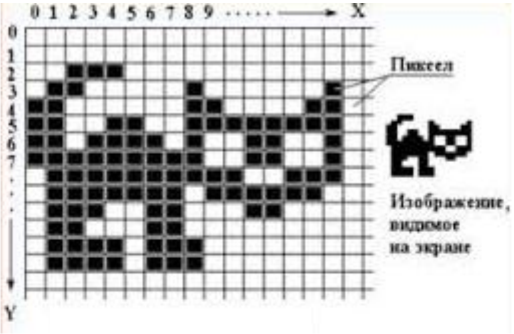
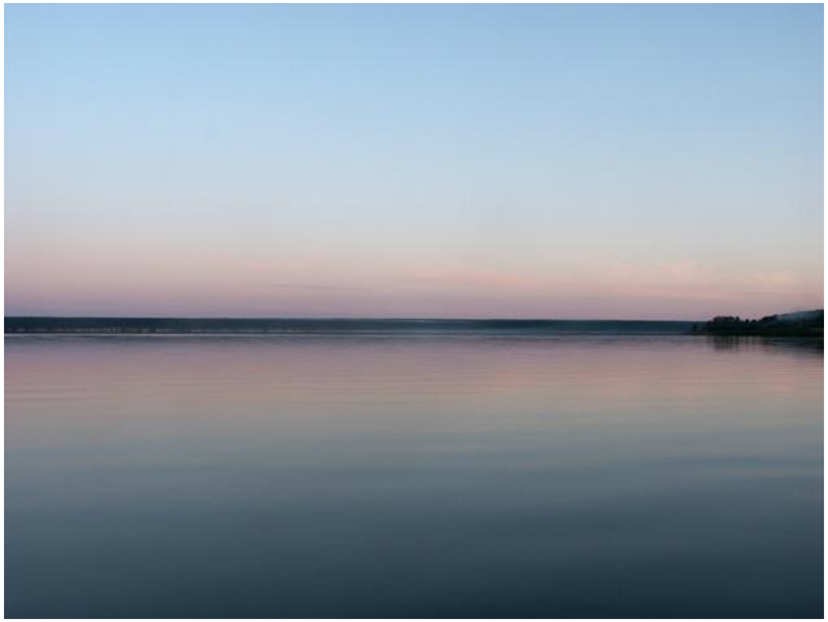
Растровая графика

Растровое изображение — изображение, представляющее собой сетку пикселей или цветных точек (обычно прямоугольных) на мониторе, бумаге и других отображающих устройствах (см. растр).



Важными характеристиками изображения являются:

- Размер изображения в пикселях — может выражаться в виде количества пикселей по ширине и по высоте;
- Количество используемых цветов или глубина цвета (эти характеристики имеют следующую зависимость: $N=2^k$, где N — количество цветов, k — глубина цвета);
- Цветовое пространство (цветовая модель) — RGB, CMYK, XYZ, YCbCr и др.;
- Разрешение изображения — величина, определяющая количество точек (элементов растрового изображения) на единицу площади (или единицу длины).



Компьютерное ***растровое изображение*** представляется в виде прямоугольной матрицы, каждая ячейка которой представлена цветной точкой

Сама сетка получила название растровой карты (**bitmap**), а ее единичный элемент (квадратная ячейка) называется ***пикселем*** (от английского pixel - picture element).

Растровая карта представляет собой набор (массив) троек чисел: две координаты пикселя на плоскости и его цвет

При оцифровке изображения оно делится на такие крошечные ячейки, что глаз человека их не видит, воспринимая все изображение как целое.

Пиксели подобны зернам фотографии и при значительном увеличении они становятся заметными.

Растровое изображение ближе к фотографии поскольку позволяет более точно воспроизводить основные характеристики фотографии: освещенность, прозрачность и глубина резкости

В отличии от векторных изображений, при создании растровых изображений математические формулы не используются, поэтому для синтеза растровых изображений необходимо задавать разрешение (resolution) и размеры изображения

С развитием компьютерной техники возможное разрешение увеличивается. (VGA - 640x480, SVGA - 1024x768, 1280x1024, 1600x1280, 1980x1080)

Растровые изображения можно получить и непосредственно в программах растровой графики или в программах векторной графики путем преобразовании векторных изображений в растровые

Преимущества

- Растровая графика позволяет создать практически любой рисунок, вне зависимости от сложности;
- Распространённость — растровая графика используется сейчас практически везде: от маленьких значков до плакатов;
- Высокая скорость обработки сложных изображений, если не нужно масштабирование;
- Растровое представление изображения естественно для большинства устройств ввода-вывода графической информации, таких как мониторы (за исключением векторных устройств вывода), матричные и струйные принтеры, цифровые фотоаппараты, сканеры, а также сотовые телефоны.

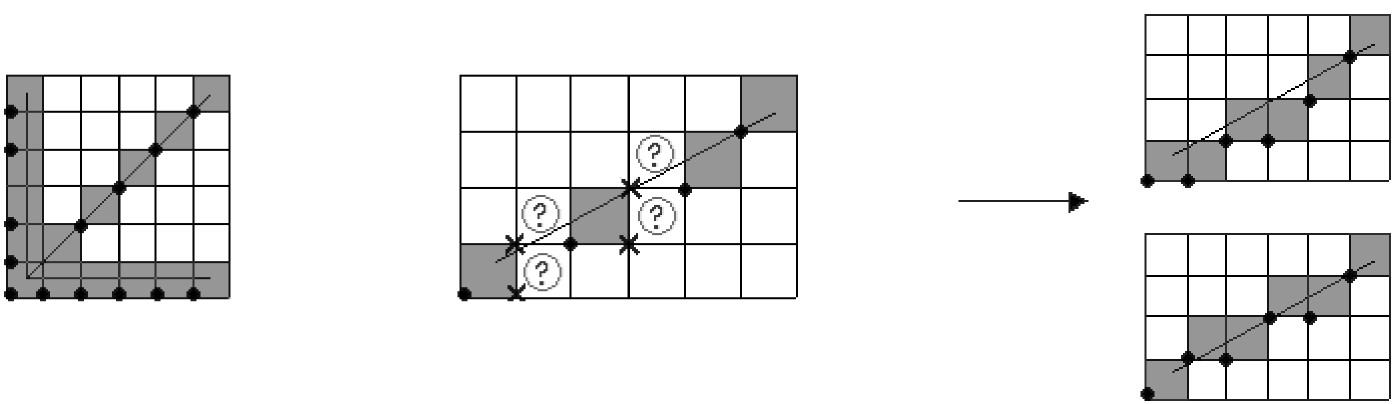
Недостатки

- Большой размер файлов у простых изображений;
- Невозможность идеального масштабирования;
- Невозможность вывода на печать на векторный графопостроитель.

Алгоритмы вычерчивания отрезков

Для работы с устройствами растровой графики нужны специальные методы генерации прямых и кривых линий, закраски многоугольников, рисования текста.

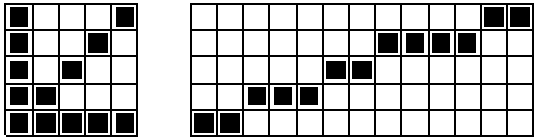
Для горизонтальных, вертикальных и проведенных под углом в 45 градусов отрезков выбор растровых элементов очевиден. При любой другой ориентации выбрать пикселы, наилучшим образом аппроксимирующих отрезок, труднее.



Основные требования к алгоритмам:

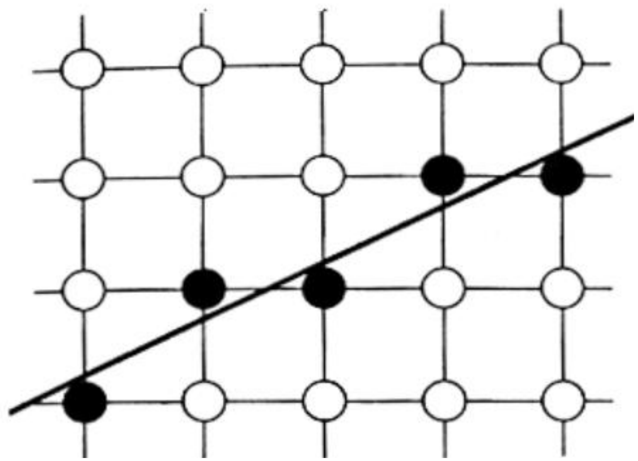
- отрезки должны выглядеть прямыми;
- отрезки должны начинаться и заканчиваться в заданных точках;
- яркость вдоль отрезка должна быть постоянной и не зависеть от наклона и длины;
- скорость выполнения.

Существует несколько алгоритмов выполняющих эту задачу, например, цифровой дифференциальный анализатор и алгоритм Брезенхема.



Цифровой дифференциальный анализатор

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



Уравнение прямой

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1), x \in [x_1, x_2]$$

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

$$\frac{dy}{dx} = const \quad \text{или} \quad \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

Решение представляется в виде:

$$y_{i+1} = y_i + \Delta y$$

$$y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1} \Delta x$$

где x_1, y_1 и x_2, y_2 - концы разлагаемого отрезка и y_i - начальное значение для очередного шага вдоль отрезка. Ниже приводится простой алгоритм, работающий в первом квадранте:

Процедура разложения в растр отрезка простейшим методом.

Abs (x) - функция взятия модуля числа *x*.
Round (x) - функция округления вещественного числа *x* до целого.
Plot (x,y) - функция вывода точки с координатами *(x,y)* на экран.

```
Длина = x2 - x1;                   // аппроксимируем длину отрезка

Dx = 1;
Dy = abs((y2-y1)/(x2-x1));         // полагаем Dx равным единице растра

x = x1; y = y1; i = 1;             // устанавливаем координаты начальной точки

while (i < Длина);
   Plot (x, Round(y));
   x = x + Dx
   y = y + Dy                      // начало основного цикла
    i = i + 1;
end while
finish
```

```
double Dx , Dy;
if (abs(x2-x2) >= abs(y2-y1)) Длина = abs(x2-x1);
   else                                          Длина = abs(y2-y1);
Dx=(x2-x1)/Длина;   Dy=(y2-y1)/Длина;     // или Dx или Dy равно 1

x=x1+0.5*sign(Dx); y=y1+0.5*sign(Dy);      // начальные точки
i=1;
while (i <= Длина) { setpixel(int(x), int(y)); x=x1+Dx; y=y1+Dy; i=i+1; }
```

Провести отрезок из точки (0,0) в (5,5)

x1=y1=0; x2=y2=5; Dx = Dy =1; => (0,0), (1,1), (2,2), (3,3), (4,4)

Провести отрезок из точки (0,0) в (-8, -4)

x1=y1=0; x2=-8. y2=-4; Dx=-1; Dy =-0.5; =>
(-1,-1), (-2,-1), (-3,-2), (-4,-2), (-5,-3), (-6,-3), (-7,-4) (-8,-4)

во-втором случае не хватает точки (0,0), т.е. результат зависит от ориентации.

Однако, в некоторых случаях в отрезке, разложенным в растр простейшим методом, могут появиться разрывы, например, если выбрать такой отрезок, у которого разброс по оси Y больше разброса по оси X в несколько раз.

Вдобавок предложенный алгоритм имеет тот недостаток, что он использует вещественную арифметику.

Алгоритм Брезенхема

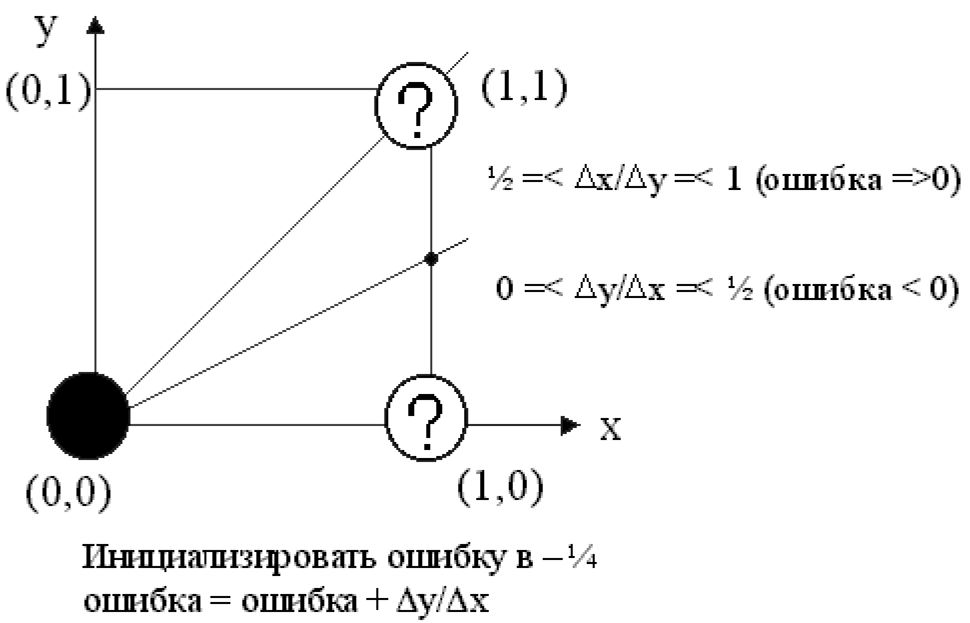
Алгоритм выбирает оптимальные растровые координаты для представления отрезка.

В процессе работы одна из координат - либо x, либо y (в зависимости от углового коэффициента) - изменяется на единицу. Изменение другой координаты (на 0 или 1) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Такое расстояние мы назовем **ошибкой**.

Алгоритм построен так, что требуется проверить лишь знак этой ошибки. На рис. это иллюстрируется для отрезка в первом октанте, т.е. для отрезка с угловым коэффициентом, лежащим в диапазоне от 0 до 1.

Из рисунка можно заметить, что если угловой коэффициент отрезка из точки (0,0) больше, чем 1/2, то пересечение с прямой x = 1 будет расположено ближе к прямой y = 1, чем к прямой y = 0.

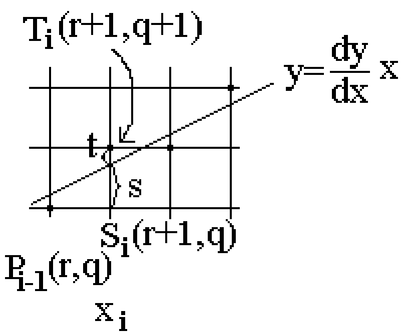
Следовательно, точка растра (1,1) лучше аппроксимирует ход отрезка, чем точка (1,0). Если угловой коэффициент меньше 1/2, то верно обратное.



Для углового коэффициента, равного 1/2, нет какого либо предпочтительного выбора. В данном случае алгоритм выбирает точку (1,1).

Для простоты будем считать, что тангенс угла наклона принимает значение от 0 до 1 (1 октант - **от 0 градусов до 45**). Затем распространим на общий случай.

В алгоритме используется управляющая переменная d_i; , которая на каждом шаге пропорциональна разности между s и t. Как найти эту переменную, через предыдущие значения?



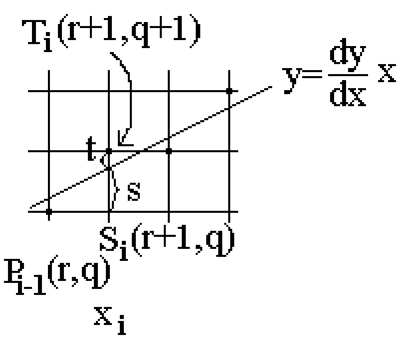
На рисунке приведен i-ый шаг, когда пиксел P_{i-1}, уже найден, как ближайший к отображенному отрезку, и теперь требуется определить какой из пикселов должен быть установлен T_i, или S_i, . Если $s<t$ то S_i, в противном случае T_i, (если $s-t<0 => S_i$, иначе T_i)

Отрезок проводится из точки (x1,y1) в точку (x2,y2). Пусть первый отрезок находится ближе к началу координат, тогда переносим начало отрезка в (0,0). Конечная точка имеет координаты (Dx=x2-x1, Dy=y2-y1).

Уравнение прямой теперь имеет вид

$$y = \frac{\Delta y}{\Delta x} x$$

Обозначим координаты P_{i-1} (после переноса) через (r,q), тогда $S_i=(r+1,q)$, $T_i=(r+1,q+1)$, если $s<t$ выбираем S_i иначе T_i .



$$s=\frac{\Delta y}{\Delta x}(r+1)-q,\quad t=q+1-\frac{\Delta y}{\Delta x}(r+1)$$

Далее

$$s-t=2\frac{\Delta y}{\Delta x}(r+1)-2q-1$$

$$Dx(s-t)=2(rDy-qDx)+2Dy-Dx=d_i$$

Т.к. $Dx>0$, то выражение для определения $Dx(s-t)$ можно использовать для проверки знака $(s-t)$.

Поскольку $r=x_{i-1}$, $q=y_{i-1}$

$d_i = 2x_{i-1}\,Dy - 2\,y_{i-1}\,Dx + 2\,Dy - Dx$, добавляя по 1 к каждому индексу получим

$d_{i+1}= 2x_i\,Dy - 2\,y_i\,Dx + 2\,Dy - Dx$, вычитаем одно из другого

$d_{i+1} - d_i = 2Dy(x_i - x_{i-1}) - 2\,Dy(y_i - y_{i-1}) - Dx$, учитывая, что $(x_i - x_{i-1}) = 1$

$$d_{i+1}= d_i + 2\,Dy - 2\,Dy(y_i - y_{i-1})$$

Таким образом, получен интерактивный способ вычислений d_{i+1} по предыдущему значению d_i.

Начальное значение d_1 можно получить из

$d_i = 2x_{i-1}Dy - 2\,y_iDx + 2\,Dy - Dx$ при $i=1$ с учетом, что $(x_0,y_0)=(0,0)$

$d_1 = 2\,Dy - Dx$

Последовательно вычисляя d_i определяем

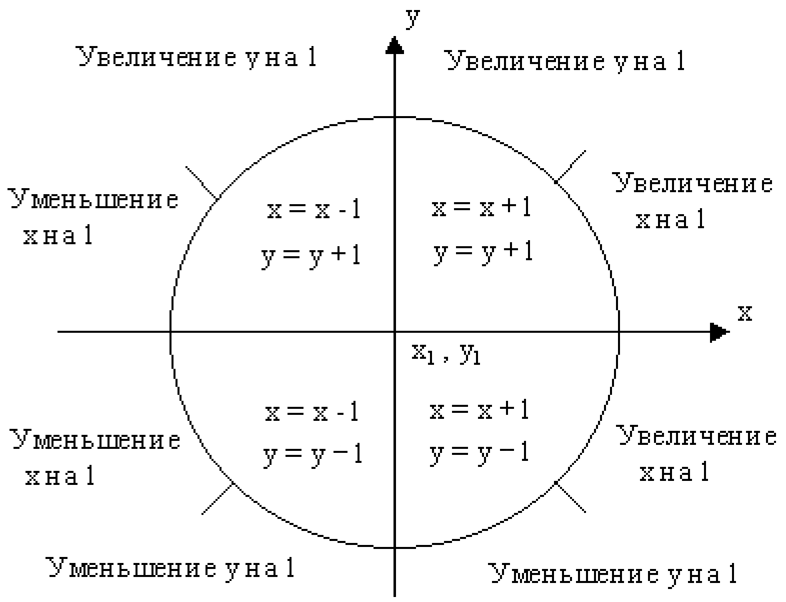
если $d_i >= 0$, выбираем T_i, $=> y_i = y_{i-1} + 1$, $d_{i+1} = d_i + 2\,(Dy - 2Dy)$
если $d_i < 0$, выбираем S_i, $=> y_i = y_{i-1}$, $d_{i+1} = d_i + 2\,Dy$

Вычисления требуют только операций сложения, вычитания и сдвиг влево (умножение на 2 реализуется как сдвиг влево).

Обобщение алгоритма на другие октанты очень просто.

Чтобы реализация алгоритма Брезенхема была полной необходимо обрабатывать отрезки во всех октантах. Модификацию легко сделать, учитывая в алгоритме номер квадранта, в котором лежит отрезок и его угловой коэффициент.

Когда абсолютная величина углового коэффициента больше 1, y постоянно изменяется на единицу, а критерий ошибки Брезенхема используется для принятия решения об изменении величины x. Выбор постоянно изменяющейся (на +1 или -1) координаты зависит от квадранта



```
void line( int x0, int y0, int x1, int y1, int c)
{
     int dx, dy, ch=0, i=0, e, dx2, dy2;

     x1 -= x0;      dx = abs(x1);
     y1 -= y0;      dy = abs(y1);
     if( !x1 && !y1 ) return;                    // Если начало совпадает с концом отрезка
     if( x1 ) x1 = x1 < 0 ? -1 : 1;
     if( y1 ) y1 = y1 < 0 ? -1 : 1;

    if( dy > dx ) {  int t = dy; dy = dx; dx = t;  ch=1; }    // меняем местами x и y

    dx2 = dx<<1; dy2 = dy<<1;                              // dx2 = 2*dx; dy2 = 2*dy;
    e = dy2 - dx ;                                         // Начальное значение ошибки;
    for( i=0; i < dx ; ++i )
        {
            putpixel( x0, y0, c);
            if ( e > 0 ) {  if( ch ) x0 += x1; else y0 += y1;   e -= dx2;   }
             else        {  if( ch ) y0 += y1; else x0 += x1;   e += dy2;  }
        }
}
```

Растровая развертка окружностей

Существует несколько очень простых, но неэффективных способов преобразования окружностей в растровую форму.

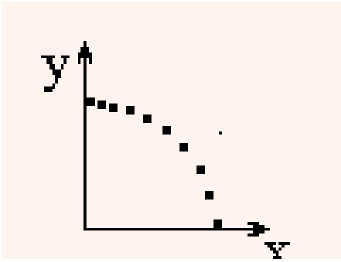
Уравнение окружности с центром в начале координат $x^2+y^2=R^2$.

Решая это уравнение относительно y получим $y = \pm\sqrt{R^2 - x^2}$

Чтобы отобразить 1/4 часть окружности, можно увеличивать x с шагом от 0 до R и на каждом шаге вычислять y. Остальные четверти отображаются симметрично.

Другим способом, является вычисление точек окружности с помощью параметрических уравнений

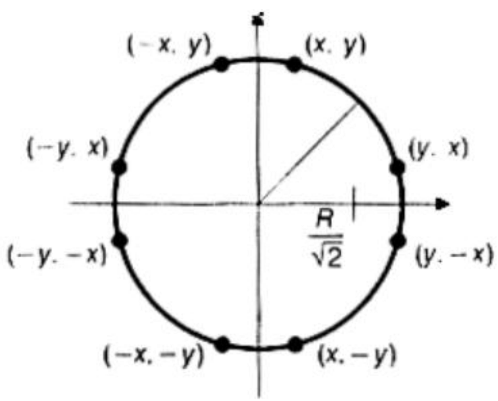
$x=R\cdot\cos(\varphi)$, $y=R\cdot\sin(\varphi)$ путем пошагового изменения φ.



Эти методы не эффективны поскольку:

- в них входят операции с плавающей запятой;
- при значениях x близких к R появляются большие незаполненные промежутки, т.к. тангенс угла наклона касательной к окружности стремится к бесконечности.

Процесс преобразования окружности в растровую форму можно улучшить, если полнее использовать симметрию окружности. Если точка лежит на окружности, очень просто вычислить семь других точек, принадлежащих окружности.

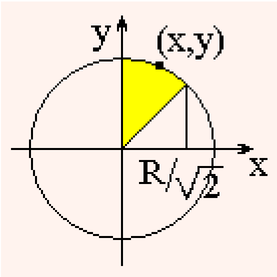


Поэтому, если найти значения в одном октанте, можно получить еще семь дополнительных точек с помощью следующей процедуры

Circ_8(x,y,c) { P(x,y,c); P(y,x,c); P(y,-x.c); P(-x,y,c); P(-x,-y,c); P(-y,-x,c); P(-x,y,c); P(y,-y,c); }

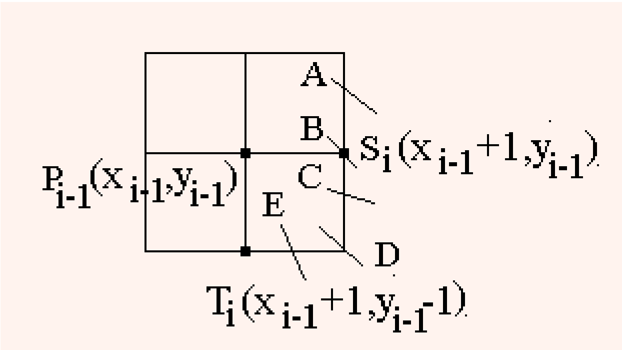
Алгоритм Брезенхема для окружностей

Брезенхем разработал пошаговый генератор дуг, который более эффективен, чем каждый из рассмотренных выше методов. Генерируются точки окружности с центром в начале координат путем пошагового обхода вокруг окружности.



Рассмотрим алгоритм лишь для дуги окружности 45 градусов от x=0 до R/sqrt(2) и воспользуемся процедурой Circ_8(x,y,c) для отображения точек всей окружности.

Рассмотрим небольшой участок сетки, а также возможные способы прохождения истинной окружности через сетку. A-E соответствует дуге в 45 градусов.

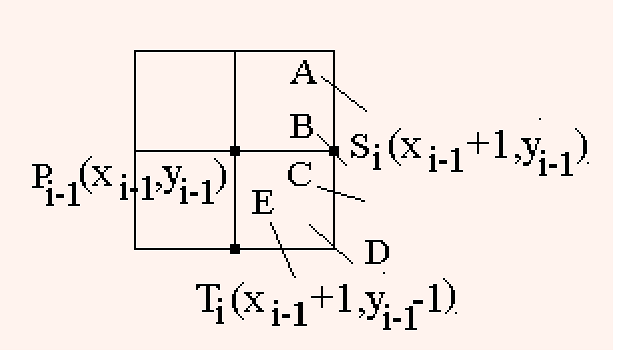


Пусть точка P_i была выбрана как ближайшая к окружности при $x=x_i$.

Теперь найдем, какая из точек (T_i или S_i) расположена ближе к окружности при $x= x_{i-1} +1$.

Выражения

$D(S_i) = [\ (x_{i-1}+1)^2 + (y_{i-1})^2\ ]\ - R^2$
$D(T_i) = [\ (x_{i-1}+1)^2 + (y_{i-1}-1)^2\ ]\ - R^2$



описывают разность между квадратами расстояний от истиной окружности до S_i, T_i.

Если $abs(D(S_i)) >= abs(D(T_i))$, то T_i ближе к реальной окружности, чем S_i,.
Иначе S_i ближе.

Пусть $d_i=abs(D(S_i)) - abs(D(T_i))$, при $d_i >= 0 \Rightarrow T_i$ иначе S_i,.

Рассмотрим случаи от A до E.
Для C => $D(S_i) > 0$ поскольку S_i, лежит за пределами окружности
$D(T_i) < 0$ поскольку T_i внутри окружности
Следовательно: $d_i= D(S_i)+ D(T_i)$, если $d_i >= 0 \Rightarrow T_i$ иначе S_i,.

A и B => $D(T_i) < 0$, $D(S_i) <= 0$, $d_i < 0 \Rightarrow S_i$,.
D и E => $D(S_i) > 0$ $D(T_i) >= 0$ $d_i >= 0$

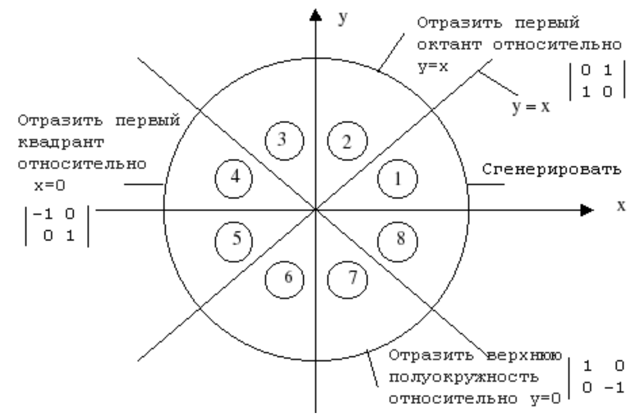
Для вычисления управляющей переменной d_i требует несколько умножений. Однако алгебраические преобразования показывают, что $d_1 = 3-2R$.

Если выбирается S_i, (при $d_i < 0$) $d_{i+1} = d_i + 4 x_{i-1} + 6$.

Если выбирается T_i (при $d_i >= 0$) $d_{i+1} = d_i + 4 (x_{i-1} - y_{i-1})+10$.

```
// Circ_8(x,y,c) { P(x,y,c); P(y,x,c); P(y,-x.c); P(-x,y,c); P(-x,-y,c); P(-y,-x,c); P(-x,y,c); P(y,-x,c); }

static void circ_8(int ix0, int iy0, int x, int y, int col )
{
  int x1=ix0+x, y1=iy0+y, x2=ix0-x, y2=ix0-y;
  putpixel(ix0, iy0, x1, y1, col);  P(n, ix0, iy0, y1, x1, col);
  putpixel(ix0, iy0, y1, x2, col);  P(n, ix0, iy0, x2, y1, col);
  putpixel(ix0, iy0, x2, y2, col);  P(n, ix0, iy0, y2, x2, col);
  putpixel(ix0, iy0, x2, y1, col);  P(n, ix0, iy0, y1, x2, col);
}
```




```
void circ(int n, int ix0, int iy0, int ir, int col)
{
  int x,y,d;
  x=0; y=ir; d=3-(ir<<1);
  while (x < y)
   {
     circ_8( ix0, iy0, x, y, col);
     if (d < 0) d+=(x<<2)+6; else d+=((x-y)<<2)+10, --y;
     x++;
    }
   if (x == y) circ_8( ix0, iy0, x, y, col);
}
```