной оси
   и подправляем ошибку накопления */
    Y0++;
    ErrorTerm+=DeltaYx2MinusDeltaXx2;
  } else {
/* добавляем ошибку накопления */
    ErrorTerm+= DeltaYx2;
  }
/* двигаемся по основной оси */
  X0+= XDirection;
```

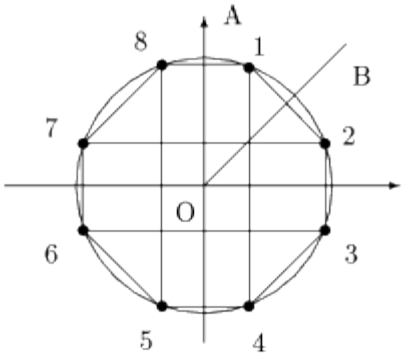
```
  /* ставим точку */
    setZPixel(X0,Y0,getZPixel(DeltaX),Color);
  }
}
```



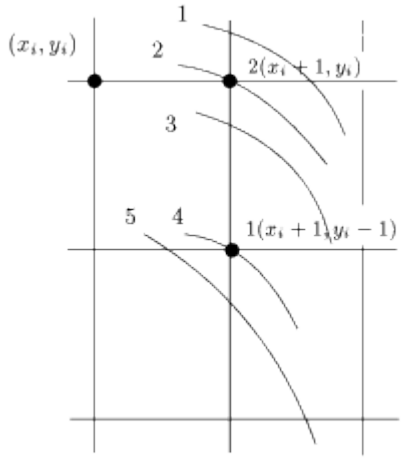
1:

Окружность. Алгоритм Брезенхема

Будем рассматривать 1/8 окружности(для уменьшения числа операций и сравнений). Каждая точка этого фрагмента должна быть еще семь раз отображена с помощью преобразований симметрии для получения полной окружности.



Приступим к разбору ключевой идеи алгоритма. Пусть мы находимся в некоторой промежуточной фазе построения. Мы только что поставили точку (x_i, y_i) и теперь должны сделать выбор между точками 1(x_i+1, y_i-1) и 2(x_i+1, y) (Напомним, что мы строим часть окружности, заключенную в $\angle AOB$, следовательно, подняться выше мы не можем и спуститься вниз более чем на одну точку не можем тоже.)



Реальная окружность может быть расположена относительно точек 1 и 2 одним из пяти способов 1-5. Если мы выбераем точку 1, то тем самым говорим, что $(x_i+1)^2+(y_i-1)^2 \approx R^2$. Если же выбераем точку 2, то допускаем, что $(x_i+1)^2+(y_i)^2 \approx R^2$. Рассмотрим две погрешности Δ^i_1 и Δ^i_2:

$$\Delta^i_1 = (x_i+1)^2+(y_i-1)^2-R^2$$

$$\Delta^i_2 = (x_1+1)^2+(y_i)^2-R^2$$

и контрольную величину$\Delta^i = \Delta^i_1+\Delta^i_2$. При выборе точки, следующей за (x_i, y_i), станем руководствоваться следующим критерием:

если $\Delta^i > 0$, выберем точку 1;

если $\Delta^i \le 0$, выберем точку 2.

Обоснуем разумность такого выбора. Рассмотрим знаки погрешностей Δ^i_1 и Δ^i_2 и их влияние на знак контрольной величины Δ^i для всех пяти возможных положений окружности.

Для положения 1.

$\Delta^i_1 < 0$, $\Delta^i_2 < 0 \Rightarrow \Delta^i_1+\Delta^i_2 < 0 \Rightarrow$ выбирается 2.

Для положения 2.

$\Delta^i_1 < 0$, $\Delta^i_2 = 0 \Rightarrow \Delta^i_1+\Delta^i_2 < 0 \Rightarrow$ выбирается 2.

Для положения 3 возможны варианты (учитывая, что $\Delta^i_1 < 0$, $\Delta^i_2 > 0$).

Вариант 3.1. $|\Delta^i_1| \ge |\Delta^i_2| \Rightarrow \Delta^i_1+\Delta^i_2 < 0 \Rightarrow$ выбирается 2.

Вариант 3.2. $|\Delta^i_1| < |\Delta^i_2| \Rightarrow \Delta^i_1+\Delta^i_2 > 0 \Rightarrow$ выбирается 1.

Для положения 4.

$\Delta^i_1 = 0$, $\Delta^i_2 > 0 \Rightarrow \Delta^i_1+\Delta^i_2 > 0 \Rightarrow$ выбирается 1.

Для положения 5.

$\Delta^i_1 > 0$, $\Delta^i_2 > 0 \Rightarrow \Delta^i_1+\Delta^i_2 > 0 \Rightarrow$ выбирается 1.

Далее получим выражение для контрольной величины Δ^i

$\Delta^i = \Delta^i_1+\Delta^i_2 = (x_i+1)^2+(y_i-1)^2-R^2+(x_i+1)^2+(y_i)^2-R^2 = 2x_i^2+2y_i^2+4x_i-2y_i+3-2R^2$.

Выражение для Δ^{i+1} существенным образом зависит от выбора следующей точки. Необходимо рассмотреть два случая: $y_{i+1} = y_i$ и $y_{i+1} = y_i-1$.

$$\Delta^{i+1} \text{ [при } y_{i+1} = y_i] = 2x^2_{i+1}+2y^2_{i+1}+4x_{i+1}-2y_{i+1}+3-2R^2 = 2(x_i+1)^2+2y_i^2+4(x_i+1)-2y_i+3-2R^2 = \Delta^i+4x_i+6.$$

$$\Delta^{i+1} \text{ [при } y_{i+1} = y_i\text{-1]} = 2x^2_{i+1}+2y^2_{i+1}+4x_{i+1}-2y_{i+1}+3-2R^2 = 2(x_i+1)^2+2(y_i-1)^2+4(x_i+1)-2(y_i-1)+3-2R^2 = \Delta^i+4(x_i-y_i)+10.$$

Теперь, когда получено рекуррентное выражение для Δ^{i+1} через Δ^i, остается получить Δ^1 (контрольную величину в начальной точке.) Она не может быть получена рекуррентно, ибо не определено предшествующее значение, зато легко может быть найдена непосредственно

$$x_1 = 0,\ y_1 = R \Rightarrow \Delta^1{}_1 = (0+1)^2+(R-1)^2-R^2 = 2-2R,$$

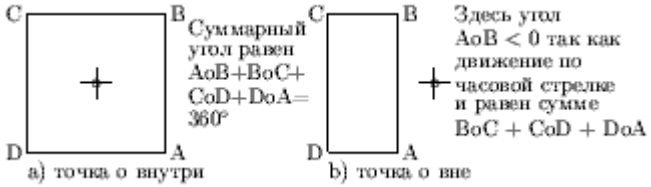
$$\Delta^1{}_2 = (0+1)^2+R^2-R^2 = 1$$

$$\Delta^1 = \Delta^1{}_1+\Delta^1{}_2 = 3-2R.$$

Таким образом, алгоритм построения окружности, реализованный в bres_circle, основан на последовательном выборе точек; в зависимости от знака контрольной величины Δ^i выбирается следующая точка и нужным образом изменяется сама контрольная величина. Процесс начинается в точке (0, *r*), а первая точка, которую ставит процедура sim, имеет координаты (*xc*, *yc*+*r*). При *x* = *y* процесс заканчивается.

Для эллипса надо рассматривать ¼ фигуры.

3) Заполнение прямоугольников, многоугольников и эллиптических секторов.



Построчное заполнение

Реально используются алгоритмы построчного заполнения, основанные на том, что соседние пикселы в строке скорее всего одинаковы и меняются только там где строка пересекается с ребром многоугольника. Это называется когерентностью растровых строк (строки сканирования Y_i, Y_{i+1}, Y_{i+2} на рис.). При этом достаточно определить X-координаты пересечений строк сканирования с ребрами. Пары отсортированных точек пересечения задают интервалы заливки.

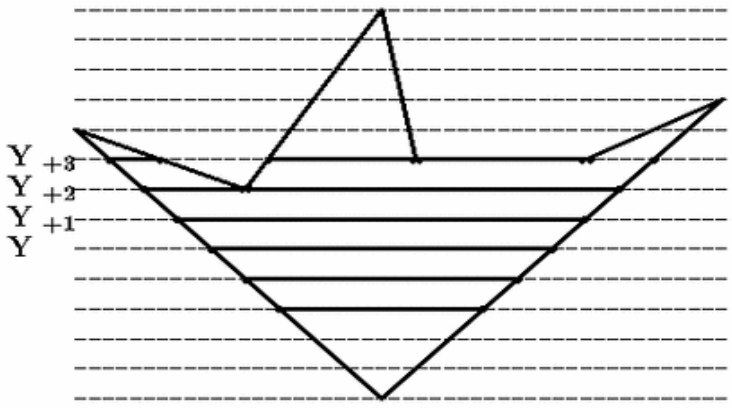


Рис. 0.4.2: Построчная закраска многоугольника

кроме того, если какие-либо ребра пересекались i-й строкой, то они скорее всего будут пересекаться также и строкой i+1. (строки сканирования Y_i и Y_{i+1} на рис. 0.2). Это называется
когерентностью ребер. При переходе к новой строке легко вычислить новую X-координату точки пересечения ребра, используя X-координату старой точки пересечения и тангенс угла наклона ребра:

$$X_{i+1} = X_i + \frac{1}{k}$$

(тангенс угла наклона ребра - k = dy/dx, так как dy = 1, то 1/k = dx).

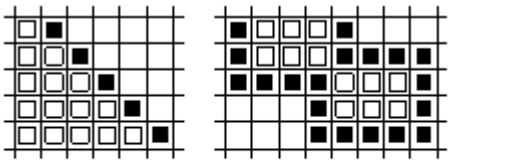
ЗАЛИВКА ОБЛАСТИ С ЗАТРАВКОЙ

По способу задания области делятся на два типа:

- гранично-определенные, задаваемые своей (замкнутой) границей такой, что коды пикселов границы отличны от кодов внутренней, перекрашиваемой части области. На коды пикселы внутренней части области налагаются два условия - они должны быть отличны от кода пикселов границы и кода пиксела перекраски. Если внутри гранично-определенной области имеется еще одна граница, нарисованная пикселами с тем же кодом, что и внешняя граница, то соответствующая часть области не должна перекрашиваться;

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

в этом состоит основное отличие заливки области с затравкой от заполнения



а) 4-х б) 8-ми
связные, внутренне-определенные области
— пикселы области; — пикселы границы.

Построчный алгоритм заливки с затравкой использует пространственную когерентность:

- пикселы в строке меняются только на границах;
- при перемещении к следующей строке размер заливаемой строки скорее всего или неизменен или меняется на 1 пиксел.

Таким образом, на каждый закрашиваемый фрагмент строки в стеке хранятся координаты только одного начального пиксела [], что приводит к существенному уменьшению размера стека. Последовательность работы алгоритма для гранично определенной области следующая:

1. Координата затравки помещается в стек, затем до исчерпания стека выполняются пункты 2-4.
2. Координата очередной затравки извлекается из стека и выполняется максимально возможное закрашивание вправо и влево по строке с затравкой, т.е. пока не попадется граничный пиксел. Пусть это Хлев и Хправ, соответственно.

3. Анализируется строка ниже закрашиваемой в пределах от Хлев до Хправ и в ней находятся крайние правые пикселы всех незакрашенных фрагментов. Их координаты заносятся в стек.
4. То же самое проделывается для строки выше закрашиваемой.

4) Отсечение в растровой графике.

5) Отсечение линий, кругов, эллипсов и многоугольников.

Точка относительно отрезка (прямой).

Пусть прямая задана двумя точками $P_1(x_1,y_1)$ и $P_2(x_2,y_2)$, и требуется установить, как расположена точка $O(x_3,y_3)$ относительно прямой P_1P_2.

Способ 1.

Направление от P_1 к P_2 примем за положительное. Тогда уравнение прямой (ориентированной) записывается в виде:

$F(x,y)=(y_2-y_1)*(x-x_1)-(x_2-x_1)*(y-y_1)=0$ (*)

- Если $F(x_3,y_3)=0$, то точка $O(x_3,y_3)$ лежит на прямой;
- Если $F(x_3,y_3)>0$, то точка $O(x_3,y_3)$ расположена справа от прямой;
- Если $F(x_3,y_3)<0$, то точка $O(x_3,y_3)$ расположена слева от прямой;

Способ 2.

Заметим, что нормаль к прямой, записанной в виде (*), выглядит следующим образом:

$=(y_2-y_1,-(x_2-x_1))$. В этом случае, для того чтобы узнать, как расположена точка O относительно прямой, надо вычислить $F(x_3,y_3)=(\ ,)$ и сравнить значение с 0:

- Если $F(x_3,y_3)=0$, то точка $O(x_3,y_3)$ лежит на прямой;
- Если $F(x_3,y_3)>0$, то точка $O(x_3,y_3)$ лежит по туже сторону, что и нормаль;
- Если $F(x_3,y_3)<0$, то точка $O(x_3,y_3)$ лежит с противоположенной от нормали стороны;

Способ 3.

Известно, что три точки задают плоскость. Соединим точку O с точками P_1 и P_2.

Выберем ось Z, так чтобы она смотрела на нас и проходила через точку O.

Тогда ==(0,0,)

- Если A=0точка O лежит на прямой P_1P_2;

- Если A>0точка O лежит по правую сторону от ;

- Если A<0точка O лежит по левую сторону от ;

Пересечение двух отрезков.

Пусть даны два отрезка P_1P_2 и Q_1Q_2 и надо найти точку их пересечения.

Способ 1. Деление средней точкой (Sproul & Sutherland '68)
Алгоритм нахождения пересечения P_1P_2 с Q_1Q_2 делением средней точкой:
Если точки P_1 и P_2 (Q_1 и Q_2) лежат по одну сторону от Q_1Q_2 (P_1P_2)
То
отрезки не пересекаются
Иначе
найти пересечение P_1P_2 с Q_1Q_2.
Найти пересечение P_1P_2 с Q_1Q_2:
{

если , то return ;

Если P лежит на Q_1Q_2, то return P;
Если P и P_1 лежат по одну сторону от Q_1Q_2
То
Найти пересечение PP_2 с Q_1Q_2
Иначе
Найти пересечение PP_1 с Q_1Q_2
}

Смысл алгоритма состоит в том, что мы каждый раз, делим один отрезок пополам до тех пор, пока не приблизимся к другому отрезку на достаточно малое расстояние.

Способ 2. Аналитический способ
Возьмем параметрическое задание прямых P_1P_2 и Q_1Q_2
$F_1(t)=P_1+(P_2-P_1)*t$
$F_2(s)=Q_1+(Q_2-Q_1)*s$
и найдем их пересечение. Имеем два уравнения и две неизвестные (s и t), значит система разрешима.

- Если решений нет или прямые параллельны.

- Если и пересечение F1(t).

- Если или пересечения нет.

<u>Способ 3.</u> Cyrus & Beck ’78
Рассмотрим параметрическое задание прямой P_1P_2
$P(t)=P_1+(P_2-P_1)*t$.

Пусть I – точка пересечения P_1P_2 с Q_1Q_2, a - вектор нормали к Q_1Q_2, проведенный из точки Q_1.

Тогда для I выполняется равенство:

,т.е. .
Т.о. имеем одно уравнение с одной неизвестной t.

- Если решений нет или отрезки параллельны.

- Если пересечения нет.

- Если I=P(t).

III Отсечение многоугольника выпуклым отсекателем.
(Sutherland & Hodgman ‘74)

P1…P5 – отсекаемый многоугольник.
Q1…Q4 – отсекатель.
R1…R9 – результет отсечения.

<u>Факт 1.</u> Все точки результата лежат внутри (включая границу) отсекателя.
<u>Факт 2.</u> Для выпуклого отсекателя, для любого ребра вся область лежит по одну сторону от соответствующей прямой, проходящей через ребро.
<u>**Вывод:**</u> Отсекаемый многоугольник можно отсекать по очереди полуплоскостями. соответствующим ребрам отсекателя.

Пусть L – прямая, содержащая ребро отсекателя.
Видимыми назовем все точки, расположенные на прямой, проходящей через ребро отсекателя, или лежащие в полуплоскости, соответствующей области отсекателя, , иначе точки невидимы.

<u>Факт 3.</u> Для ориентированного ребра P_iP_{i+1} отсекаемого многоугольника возможны следующие варианты:

a) Полная видимость. Ребро P_iP_{i+1} целиком лежит в полуплоскости, соответствующей области отсекателя.

b) Полная невидимость. Ребро P_iP_{i+1} целиком лежит вне полуплоскости, соответствующей области отсекателя.

c) Видны P_i и I, где . Выход за границу области.

d) Видны P_{i+1} и I, где . Вход в область.

Алгоритм:
Для удобства на входе - $P_1...P_{N-1}P_N$ ($=P_1...P_{N-1}P_1$)–замкнутый цикл.
Отсечь $P_1...P_{N-1}P_N$ прямой L:

```
{
        for (i=2;i<=N;i++)
        {
                if (P_{i-1}P_i пересекает L)
                {
                        I=intersection(P_{i-1}P_i,L);
                        output (I);
                }
                if (P_i видима) output (P_i);
        }
}
```

<u>«Плюсы» алгоритма:</u>

1) Простота;
2) Надо мало памяти – только 2 списка точек;

3) Работает для любых замкнутых цепочек;

«Минусы» алгоритма:

Годен только для выпуклых отсекателей;