Машинная графика
Computer Graphics

Лекция 2.

«Двухмерные алгоритмы»

План лекции

- Двухмерные преобразования
- Перенос на плоскости
- Поворот относительно начала системы координат
- Поворот относительно произвольной точки
- Цепной код и дифференциальный цепной код
- Алгоритмы изображения прямой линии (отрезка)
- Алгоритм цифрового дифференциального анализатора (Digital Differential Analyzer)

Двухмерные преобразования

В МГ существует возможность генерировать двухмерные (2D) и трёхмерные (3D) изображения.

При работе на плоскости возможны следующие двухмерные преобразования:

- перенос (трансляция),
- поворот,
- масштабирование,
- отражение,
- сдвиг,
- комбинация всех вышеперечисленных

Аффинные преобразования

Преобразования вида:

$$x' = a_{xx}x + a_{xy}y + a_{xz}z + b_x,$$

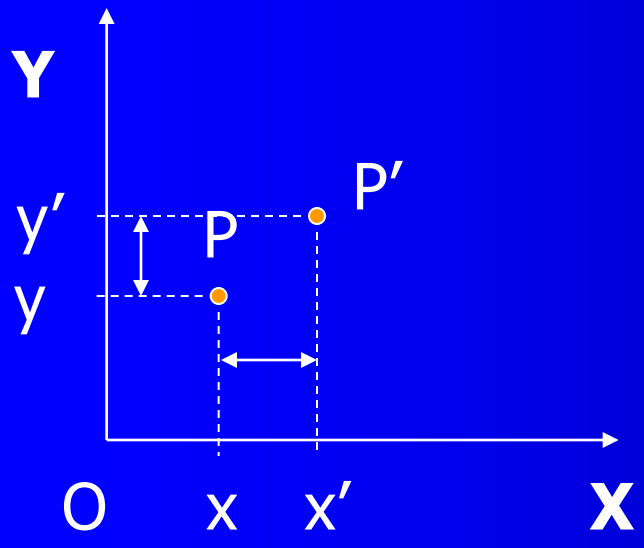
$$y' = a_{yx}x + a_{yy}y + a_{yz}z + b_y,$$

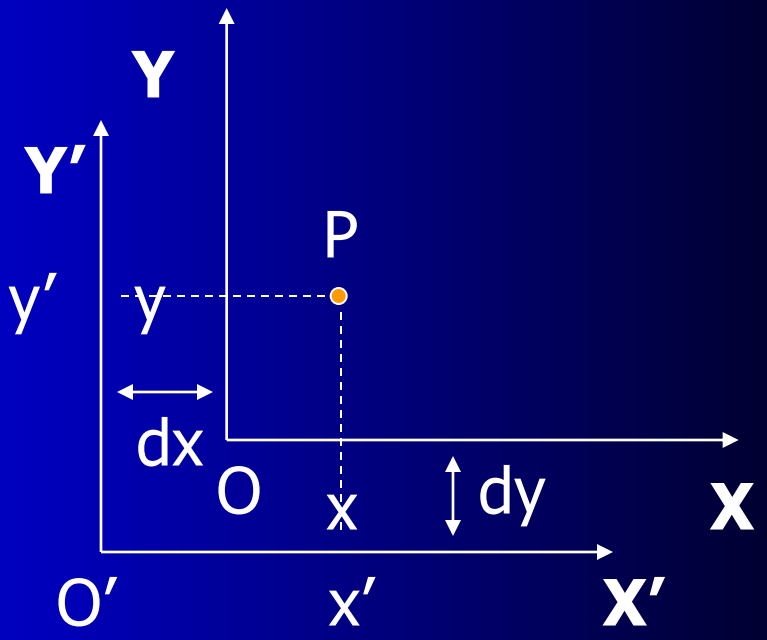
$$z' = a_{zx}x + a_{zy}y + a_{zz}z + b_z$$

называются Аффинными преобразованиями. Они обладают следующим свойством: параллельные линии преобразуются в параллельные линии. Все вышеперечисленные преобразования являются аффинными. Подмножество аффинных преобразований - перенос, поворот и отражение называются жёсткими преобразованиями, так как в процессе их выполнения сохраняются расстояния и углы между любыми двумя точками преобразуемой фигуры.

Перенос на плоскости

Перенос на плоскости – нахождение координат точки на плоскости после её переноса либо переноса координатных осей.





Перенос на плоскости

Обе операции эквивалентны и описываются одними и теми же выражениями:

$$\begin{cases} x' = x + dx \\ y' = y + dy \end{cases}$$

Т.е. данную систему можно трактовать двояко, как «точка P переместилась в P'» либо «начало координат O переместилось в O'».
Часто бывает проще перенести систему координат и провести соответствующие вычисления, чем проводить перенос одного объекта.

Поворот относительно начала системы координат

Требуется повернуть т. P с коорд. (x,y) относительно центра системы координат на угол φ, при этом т. P перейдёт в т. P′ с коорд. (x′,y′). Требуется найти новые координаты.

r – длина вектора от O до P и P′.

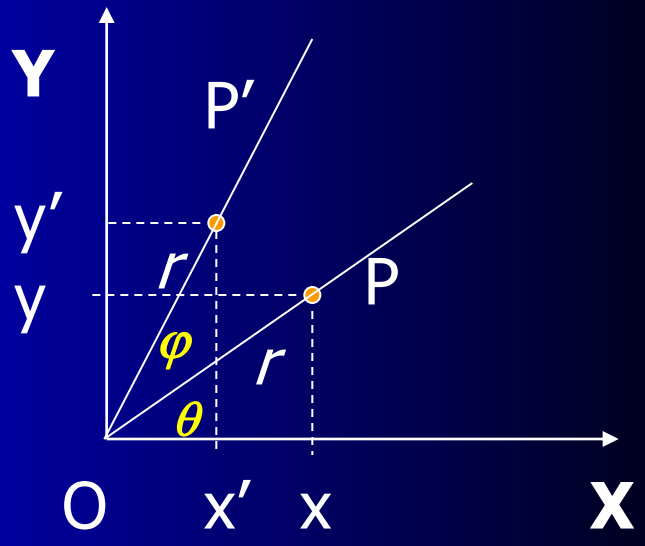
Справедлива следующая система:

$x = r\cos\theta$ $\quad x' = r\cos(\theta+\varphi)$

$y = r\sin\theta$ $\quad y' = r\sin(\theta+\varphi)$

Преобразуя *cos* суммы углов и выразив все через *r* получаем новую систему:

$$\begin{cases} x' = x\cos(\varphi) - y\sin(\varphi) \\ y' = x\sin(\varphi) + y\cos(\varphi) \end{cases}$$



Поворот относительно начала системы координат

Введём матрицу поворота $R(\varphi) = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}$

и вектора $P = \begin{bmatrix} x \\ y \end{bmatrix}$, $P' = \begin{bmatrix} x' \\ y' \end{bmatrix}$

тогда запись преобразования в матричном виде будет следующей :

$$\mathbf{P' = R(\varphi)P}$$

Поворот относительно произвольной точки

Поворот относительно произвольной точки (x_0,y_0) можно рассматривать как последовательность нескольких действий:

- сдвиг системы координат,
- поворот относительно нового центра координат
- возврат к старой системе координат.

Соответственно можно обойтись следующими изменениями:

$x \to x-x_0$, $y \to y-y_0$, $x' \to x'-x_0$, $y' \to y'-y_0$

$$\begin{cases} x' - x_0 = (x-x_0)\cos(\varphi) - (y-y_0)\sin(\varphi) \\ y' - y_0 = (x-x_0)\sin(\varphi) + (y-y_0)\cos(\varphi) \end{cases} \qquad P_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

$$\mathbf{P' = P_0 + R(\varphi)(P - P_0)}$$

Поворот растровых изображений

Для растровой графики используется та же матрица поворота. Однако если её применять без дополнительных операций, то будет наблюдаться следующий эффект:

а) б) в) г) д)

Пример поворота изображения без коррекции на произвольный угол: а) исходное изображение; б) повёрнутое на 3°; в)– на 6°; г)– на 10°; д)- на 14°

Поворот растровых изображений, алгоритм Оуэна и Македона

Оуэн и Македон предложили вариант разложения матрицы поворота:

$$R(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} =$$

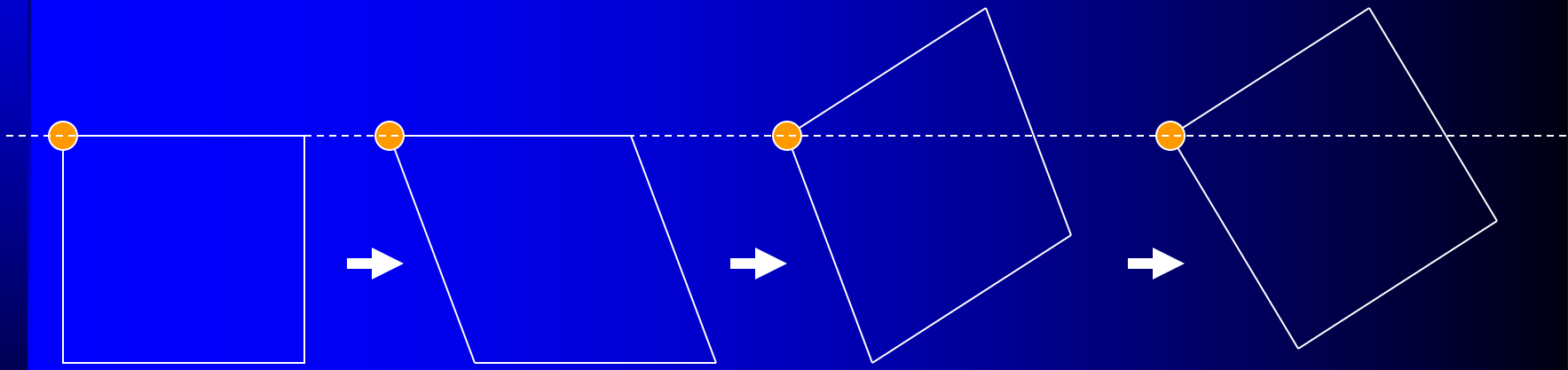
$$\begin{bmatrix} 1 & -tg\alpha/2 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ \sin\alpha & 1 \end{bmatrix} \times \begin{bmatrix} 1 & -tg\alpha/2 \\ 0 & 1 \end{bmatrix}$$

Осуществление поворота в три шага позволяет избежать указанных выше отрицательных эффектов без интерполяции «дырок».

Почему?

Поворот растровых изображений, алгоритм Оуэна и Македона

Суть алгоритма Оуэна и Македона заключается в повороте цифровых изображений путём трёх последовательных сдвигов строк, столбцов и затем снова строк изображения.



Масштабирование, отражение

2D Масштабирование: $x' = x \cdot s_x$, $y' = y \cdot s_y$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{P' = S \cdot P}$$

2D Отражение: $x' = (\ )1 \cdot x$, $y' = (\ )1 \cdot y$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} (\ )1 & 0 \\ 0 & (\ )1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Сдвиг

2D Сдвиг в направлении оси OX: $x' = x,\ y' = sh_x \cdot y$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Сдвиг в направлении оси OY: $x' = sh_y \cdot x,\ y' = y$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ sh_y & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Цепной код

Допустим мы имеем начальную точку линии, совпадающей с точкой растра. В случае 8-ми связности можем двигаться в 8-ми направлениях. Из этих восьми точек выбираем наиболее близкую к кривой и переходим к ней, затем производим перекодировку (т.е. для этой новой точки нумеруем соседей 0,1,2…7 – против часовой стрелки). Таким образом, записывая номера в сетке точек, к которым переходим, строим ход.

Целью является последовательность точек a=a,…,an , а полученная цепь имеет вид b=b1,…,bn .

3	2	1
4	x	0
5	6	7

Цепной код

Таким образом, представление контура цепным кодом состоит из абсолютного адреса начальной точки и последовательности ключевых слов («0» – «7»), указывающих направления отрезков прямой, соединяющей соседние пиксели.

При цепном кодировании возможны следующие операции:

- Измерение длины цепи
- Описание габаритного прямоугольника
- Изменение направления обхода
- Вычисление площади замкнутого контура
- Вычисления моментов инерции
- Вычисление кратчайшей цепи
- Вычисление расстояния между двумя точками
- Построение зеркальной цепи

Дифференциальный цепной код

В дифференциальном коде, соседи кодируются 0, ±1, ±2, ±3, 4, причем в целях экономии эти коды записывают в бинарном виде:

‘0’ – 0;
‘+1’ – 01;
‘-1’ – 011;
‘+2’ – 0111;
‘-2’ – 01111; и т.д.

3	2	1
4	x	0
-3	-2	-1

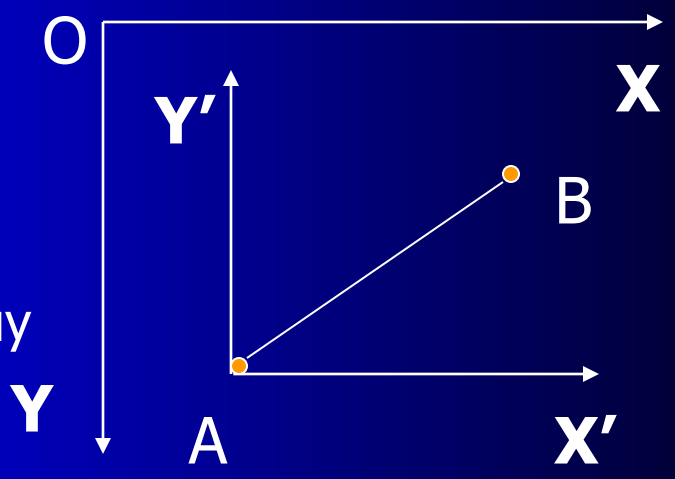
Возможны и другие варианты кодирования. «Соседи» номеруются также, но коды в 0 и 1 выглядят иначе:

‘0’ – 0;
‘+1’ – 100;
‘-1’ – 101;
‘+2’ – 1100;
‘-2’ – 1101;
‘+3’ – 1110;
‘-3’ – 1111, и т.д.

Алгоритмы изображения прямой линии

Вследствие растровой природы монитора при построении линии необходимо решать задачу выбора из матрицы составляющих экран пикселей лишь тех, которые необходимы для построения прямой (отрезка). Данный процесс называется «разложением линии в растр».

Допустим требуется разложить в растр отрезок АВ. Для начала с помощью операций:
переноса системы координат,
отражения отрезка
сводим операцию к каноническому случаю построения отрезка в первом октанте.

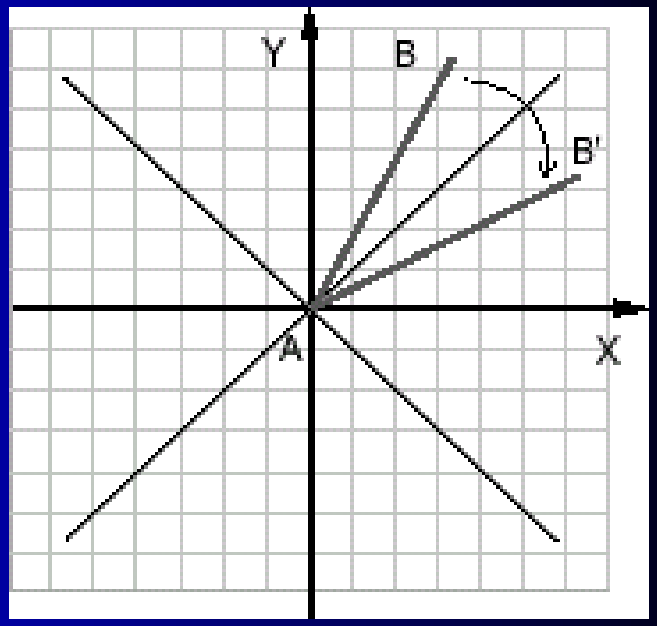


Алгоритмы изображения прямой линии

Отрезок может лежать в любом из 8 октантов, но всегда существуют симметрии относительно осей, разделяющих эти октанты (симметрии определяются матрицами:

$$\begin{pmatrix} \pm 1 & 0 \\ 0 & \pm 1 \end{pmatrix} \quad \text{и} \quad \begin{pmatrix} 0 & \pm 1 \\ \pm 1 & 0 \end{pmatrix}$$

позволяющие свести задачу к случаю отрезка, лежащего в первом октанте. Отрезок AB можно привести к каноническому случаю AB' преобразованием вида:

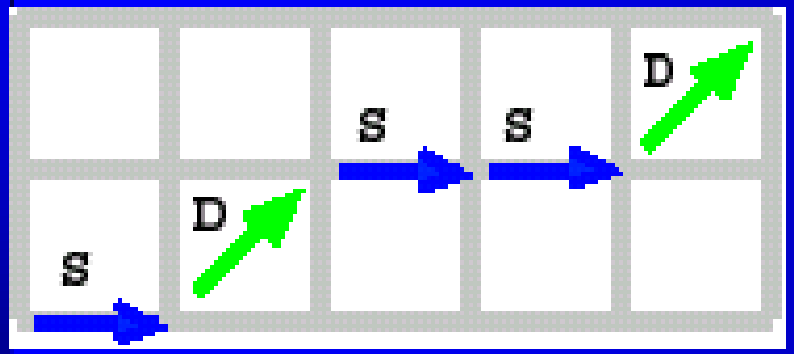


$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

и точка B (a,b) перейдет в B' (b,a)

Алгоритм цифрового дифференциального анализатора

В каноническом случае процесс рисования 8-ми связной прямой линии можно закодировать последовательностью следующего вида: sdssd…, где:



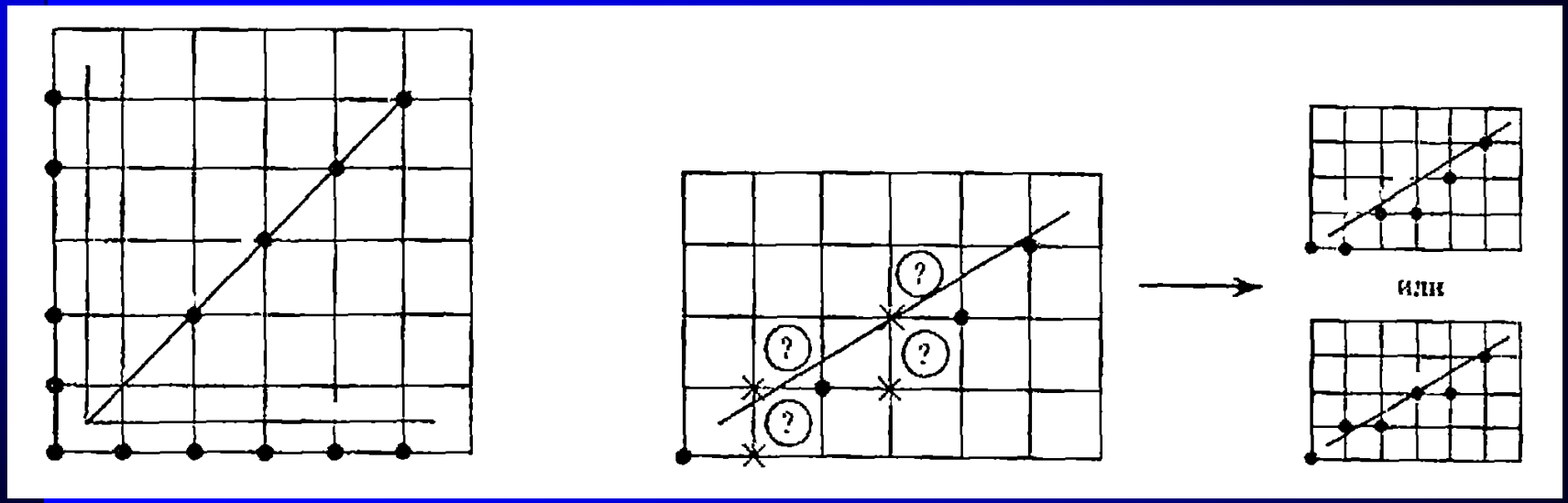
s - горизонтальное смещение

d - диагональное смещение

Алгоритм цифрового дифференциального анализатора

Основной вопрос при разложении отрезка в растр:

Какой следующий пиксель выбирать: горизонтальный или диагональный (в случае 4-х связности – вертикальный)?



Алгоритм цифрового дифференциального анализатора

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

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

Следовательно, у на каждом следующем шаге должно быть:

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

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

где x_1 y_1, и x_2, y_2 — концы разлагаемого отрезка и y_i — начальное значение для очередного шага вдоль отрезка. Фактически это уравнение представляет собой рекуррентное соотношение для последовательного вычисления значений у вдоль нужного отрезка.

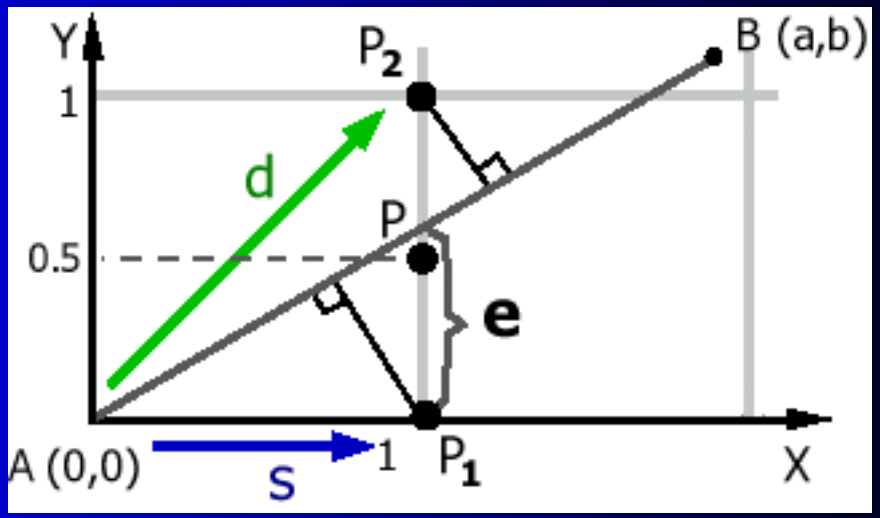
Алгоритм цифрового дифференциального анализатора

Имеем: $e_0 = < 1$; $\Delta e = < 1$

В начальный момент :

$P_1=(1,0)$; $P_2=(1,1)$

$x=0$; $y=0$; $e=e_0$;

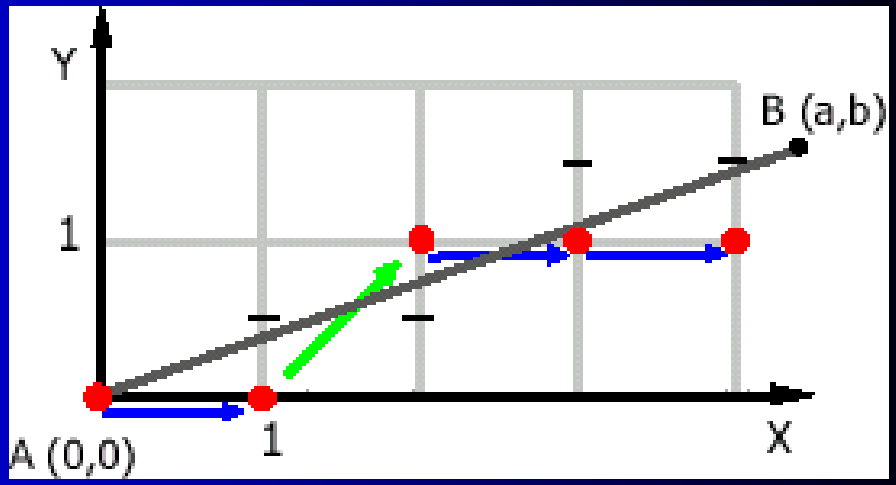


В силу подобия треугольников, для того чтобы определить, какая из точек (P1 или P2) ближе, достаточно сравнить расстояния от точки пересечения P до P1 и P2, что равносильно сравнению e с 1/2.

Алгоритм цифрового дифференциального анализатора

```
while(x ≤ a)
{
    plot(x,y);

    if (e>=1/2)
     {  // d : диагональное
        //      смещение
           x++; y++;
           e+=Δe-1 ;
       // коррекция ошибки т.к. произошло смещение по y на 1 вверх
      }
     else
      { // s : горизонтальное смещение
           x++;
           e+=Δe;
       }
}
```



Недостаток алгоритма:

Работает с числами с плавающей точкой.