Министерство образования Республики Беларусь

Учреждение Образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра Интеллектуальных информационных технологий

Лабораторная работа № 1

«Алгоритмы применения отрезков»

Выполнил: Назаров Х.М

ст. гр. 121731

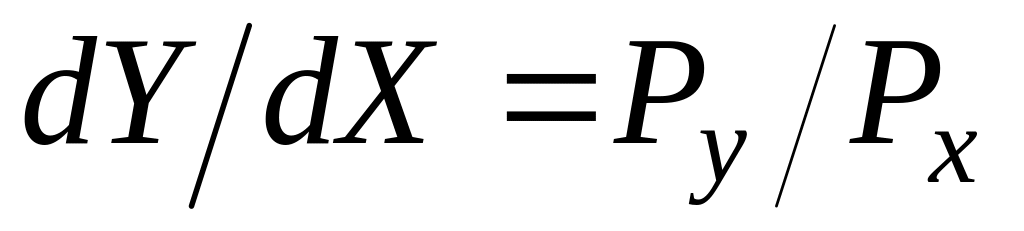
Проверил: Сальников Д. А

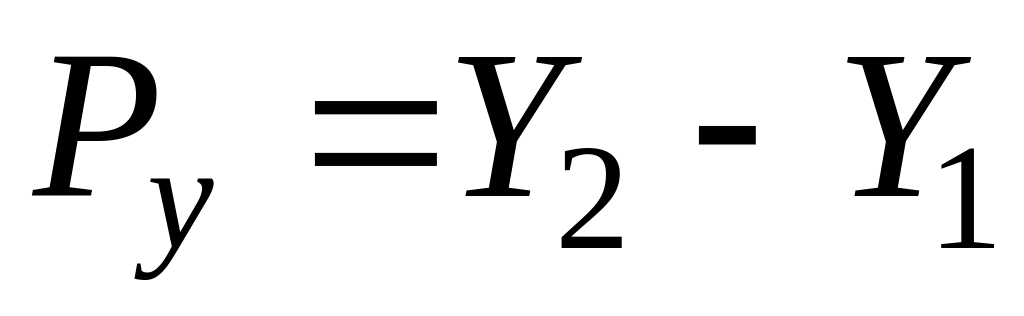
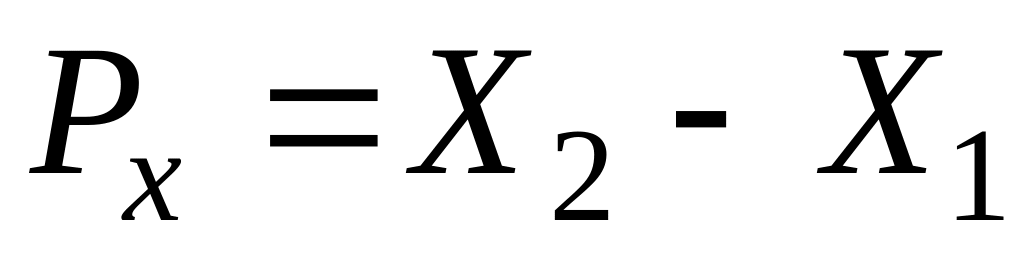
Минск 2024

**Цель:** Разработать элементарный графический редактор, реализующий построение отрезков с помощью алгоритма ЦДА, целочисленного алгоритма Брезенхема и алгоритма Ву. Вызов способа генерации отрезка задается из пункта меню и доступно через панель инструментов «Отрезки». В редакторе кроме режима генерации отрезков в пользовательском окне должен быть предусмотрен отладочный режим, где отображается пошаговое решение на дискретной сетке.

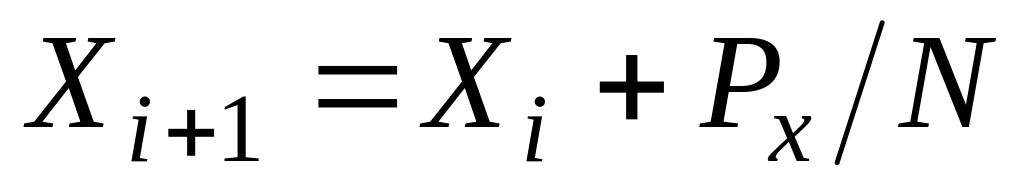
Алгоритм ЦДА.

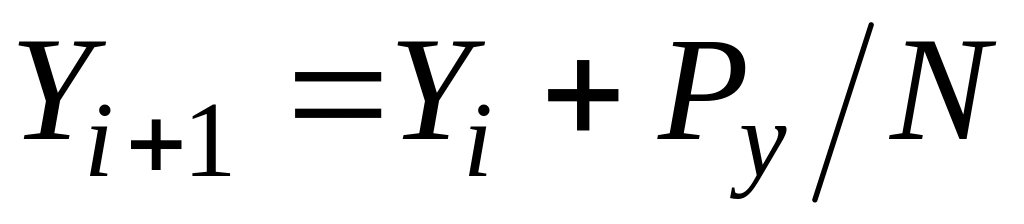
 Спомощью алгоритма ЦДА решается дифференциальное уравнение отрезка, имеющее вид:

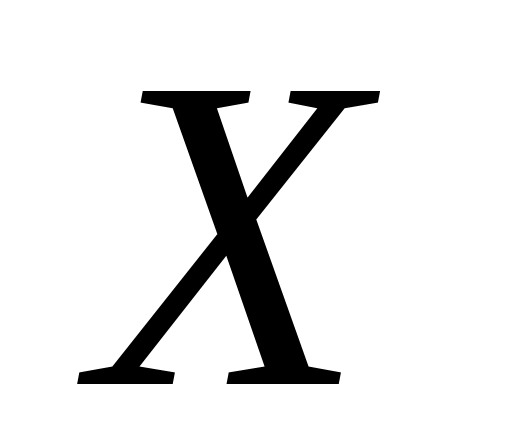
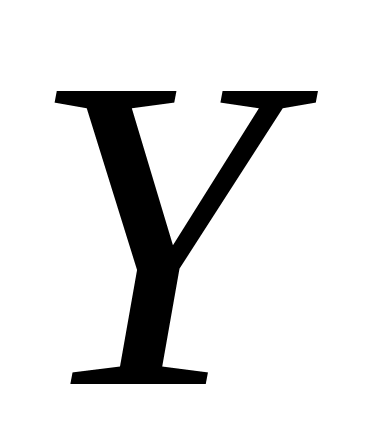
 ,

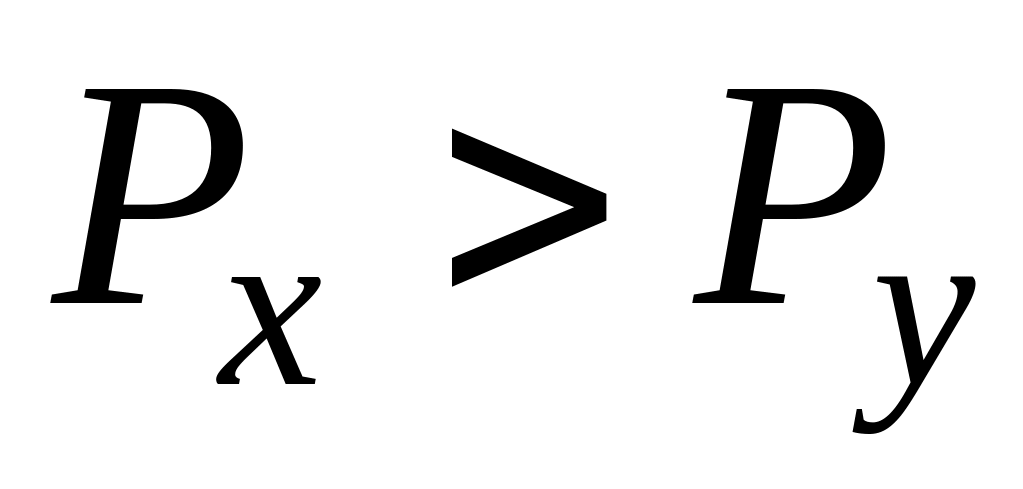
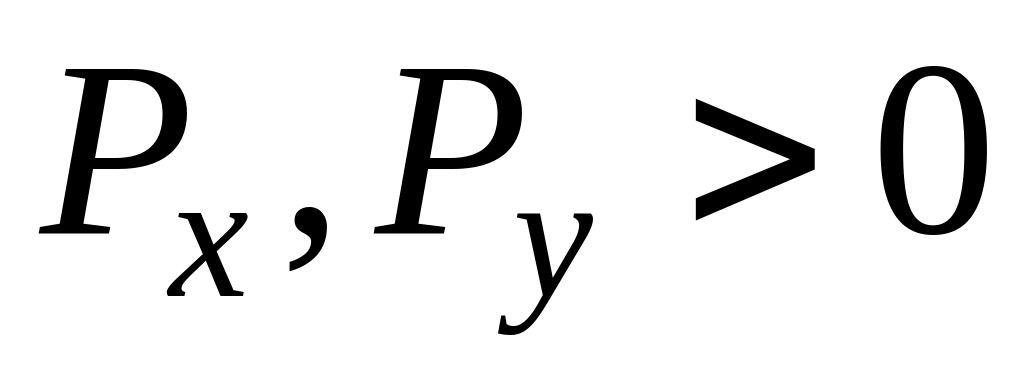
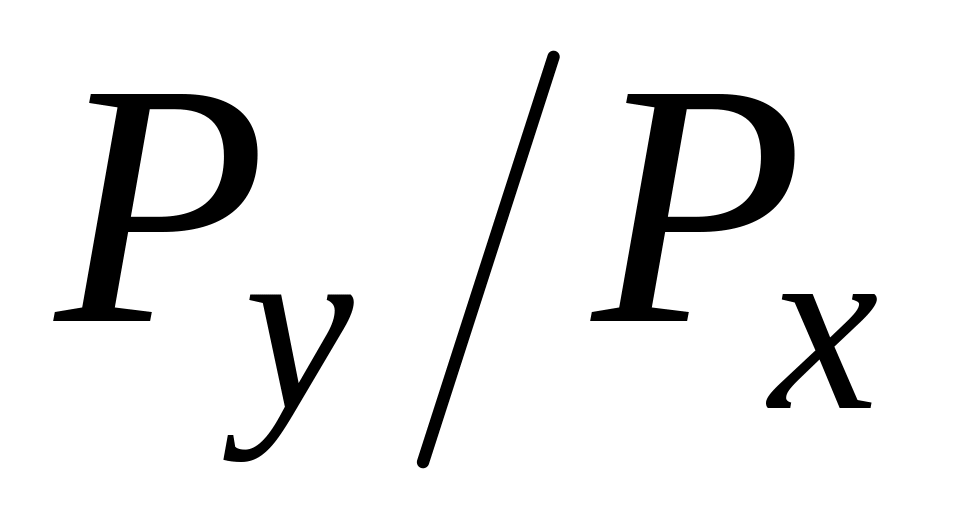
где  *–*приращение координат отрезка по оси *Y*;  – приращение координат отрезка по оси *X*; *X*2, *Y*2, *X*1,*Y*1 – начальная и конечная координаты отрезка.

При этом ЦДА формирует дискретную аппроксимацию непрерывного решения этого дифференциального уравнения. В симметричном ЦДА тем или иным образом определяется количество узлов *N*, используемых для аппроксимации отрезка. Затем за *N* циклов вычисляются координаты очередных узлов:

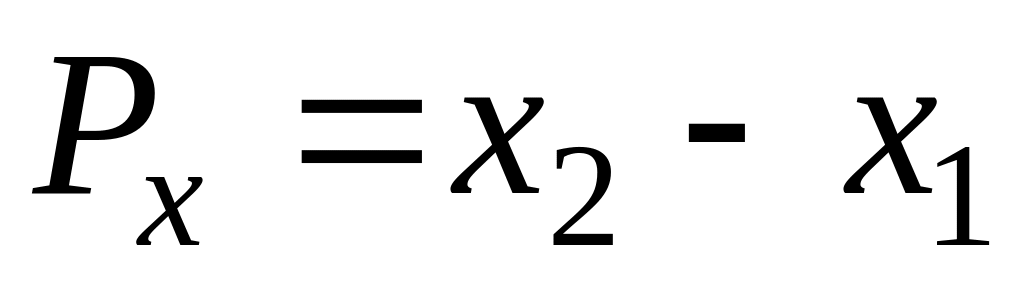
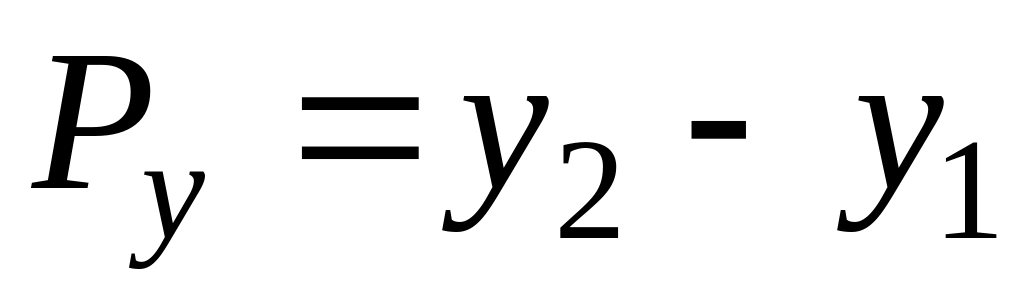
 ,

 .

Получаемые значения  и  преобразуются в целочисленные значения координаты очередного подсвечиваемого пиксела. Генератор векторов, использующий этот алгоритм, имеет тот недостаток, что точки могут прописываться неоднократно, что увеличивает время построения, или отрезок будет выглядеть прерывистым.

Кроме того, из-за независимогравлений и построенные отрезки као вычисления обеих координат нет предпочтительных напжутся не очень красивыми. Субъективно лучше смотрятся вектора с единичным шагом по большей относительной координате (несимметричный ЦДА). Для  (при  ) это означает, что координата по *X* направлению должна увеличиться на 1 *Px* раз, а координата по *Y* – направлению должна также *Px*раз увеличиться, но на  . Т.е. количество узлов аппроксимации берется равным числу пикселов вдоль наибольшего приращения. Для генерации отрезка из точки (*X*1*,Y*1) в точку (*X*2*,Y*2) в первом октанте алгоритм несимметричного ЦДА имеет вид:

*x – integer*;

*Вычислить приращения координат: *;**;

*Занести начальную точку отрезка PutPixel*(*x*1*,y*1);

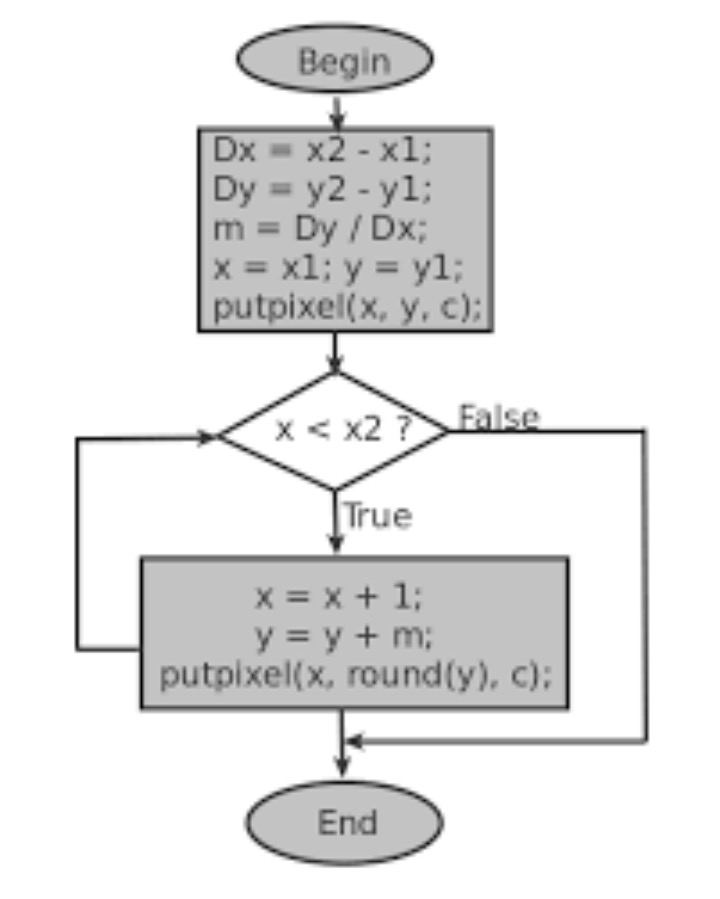
*Сгенерировать отрезок while*(*x*1*< x*2) {*x*1*:= x*1*+*1;*y*1*:= y*1*+ Py/Px*;*PutPixel*(*intl*(x1),*intl*(*y*1));}

*Примечание*: intl – приближение к нижнему, т.е. 5.3 = 5; 5.6 = 5; –5.3 = –6; –5.6 = –6.

Пример генерации отрезка по алгоритму несимметричного ЦДА приведен на рис. 1.2.

IMG_268

*Рис. 1.2.* Генерация отрезка несимметричным ЦДА



Код программы Алгоритма ЦДА.

    def dda\_line(self):

        dx = self.end['x'] - self.start['x']

        dy = self.end['y'] - self.start['y']

        steps = max(abs(dx), abs(dy))

        x\_increment = dx / steps

        y\_increment = dy / steps

        x, y = self.start['x'], self.start['y']

        for \_ in range(steps + 1):

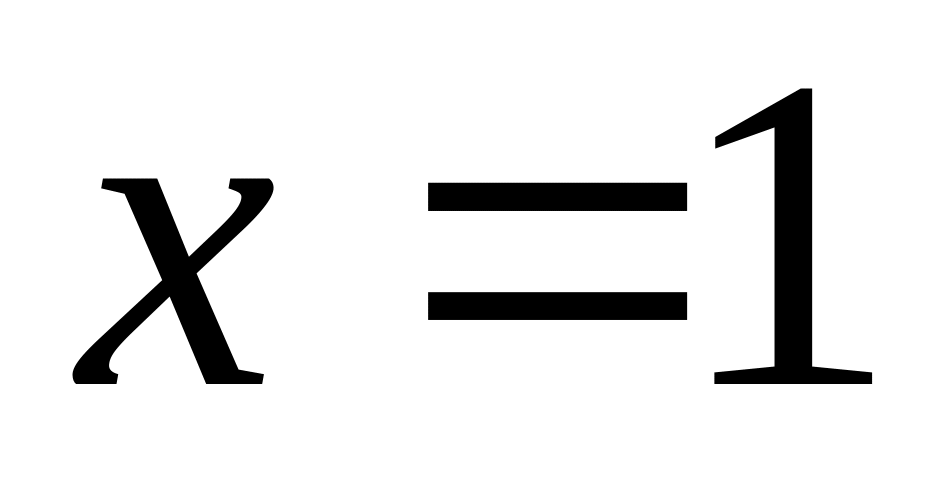
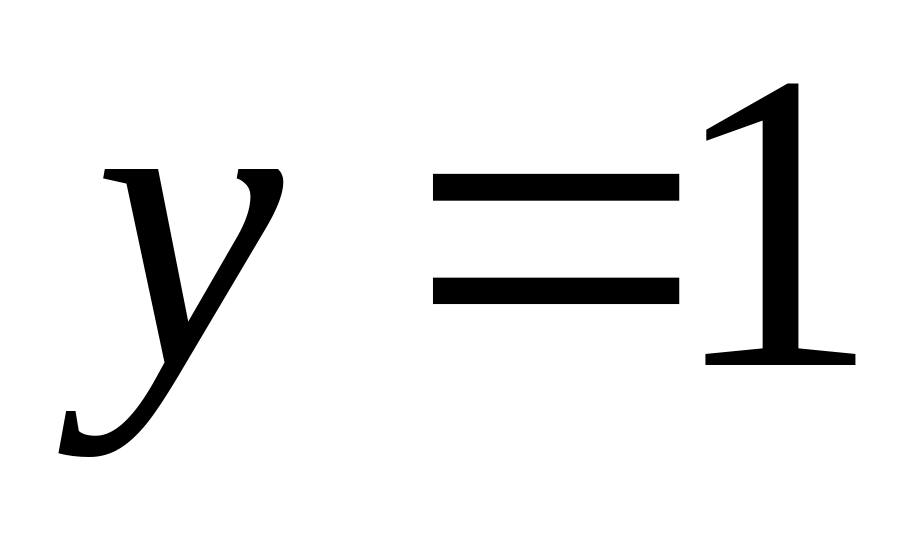
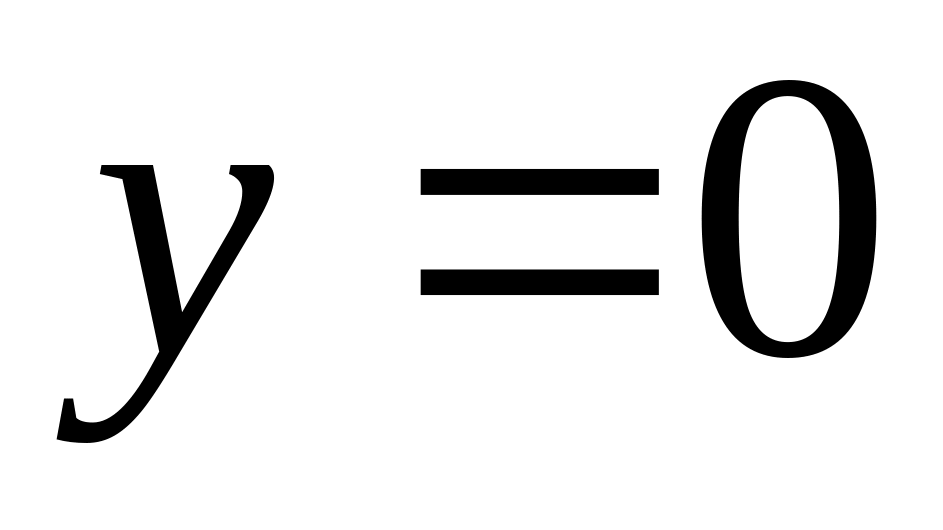
            self.plot\_pixel(round(x), round(y), 0)

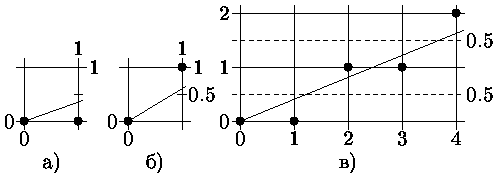
            x += x\_increment

            y += y\_increment

## Алгоритм Брезенхема

Так как приращения координат, как правило, не являются целой степенью двойки, то в ЦДА – алгоритме (см. предыдущий пункт) требуется выполнение деления, что не всегда желательно. 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 (либо на нуль, либо на единицу) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Такое расстояние назовем *ошибкой*.

Брезенхем предложил алгоритм, построенный так, что требуется проверять лишь знак этой *ошибки*. На рис. 1.3 а,б) это иллюстрируется для отрезка в первом октанте, т.е. для отрезка с угловым коэффициентом, лежащим в диапазоне от нуля до единицы. Из рисунка можно заметить, что если угловой коэффициент отрезка из точки (0,0) больше чем 1/2, то его пересечение с прямой  будет расположено ближе к прямой  , чем к прямой  . Следовательно, точка растра (1,1) лучше апроксимирует 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). Не все отрезки проходят через точки растра (рис. 1.3 в). Здесь отрезок с тангенсом угла наклона 3/8 сначала проходит через точку растра (0,0) и последовательно пересекает три пиксела. Так как желательно проверять только знак ошибки, то она первоначально устанавливается равной – 1/2.

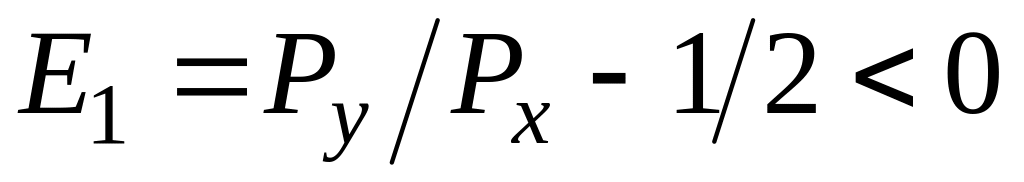


*Рис. 1.3*. Алгоритм Брезенхема генерации отрезков

Если *Е* < 0 (где *Е* – ошибка), то *Y*-координата не меняется по сравнению с предыдущей точкой. В противном случае *Y* увеличивается на 1.

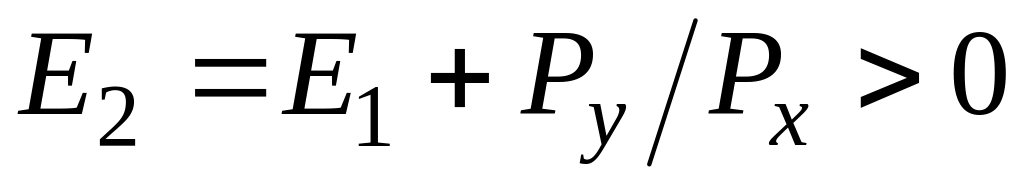
Для вычисления *Е* положим, что рассматриваемый вектор начинается в точке (0,0) и проходит через точку (4, 1.6) (рис. 1.3 в), т.е. имеет положительный наклон, меньший 1.

Из рис. 1.3 в) видно, отклонение для первого шага:

 ,

поэтому для занесения пиксела выбирается точка (1,0).

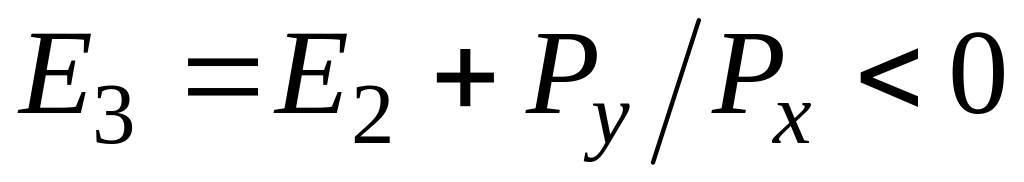
Отклонение для второго шага вычисляется добавлением приращения *Y*-координаты для следующей *X*-позиции (рис. 1.3 в):

 ,

поэтому для занесения пиксела выбирается точка (2,1). Так как отклонение считается от *Y*-координаты, которая теперь увеличилась на 1, то из накопленного отклонения для вычисления последующих отклонений надо вычесть 1:

*E*2 = *E*2 – 1

Отклонение для третьего шага:

 ,

поэтому для занесения пиксела выбирается точка (3,1).

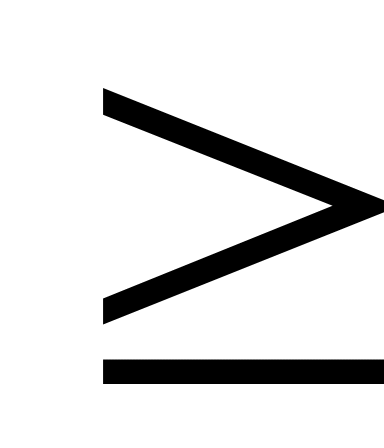
Возможны случаи:

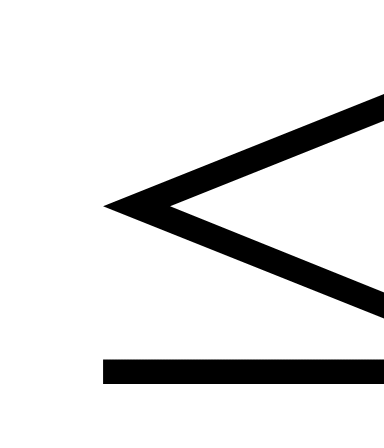
|  |  |
| --- | --- |
| *E*1 > 0 | *E*1 <= 0 |
| ближайшая точка есть: | |
| *X*1*= X*0*+*1;*Y*1*= Y*0+1 | X1 = X0 + 1; Y1 = Y0 |
| *E*2 = *E*1 + *Py*/*Px*– 1 | *E*2 = *E*1 + *Py*/*Px* |

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

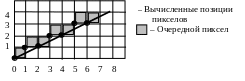
|  |  |
| --- | --- |
|  | *E*1 *=*2*\* Py – Px* |
| *E*1 > 0 | *E*2 *= E*1 *+*2 (*Py – Px*) |
| *E*1 <= 0 | *E*2 *= E*1 *+*2*\* Py* |

Таким образом получается алгоритм, в котором используются только целые числа, сложение (умножение на 2 – сдвиг влево или сложение с самими числом):

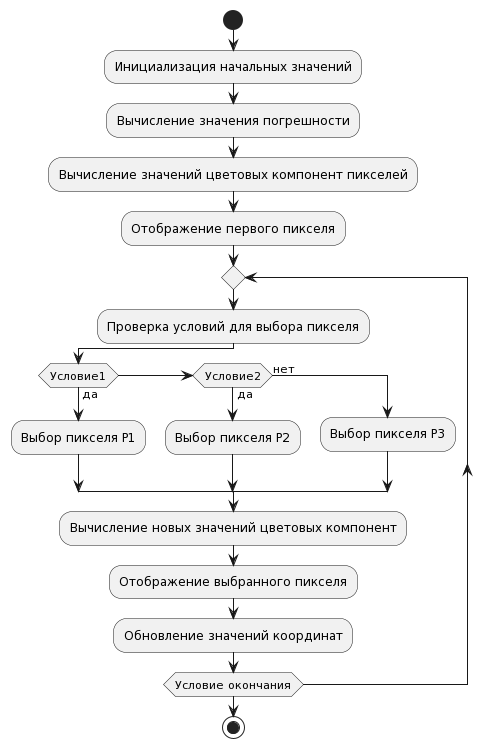
*X = x*1;*Y = y*1;*Px = x*2*– x*1;*Py = y*2*– y*1;*E =*2*\*Py – Px*;*i = Px*;*PutPixel*(*X, Y*);*/\* Первая точка вектора \*/ while*(*i= i –*1 *0*){*if*(*E **0*){*X= X +*1;*Y= Y +*1;*E= E +*2*\**(*Py – Px*);}*else X= X +*1;*E= E +*2*\*Py*;*PutPixel*(*X, Y*);*/\* Очередная точка вектора \*/*}

Этот алгоритм пригоден для случая *0 **dY dX.* Для других случаев алгоритм строится аналогичным образом.

На рис. 1.4 приведен пример генерации по алгоритму Брезенхема того же самого отрезка, что и показанного на рис. 1.2 для генерации по алгоритму несимметричного ЦДА.



*Рис. 1.4.* Генерация отрезка по алгоритму Брезенхема



Код программы алгоритма Брезентхема

    def bresenham\_line(self):

        x0, y0, x1, y1 = self.start['x'], self.start['y'], self.end['x'], self.end['y']

        dx = abs(x1 - x0)

        dy = abs(y1 - y0)

        sx = 1 if x0 < x1 else -1

        sy = 1 if y0 < y1 else -1

        err = dx - dy

        while (x0, y0) != (x1, y1):

            self.plot\_pixel(x0, y0, 0)

            e2 = 2 \* err

            if e2 > -dy:

                err -= dy

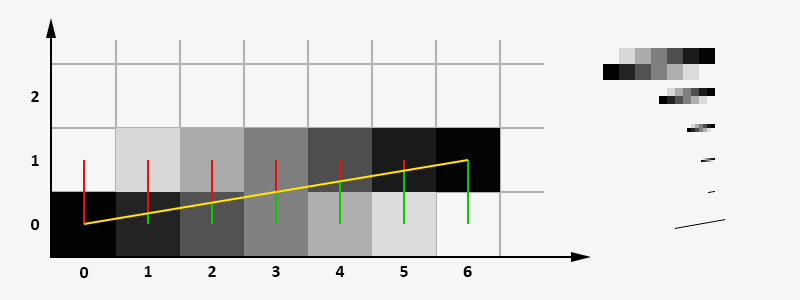
                x0 += sx

            if e2 < dx:

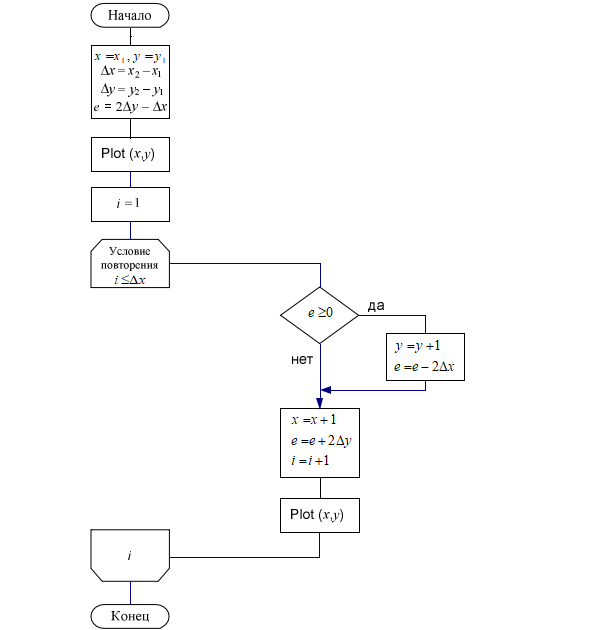
                err += dx

                y0 += sy

Алгоритм Ву.

Теперь про алгоритм У Сяолиня для рисования сглаженных линий. Он отличается тем, что на каждом шаге ведётся расчёт для двух ближайших к прямой пикселей, и они закрашиваются с разной интенсивностью, в зависимости от удаленности. Точное пересечение середины пикселя даёт 100% интенсивности, если пиксель находится на расстоянии в 0.9 пикселя, то интенсивность будет 10%. Иными словами, сто процентов интенсивности делится между пикселями, которые ограничивают векторную линию с двух сторон.  
  
**  
  
На картинке выше красным и зелёным цветом показаны расстояния до двух соседних пикселей.  
  
Для расчёта ошибки можно использовать переменную с плавающей запятой и брать значение

ошибки из дробной части.



Код программы алгоритма Ву.

    def wu\_line(self):

        def plot(x, y, intensity):

            alpha = min(1, max(0, 1 - intensity))

            self.plot\_pixel(x, y, alpha)

        x0, y0, x1, y1 = self.start['x'], self.start['y'], self.end['x'], self.end['y']

        dx = x1 - x0

        dy = y1 - y0

        if abs(dx) > abs(dy):

            if x1 < x0:

                x0, x1, y0, y1 = x1, x0, y1, y0

            gradient = dy / dx

            y = y0 + gradient \* 0.5

            while x0 <= x1:

                plot(x0, int(y), 1 - (y - int(y)))

                plot(x0, int(y) + 1, y - int(y))

                y += gradient

                x0 += 1

        else:

            if y1 < y0:

                x0, x1, y0, y1 = x1, x0, y1, y0

            gradient = dx / dy

            x = x0 + gradient \* 0.5

            while y0 <= y1:

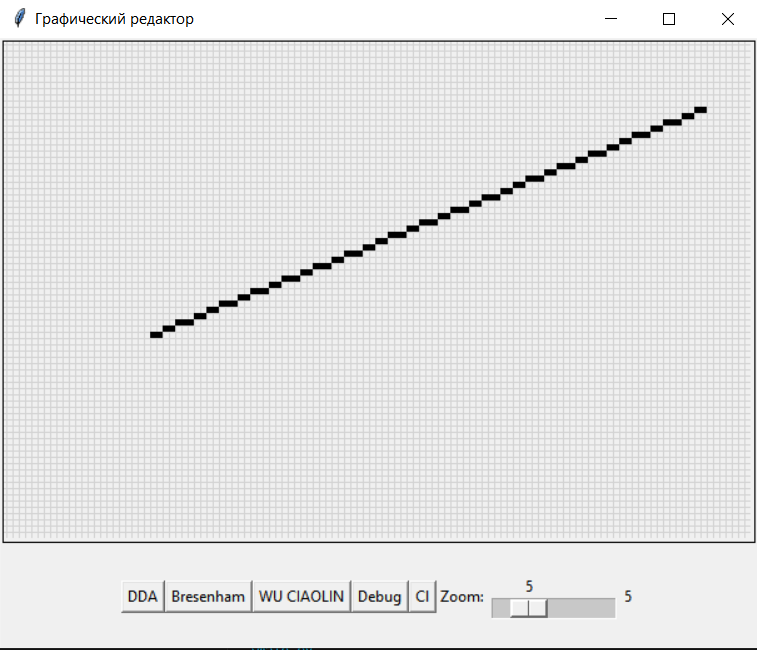
                plot(int(x), y0, 1 - (x - int(x)))

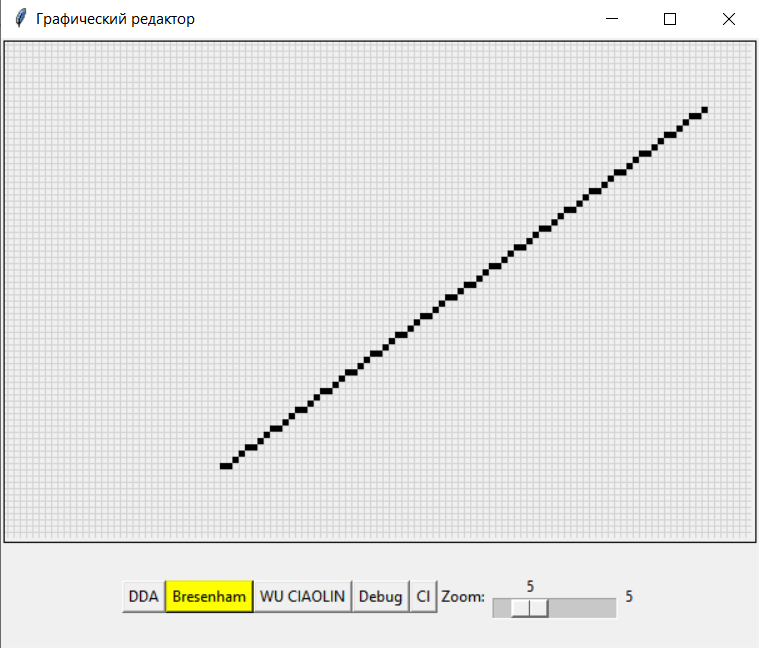
                plot(int(x) + 1, y0, x - int(x))

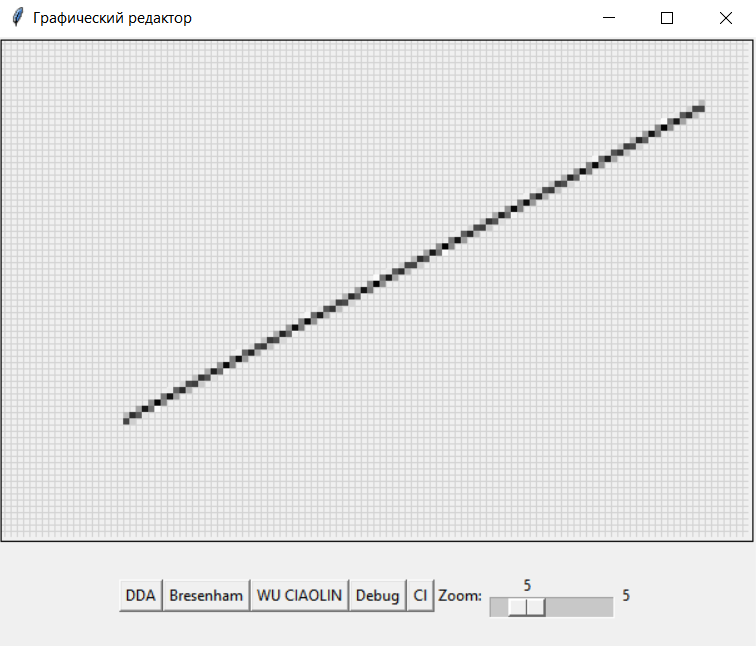
                x += gradient

                y0 += 1

Интерфейс.







**Вывод.**

В данной работе была рассмотрена разница между двумя алгоритмами рисования линий - ЦДА (Цифрового дифференциального анализатора) и Брезенхема.

Основное отличие между этими алгоритмами заключается в способе вычисления и отображения каждого пикселя линии. Алгоритм ЦДА вычисляет шаг между пикселями, разделяя разницу координат на количество пикселей, а затем округляет каждый пиксель до ближайшего целого значения. В то время как алгоритм Брезенхема вычисляет шаг на основе ошибки округления и также округляет пиксели, но с учетом этой ошибки.

Важным аспектом, который следует отметить, является то, что алгоритм ЦДА прост в реализации, но менее точен и более чувствителен к накоплению ошибок округления при отрисовке длинных линий. Алгоритм Брезенхема более точен и устойчив к ошибкам округления, поэтому он предпочтительнее при рисовании линий с постоянным наклоном. Он также более эффективен в своей реализации.

Итак, выбор между алгоритмами ЦДА и Брезенхема зависит от требований и ограничений системы. Если важна простота реализации и точность не является первоочередной задачей, алгоритм ЦДА может быть хорошим выбором. Однако, если требуется высокая точность и устойчивость к ошибкам округления, алгоритм Брезенхема будет предпочтительнее.

В целом, понимание различий между алгоритмами ЦДА и Брезенхема поможет разработчикам искусственного интеллекта и компьютерной графики выбрать наиболее подходящий алгоритм для конкретной задачи рисования линий.

Литература:

1. [1.1. Цифровой дифференциальный анализатор (цда) (studfile.net)](https://studfile.net/preview/7871522/page:2/)
2. .[Брезенхем и У на страже диагоналей / Хабр (habr.com)](https://habr.com/ru/articles/185086/)
3. https://poe.com/chat/