Генерация векторов.

Генерация вектора - соединение двух точек изображения отрезком прямой.

Рассмотрим:

1. Простой алгоритм вывода линии;

2. Два алгоритма **ЦДА** - цифрового дифференциального анализатора **(DDA - Digital Differential Analyzer)** для генерации векторов - обычный и несимметричный;

3. Алгоритм **Брезенхема** для генерации векторов.

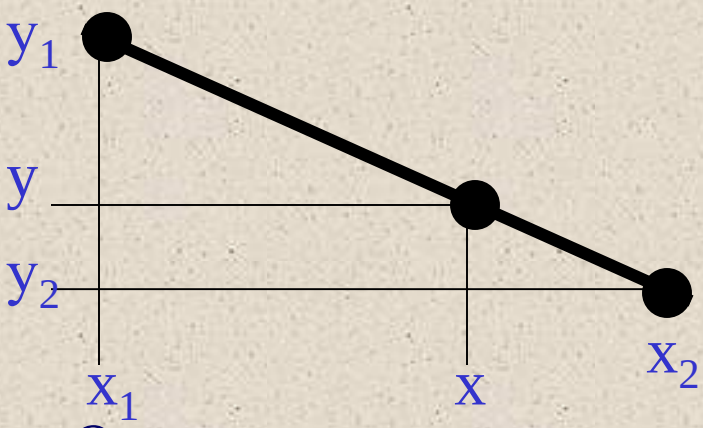


Рис. 1. Отрезок прямой

Отношение катетов для подобных треугольников:

$$\frac{x - x_1}{y - y_1} = \frac{x_2 - x_1}{y_2 - y_1}$$

-->

$$x = x_1 + (y - y_1)\frac{x_2 - x_1}{y_2 - y_1}, \quad x = f(y)$$

$$y = y_1 + (x - x_1)\frac{y2 - y1}{x_2 - x_1}, \quad y = F(x)$$

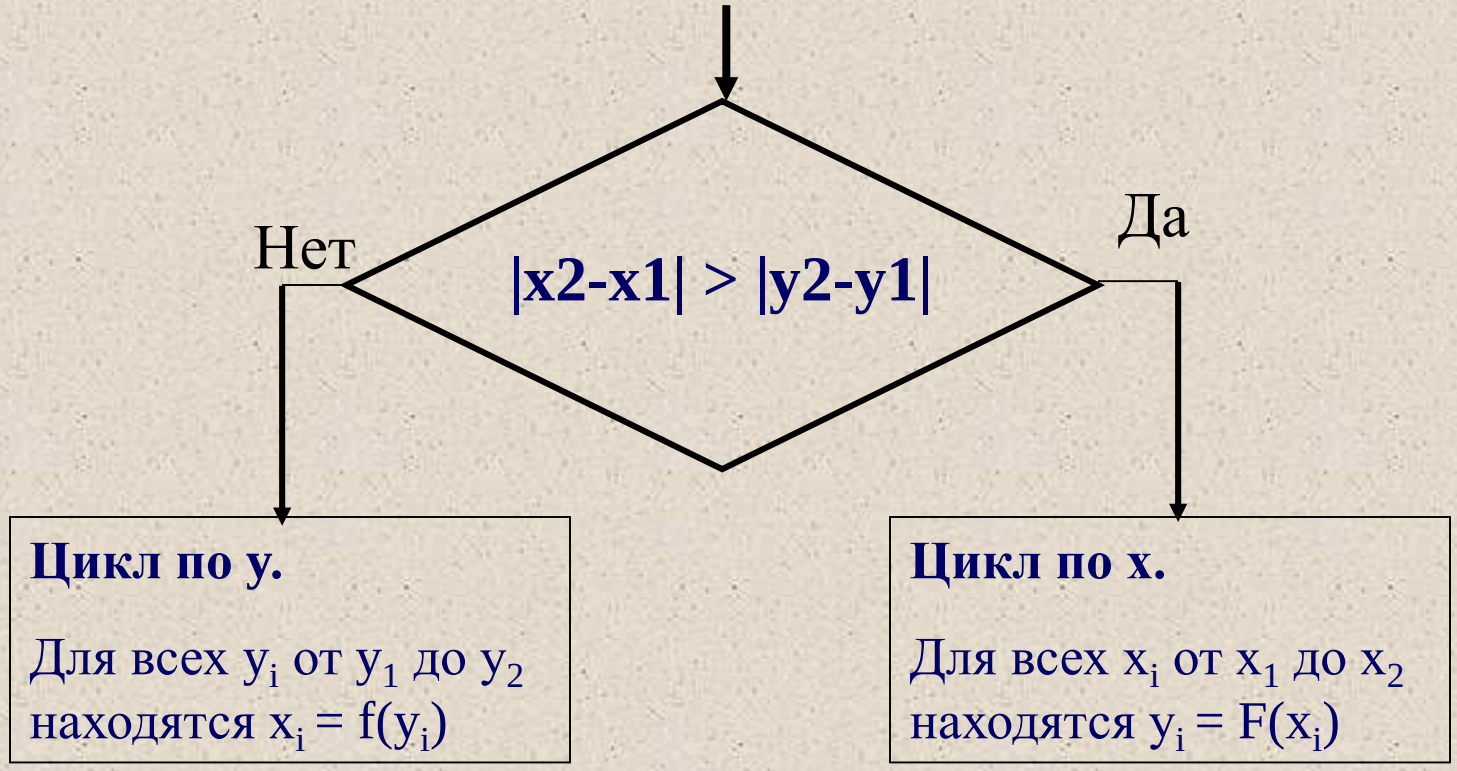


Рис. 2. Алгоритм вывода отрезка прямой линии

```
for (x=x1; x<=x2; x++)
{       y = y1 + ((x-x1) * (y2-y1))/(x2-x1);
        putpixel(x,y);
}
```

```
float k;
k = (float) (y2-y1)/(float) (x2-x1);
for (x=x1; x<=x2; x++)
{
        y = y1 + (float) (x-x1)*k;
        putpixel(x,y);
}
```

Раскроим скобки:

y=y1+(x-x1)*k = y1 + x*k - x1*k,

Заметим, что **(y1-x1*k)** - является константой, эти операции выносим из цикла.

```
float yy, k;
k  =(float)(y2-y1)/(float)(x2-x1);
yy=(float) y1 - (float) x2*k;
for (x=x1; x<=x2; x++)
{
        y = yy + (float) x*k;
        putpixel(x,y);
}
```

Заметим: разность $x_{i+1} - x_i = 1$,

а разность $(y_{i+1} - y_i) = x_1+(x_{i+1} - x_1)k - x_1 - (x_i - x_1)k =$
$= (x_{i+1} - x_i)k = 1*k = k$, **т.е. константа k.**

```
float k;
k  =(float)(y2-y1)/(float)(x2-x1);
y = y1;
for (x=x1; x<=x2; x++)
{
        putpixel(x,y);
        y + = k;
}
```

Положительные черты прямого вычисления:

1. Простота, ясность построения алгоритма.

2. Возможность работы с нецелыми значениями координат отрезка.

```
float k;
k  =(float)(y2-y1)/(float)(x2-x1);
y = y1;
for (x=x1; x<=x2; x++)
{
        putpixel(x,y);
        y + = k;
}
```

Недостатки:

1. Использование операций с плавающей точкой или целочисленных операций умножения и деления затрудняет аппаратную реализацию алгоритма.

2. При вычислении координат путем добавления приращений накапливается ошибка. На последнем шаге *y* может не совпасть с *y2* !

Алгоритмы **ЦДА** - **Ц**ифрового Дифференциального **А**нализатора **(DDA - Digital Differential Analyzer)**.

Алгоритмы **Брезенхема.**

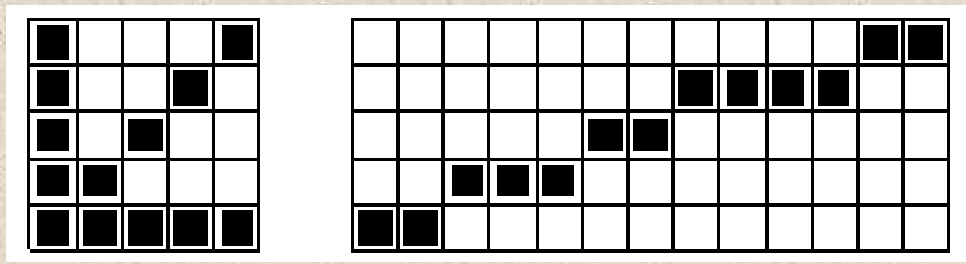
Общие требования к изображению отрезка:

· Точное позиционирование концов отрезка.

· Отрезки должны выглядеть прямыми.

· Яркость вдоль отрезка должна быть постоянной и не зависеть от длины и наклона.

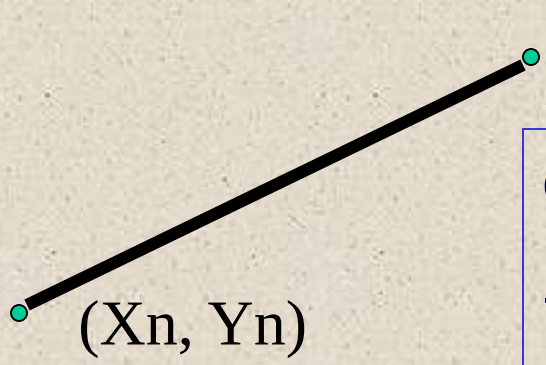
· Должны быть поддержаны требуемые атрибуты.



Имеем реально:

- Концы отрезка в общем случае располагаются на пикселах.
- Отрезок аппроксимируется набором пикселов: вертикальных, горизонтальных и под 45°.
- Яркость для различных отрезков различна (расстояние между центрами пикселов для вертикального отрезка и отрезка под 45° различно.

Цифровой дифференциальный анализатор (ЦДА)



$$\frac{dY}{dX} = \frac{Py}{Px}$$

$Py = Yk - Yn$

$Px = Xk - Xn$

а) Обычный (симметричный) алгоритм ЦДА.

N - количество узлов N, используемых для аппроксимации отрезка.

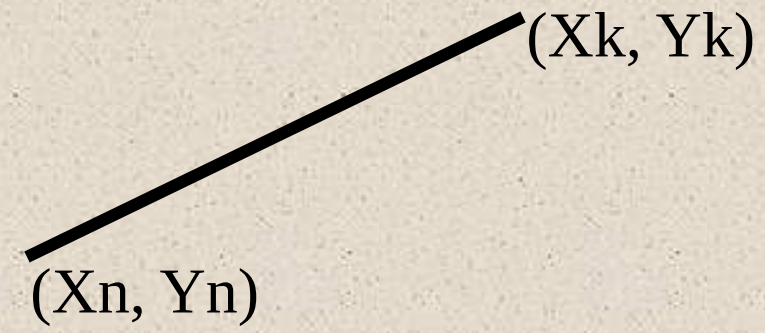
За N циклов вычисляются координаты очередных узлов:

$X_0 = Xn$; $X_{i+1} = X_i + Px/N$.

$Y_0 = Yn$; $Y_{i+1} = Y_i + Py/N$.

Расчетные X_i и Y_i преобразуют в целочисленные значения координат пиксела: либо округлением, либо отбрасыванием дробной части.

Недостатки симметричного алгоритма:



$$X_{i+1} = X_i + Px/N$$
$$Y_{i+1} = Y_i + Py/N$$

1. Точки могут прописываться дважды, что увеличивает время построения вектора.

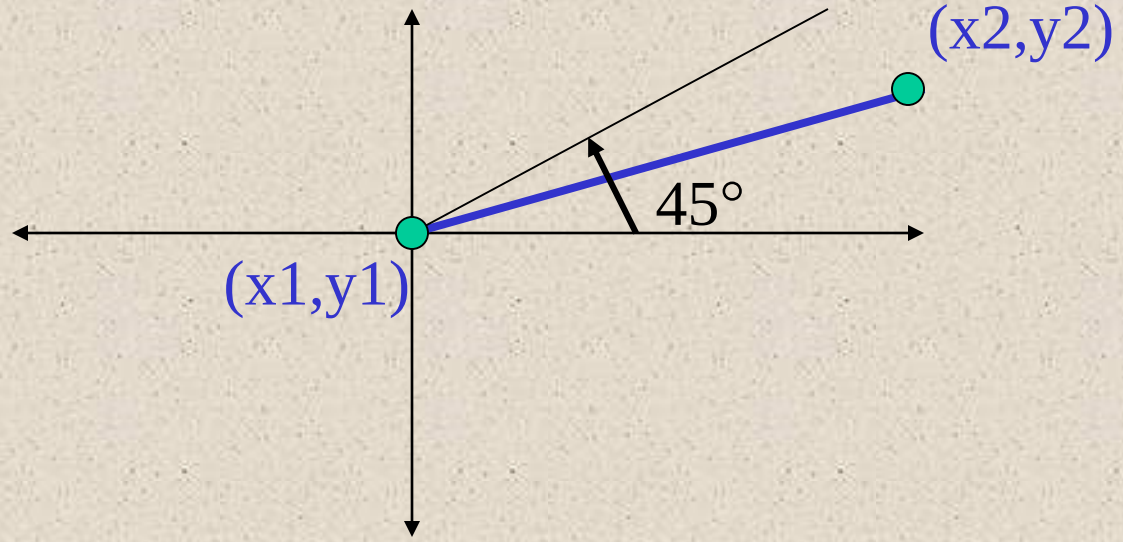
2. Вычисляются одновременно обе координаты, не определяется предпочтение одной координаты, построенный вектор имеет “грубые» ступени.

3. Предварительное вычисление количества узлов N.

Б) Несимметричный алгоритм ЦДА

1. Определяется, что больше, Px или Py ?

Пусть Px, Py > 0 и Px > Py.



Координата по X увеличивается на 1 (x2-x1)/Px раз, а координата Y тоже столько же раз, но на величину Py/Px.

Т.е. количество узлов аппроксимации равно числу пикселов вдоль наибольшего приращения.

1.Вычислить приращения координат:

Px= x2 - x1;

Py= y2 - y1;

2. Занести начальную точку отрезка

PutPixel (x1, y1);

3. Сгенерировать отрезок:

```
while (x1 < x2) {
  x1:= x1 + 1.0;
  y1:= y1 + Py/Px;
  PutPixel (x1, y1);
}
```

x1=0+1.0=**1.0**

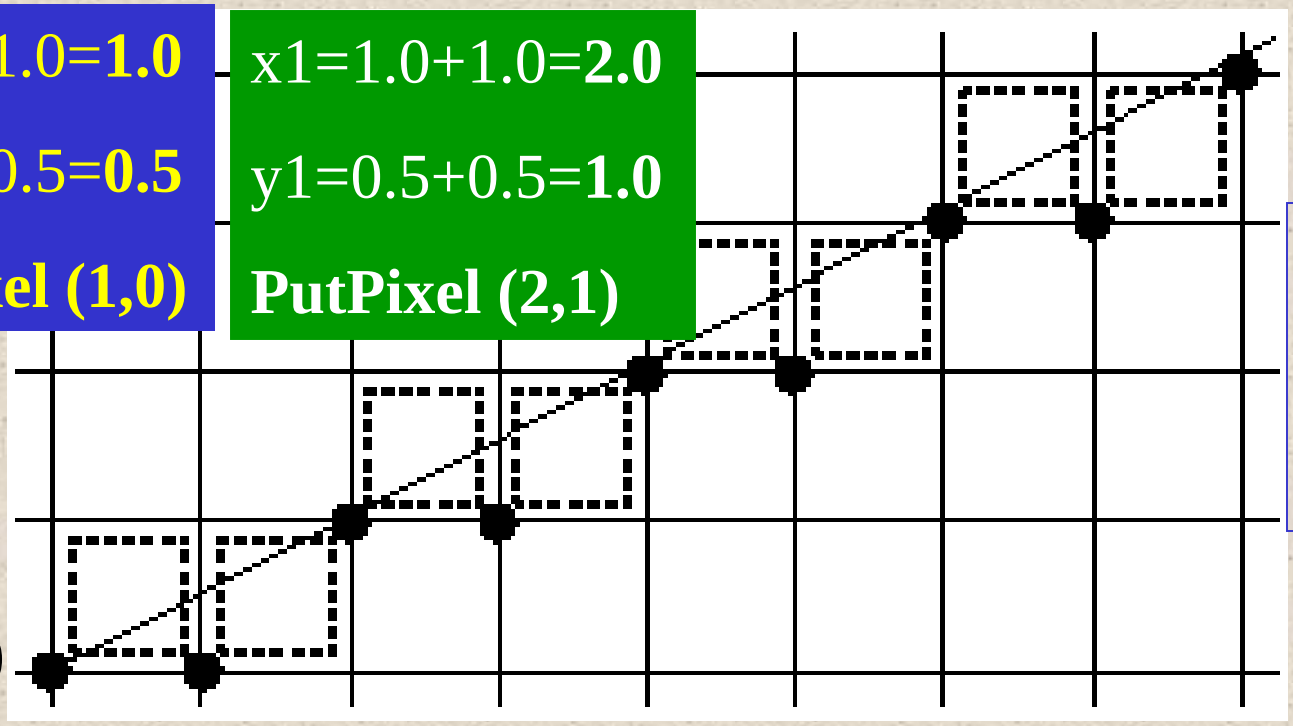
y1=0+0.5=**0.5**

PutPixel (1,0)

x1=1.0+1.0=**2.0**

y1=0.5+0.5=**1.0**

PutPixel (2,1)



Px=8

Py=4

Py/Px=0.5

Пример

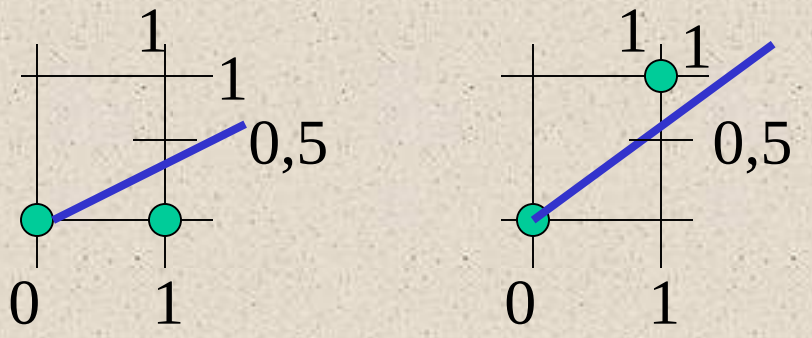
Алгоритм Брезенхема

Общий недостаток ЦДА-алгоритма - операция деления, что не желательно при аппаратной реализации.

Брезенхем (Brezenhem) предложил (1965 г.) алгоритм, не требующий деления и минимизирующий отклонение сгенерированного образа от истинного отрезка.

Идея алгоритма:

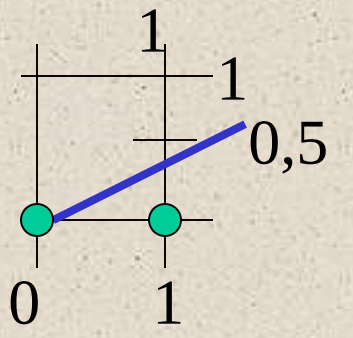
Начальная точка (0,0). Если угловой коэффициент прямой < 1/2, то следующая точка (1,0), иначе точка (1,1).

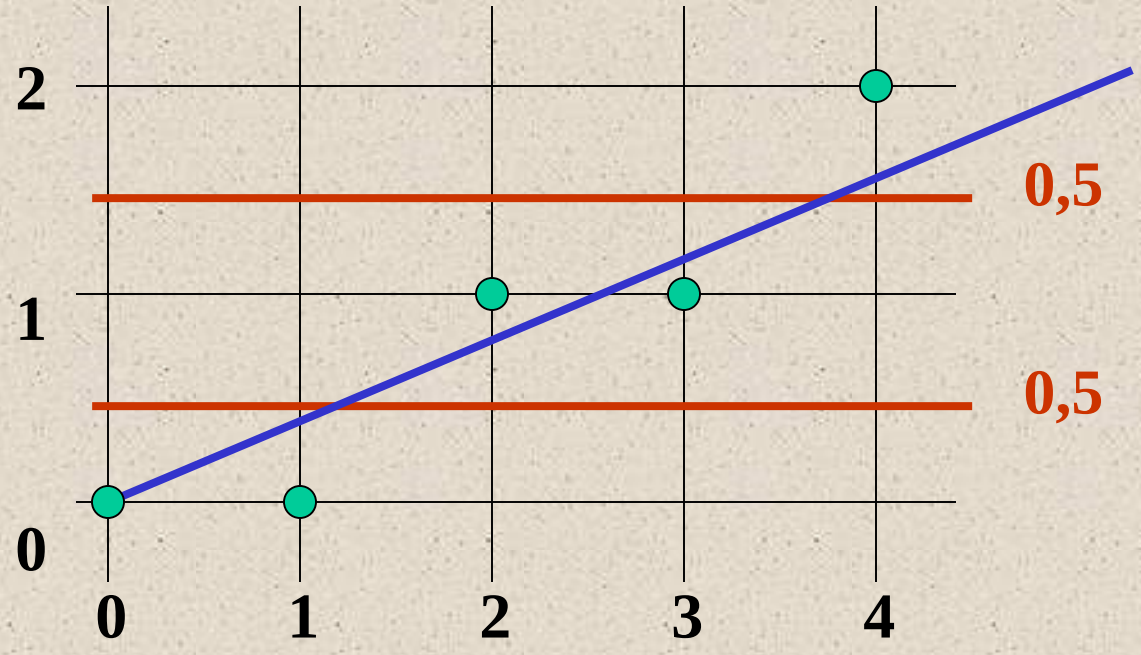


Для принятия решения, куда заносить очередной пиксел, вводится величина отклонения E точкой позиции от середины между двумя возможными растровыми точками в направлении наименьшей относительной координаты.

Знак E - критерий для выбора ближайшей растровой точки.

Если E<0, то точное Y-значение округляется до последнего наименьшего целочисленного значения Y, т.е. Y-координата не меняется по сравнению с предыдущей точкой. В противном случае Y увеличивается на 1.





Угол наклона <1.

Шаг 1. E1 = Py/Px - 1/2 < 0,

PutPixel (1,0)

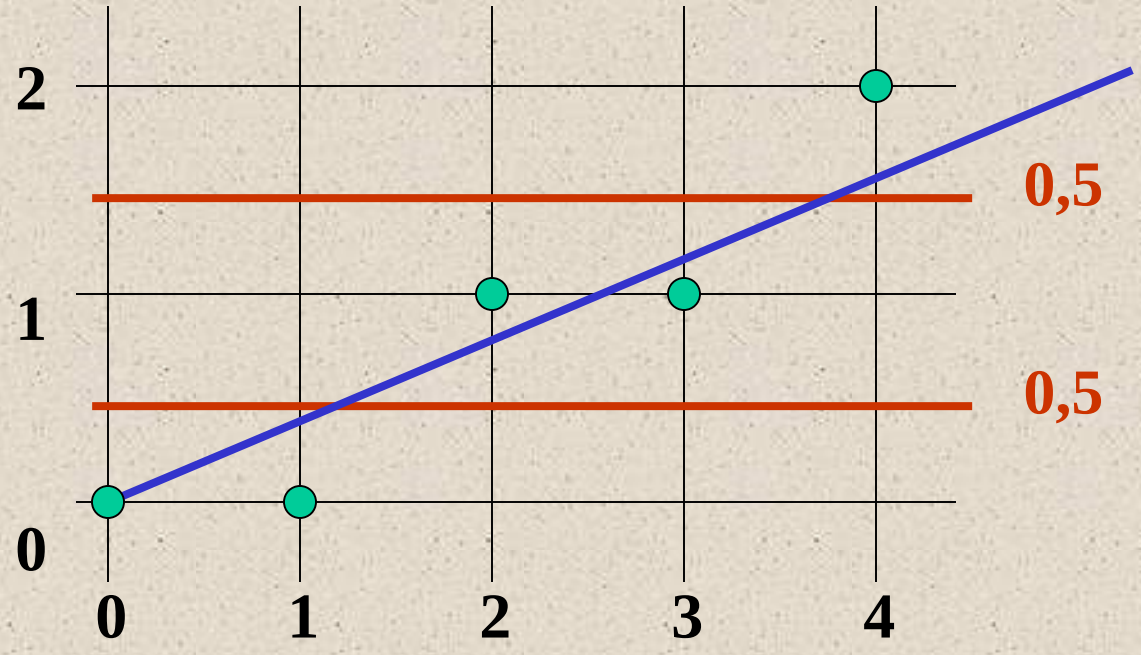
Шаг 2. E2 = E1 + Py/Px >0,

PutPixel (2,1)

E2 = E2 - 1 (*Из накопленного отклонения вычитается 1)*

Шаг 3. E3 = E2 + Py/Px < 0,

PutPixel (3,1)



Угол наклона <1.

Обозначим: X, Y - растровые точки, x, y -точки вектора.

Получим : E1 = y1 - 1/2 = dY/dX - 1/2.

Если E1>0

Ближайшая точка:

X1 = X0 +1; Y1 = Y0 +1:

E2 = E1 + Py/Px - 1;

Если E1<0

Ближайшая точка:

X1 = X0 + 1: Y1 = Y0:

E2 = E1 + Py/Px.

Так как интересует только знак E, то можно избавиться от неудобного деления умножением E на 2Px:

E1 = Py/Px - 1/2 (*см. Шаг 1*)

E1 = 2Px Py/Px - 2Px/2 = 2Py - Px

И так:

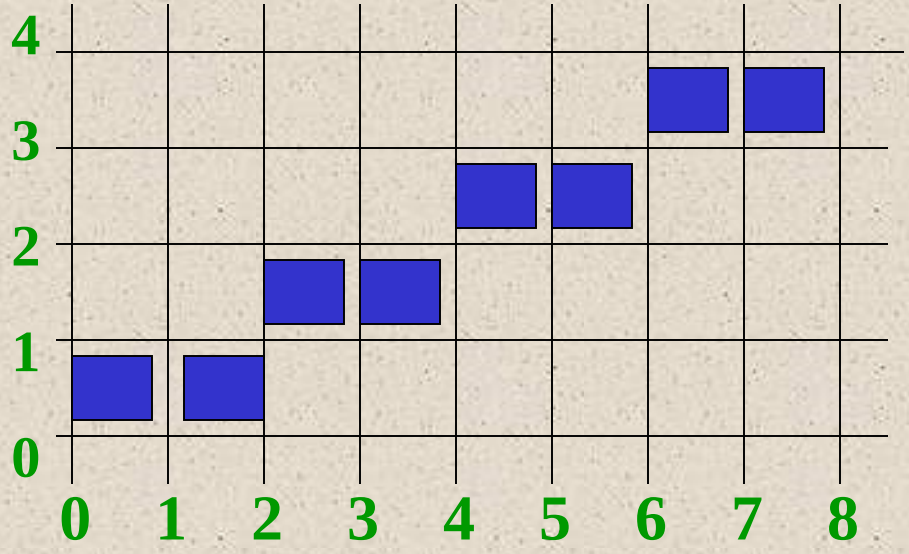
E1 = 2Py - Px

E1>0 E2 = E1 + 2(Py - Px)

E<=0 E2 = E1 + 2Py

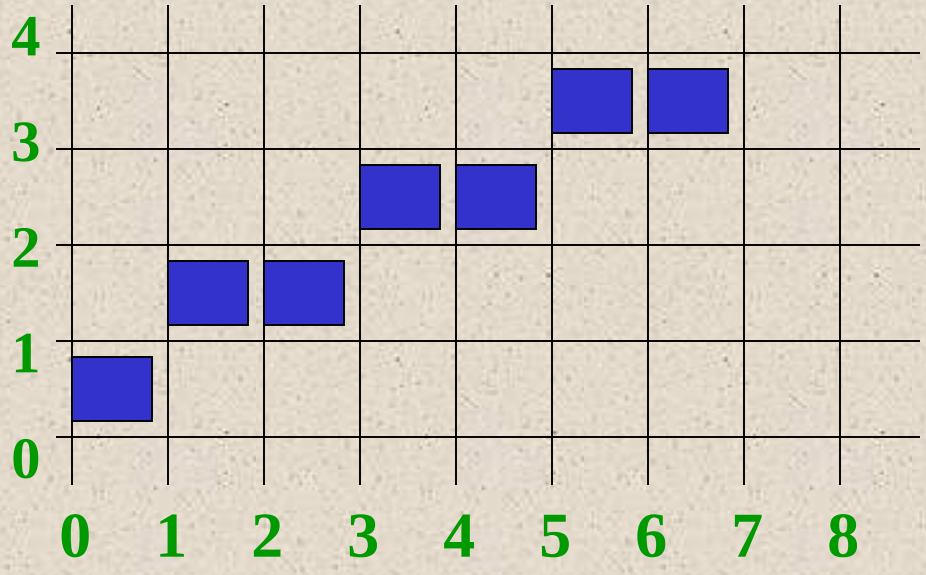
Получим алгоритм, в котором используются только целые числа, сложение, вычитание и сдвиг.

```
X= x1;
Y= y1;
Px= x2 - x1;
Py= y2 - y1;
E= 2*Py - Px;
l= Px;
putpixel(X, Y);          // Первая точка вектора
for (int i= 0; i<l; i++)
{    if (E ≥ 0)
     {   X= X + 1;
         Y= Y + 1;
         E= E + 2*(Py - Px);
      }
     else  X= X + 1;
     E= E + 2*Py;
     putpixel(X, Y);     // Очередная точка вектора
}
```



ЦДА - алгоритм

(несимметричный)



Алгоритм Брезенхема