



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Дисциплина « Компьютерная графика»

Лабораторная работа №3

по теме:

«Основы растровой графики. Алгоритмы разложения отрезка в растр.

Реализация и исследование алгоритмов построения отрезков.»

Работу выполнил:

студент группы ИУ7-43Б

Сукочева А.

Работу проверил:

Куров А. В.

2020 г.

Цель работы:

Изучить и реализовать алгоритмы разложения отрезка в растр.

Проанализировать и сравнить визуальные и временные характеристики.

Задание:

1. Построение одиночных отрезков.
2. Сравнение визуальных характеристик отрезков, построенных разными алгоритмами.
3. Исследование временных характеристик. (Результат - Гистограммой.)
4. Построение пучка прямых.

Для рисования отрезка необходимо реализовать:

1. Выбор алгоритма из списка.
2. Выбор цвета.
3. Задание координат начала и конца отрезка.
4. Рисование фоновым цветом.

Теоретический материал:

Экран растрового дисплея можно рассматривать как матрицу пикселей, каждый из которых можно высветить независимо друг от друга. В силу этого нельзя провести отрезок непосредственно из одной точки в другую точку. Поэтому необходимы специальные алгоритмы, позволяющие выбирать пиксели, аппроксимирующие отрезок. Такой процесс определения пикселей, наилучшим образом аппроксимирующих заданный отрезок, называется разложением отрезка в растр.

Требования:

1. Быстрота алгоритма.
2. Яркость и интенсивность не должны зависеть от длины, угла, наклона и должны быть постоянными на протяжении всего отрезка.
3. Отрезок должен быть определенной длины.
4. Отрезок должен начинаться и заканчиваться в заданных точках.
5. Не выполнять лишние вычисления.

Алгоритмы:

1. Алгоритм цифрового дифференциального анализатора(ЦДА).
2. Алгоритмы Брезенхема:
 1. С дробными числами.
 2. С целыми числами.
 3. Со сглаживанием.
3. Алгоритм Ву.
4. Библиотечный алгоритм.

Алгоритм цифрового дифференциального анализатора(ЦДА):

```
def differential_analyzer(start, stop):
    dx = stop[0] - start[0]
    dy = stop[1] - start[1]

    if fabs(dx) - fabs(dy) >= 0:
        l = fabs(dx)
    else:
        l = fabs(dy)

    dx, dy = dx / l, dy / l
    x, y = start[0], start[1]

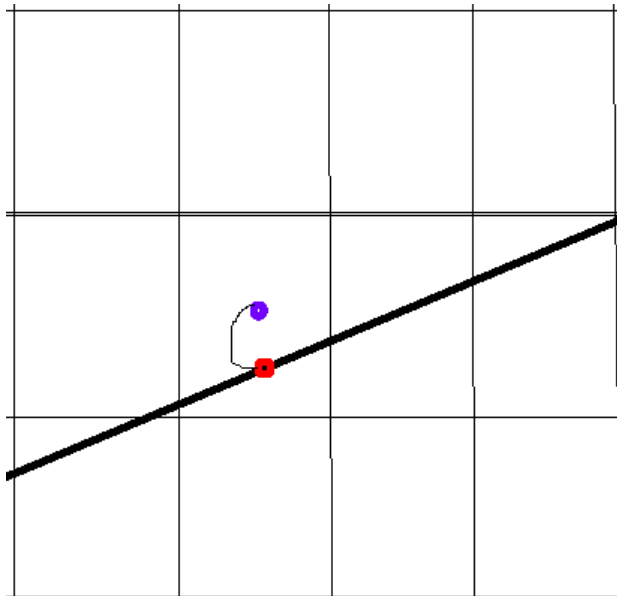
    for _ in range(int(l + 1)):
        print_pixel(x, y)
        x += dx
        y += dy
```

Для начала нам нужно узнать приращение концов отрезка. За это отвечает dx и dy . Приращение по модулю равное единице выбирается для той оси, для которой приращение концов отрезка наибольшее (По модулю). Для другой оси выбирается приращение < 1 . Суть состоит в том, что по одной оси мы стабильно идем с шагом по модулю равному 1, а по другой оси мы идем с таким шагом, который позволит нам дойти до нужной точки без пробелов (пустых, не закрашенных пикселей). Алгоритм становится медленным из-за того, что нам нужно округлять значение и мы считаем float для одного приращения. (Для второго приращения будет int потому что мы всегда прибавляем единицу по модулю).

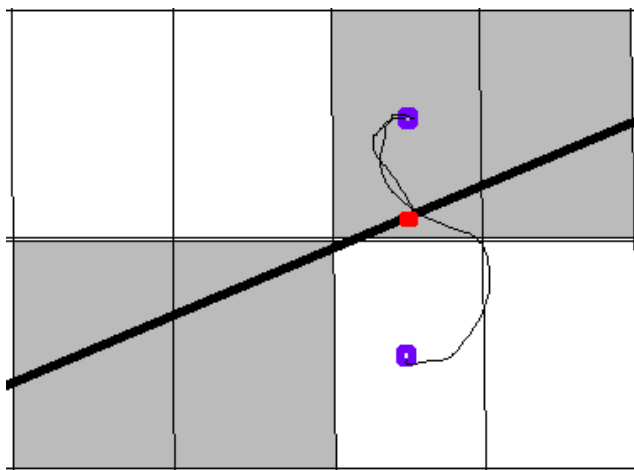
Алгоритм Брезенхема с дробными числами:

```
def bresenham_float(start, stop):  
    dx = stop[0] - start[0]  
    dy = stop[1] - start[1]  
    x, y = start[0], start[1]  
    sx, sy = sign(dx), sign(dy)  
    dx = fabs(dx)  
    dy = fabs(dy)  
  
    swap = 0  
    if dy > dx:  
        swap = 1  
        dx, dy = dy, dx  
    m = dy / dx  
    e = m - 0.5  
  
    for _ in range(int(dx + 1)):  
        print_pixel(x, y)  
        if e >= 0:  
            if swap == 0:  
                y += sy  
            else:  
                x += sx  
            e -= 1  
  
        if swap == 0:  
            x += sx  
        else:  
            y += sy  
        e += m
```

Брезенхем ввел понятие ошибки. Ошибкой является расстояние между действительным положением отрезка (Красный кружок) и ближайшем пикселем (Синий кружок). Значение ошибки было в пределах -0.5 до 0.5 (т.к. пиксель равен 1, если линия проходит ниже пикселя (как на рисунке) то значение ошибки отрицательное, иначе (если выше) то положительное).



Каждый раз, когда значение ошибки выходило за эти пределы нам нужно было увеличивать одну из координат (Вторая координата стабильно увеличивалась на единицу за каждый цикл), а значение ошибки скорректировать. Корректировка заключалась в том, что мы из ошибки вычитали единицу. (Т.е. на нашем примере (рисунок выше)



видно, что ошибка вышла за предел равный 0.5 (Она приблизительно равна 0.6) Тогда мы понимаем, что нам нужно закрасить верхний пиксель и для этого из ошибки вычитаем 1, получается (приблизительно) -0.4 , как мы видим, это новое значение ошибки. Далее было предложено анализировать не само значение ошибки, а ее знак. Т.е. ошибку нужно было сместить на -0.5 и тогда она была в пределах от -1 до 0 . За каждый цикл вычислялась ошибка, анализировалось ее значение, и выполнялись вышеизложенные действия. Ускорить данный алгоритм можно, путем избавления от вещественных чисел. (Что будет показано далее)

Алгоритм Брезенхема с целыми числами:

```
def bresenham_int(start, stop):
    dx = stop[0] - start[0]
    dy = stop[1] - start[1]
    x, y = start[0], start[1]
    sx, sy = sign(dx), sign(dy)
    dx = fabs(dx)
    dy = fabs(dy)

    swap = 0

    if dy > dx:
        swap = 1
        dx, dy = dy, dx

    e = 2 * dy - dx

    for _ in range(int(dx + 1)):
        print_pixel(x, y)

        if e >= 0:
            if swap == 0:
                y += sy
            else:
                x += sx
            e -= (2 * dx)

        if swap == 0:
            x += sx
        else:
            y += sy
        e += (2 * dy)
```

Для повышения скорости было предложено работать с целыми числами, для этого ошибка вычислялась как целочисленная переменная:

$e = \frac{dy}{dx} - 0.5$ (Домножим на $2dx$) $\rightarrow 2dx e = 2dy - dx$ (Целочисленное значение ошибки). $e = 2dy - dx$, тогда в теле цикла мы должны домножить на $2dx$. $e = e - 1 \rightarrow e = e - 2dx$; $e = e + 1 \rightarrow e = e + 2dy$.

Алгоритм Брезенхема со сглаживанием:

```
def bresenham_smooth(start, stop):
    dx = stop[0] - start[0]
    dy = stop[1] - start[1]
    x, y = start[0], start[1]
    sx, sy = sign(dx), sign(dy)
    dx = fabs(dx)
    dy = fabs(dy)

    swap = 0
    if dy > dx:
        swap = 1
        dx, dy = dy, dx
    m = dy / dx
    e = 0.5

    for _ in range(int(dx + 1)):
        print_pixel_color(x, y, 1 - e)

        if e >= 1:
            if swap == 0:
                y += sy
            else:
                x += sx
            e -= 1
        if swap == 0:
            x += sx
        else:
            y += sy
        e += m
```

Суть состоит в том, чтобы интенсивность пикселя выбиралась пропорционально площади части пикселя, находящейся под отрезком. Т.к. пиксель имеет стороны равные 1, то прямая, проходящая через пиксель поделит его на 2 части. Площадь прямой будет в пределах от 0 до 1, это значение и будет считаться процентом интенсивностью. Начальная площадь равна 0.5, т.к. прямая проходит через центр квадрата. Далее можно связать ошибку (о которой говорилось ранее) с площадью, и рассматривать ее как мы делали это ранее.

Алгоритм ВУ:

```
def vu(start, stop):
    dx = stop[0] - start[0]
    dy = stop[1] - start[1]
    x, y = start[0], start[1]
    sx, sy = sign(dx), sign(dy)
    dx = fabs(dx)
    dy = fabs(dy)

    swap = 0
    if dy > dx:
        swap = 1
        dx, dy = dy, dx
    m = dy / dx
    e = 0.5
    w = 1

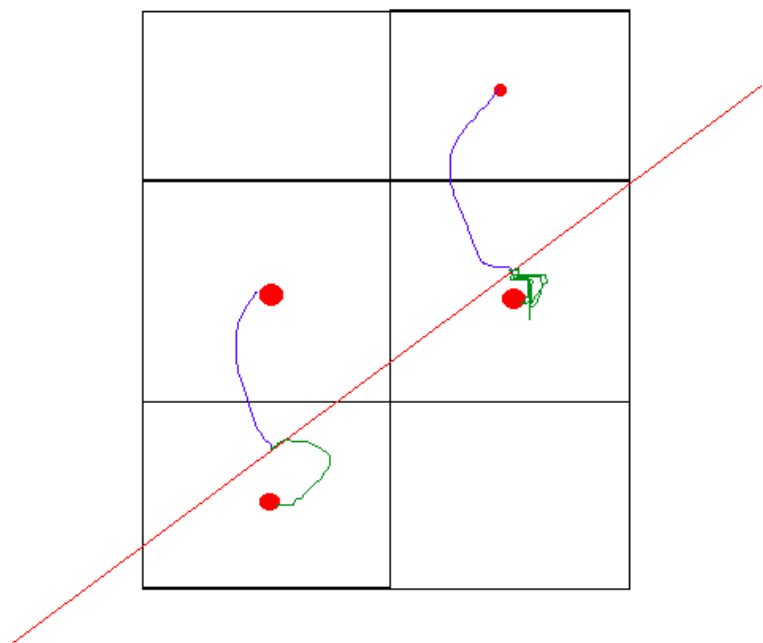
    for _ in range(int(dx + 1)):
        if swap == 0:
            print_pixel_color(x, y, e)
            print_pixel_color(x, y + sy, e - 1)
        else:
            print_pixel_color(x, y, e)
            print_pixel_color(x + sx, y, e - 1)

        if e >= w - m:
            if swap == 0:
                y += sy
            else:
                x += sx
            e -= 1
        if swap == 0:
            x += sx
        else:
            y += sy
        e += m
```

Суть состоит в том, что отрезок высвечивается толщиной в 2 пикселя. Суммарная интенсивность двух пикселей, высвечивающихся на каждом шаге постоянная $I_{i1} + I_{i2} = I_{const}$. Сглаживание осуществляется за счет перераспределения интенсивности I_{const} между двумя пикселями (На каждом шаге). Интенсивность пикселя определяется в зависимости от расстояния между пикселями и точкой, расположенной на идеальном

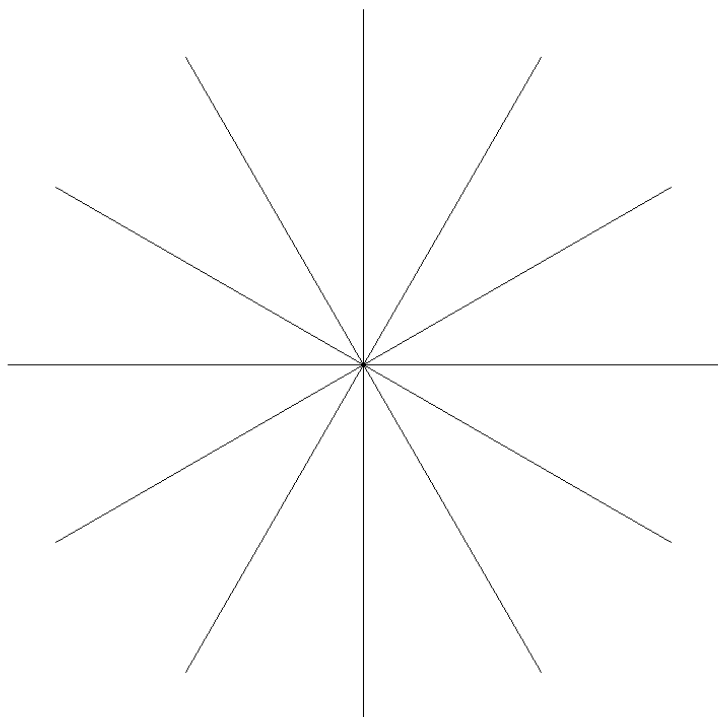
отрезке: Чем ближе пиксель расположен к точке на идеальном отрезке, тем больше его интенсивность.

На рисунку приведен пример: мы рассматриваем пиксели, между которых проходит отрезок (Они отмечены красными точками). Синим и зеленым цветом показаны расстояния между пикселями и точкой, расположенной на идеальном отрезке. Как мы видим, расстояние, отмеченное синим цветом больше, тем самым интенсивность данного пикселя будет меньше, а для пикселя, расстояние у которого отмечено зеленым цветом (Оно меньше) интенсивность будет больше. Сумма интенсивности данных пикселей будет постоянной, то есть сумма интенсивности первых двух пикселей (синий и зеленый) будет равна сумме интенсивности вторых пикселей.



Результат работы:

ЦДА:



Выбрать цвет отрезка

Рисовать фоновым цветом

Метод ЦДА

Метод Брезенхема (float)

Метод Брезенхема (int)

Брезенхем (Сглаживание)

Метод ВУ

Библиотечный метод

Начальная точка (x y):

Конечная точка (x y):

Нарисовать отрезок

Длина пучка: 350

Шаг изменения угла: 30

Нарисовать пучок

Цвет линии:

Показать временные характеристики

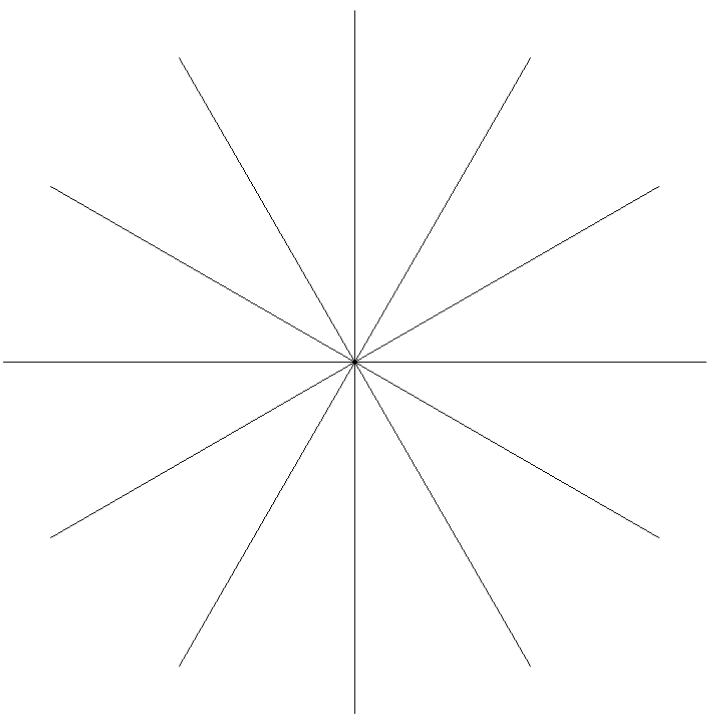
Стереть всё

Брезенхем (float)

Лабораторная работа №3

Инструкция

800x800



Выбрать цвет отрезка

Рисовать фоновым цветом

☐ Метод ЦДА

☒ Метод Брезенхема (float)

☐ Метод Брезенхема (int)

☐ Брезенхем (Сглаживание)

☐ Метод ВУ

☐ Библиотечный метод

Начальная точка (x y):

Конечная точка (x y):

Нарисовать отрезок

Длина пучка:

Шаг изменения угла:

Нарисовать пучок

Цвет линии:

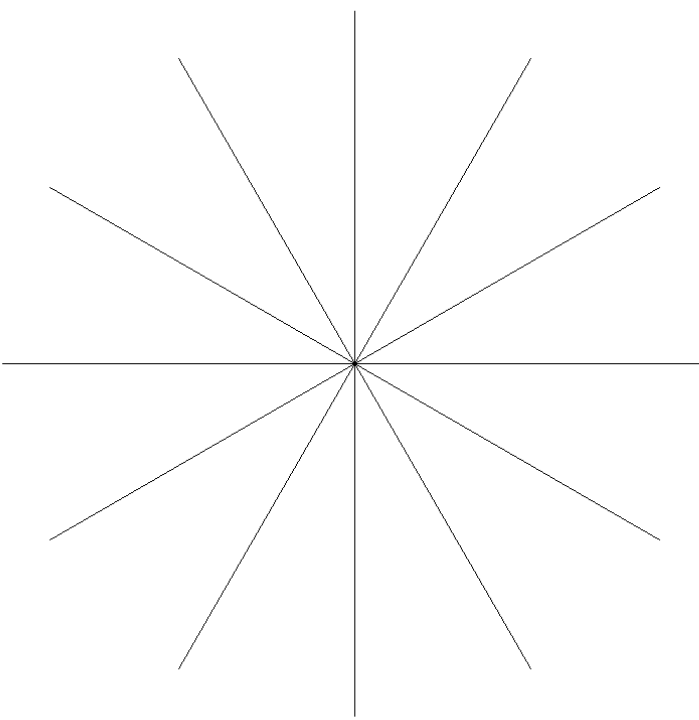
Показать временные характеристики

Стереть всё

Брезенхем (int)

Инструкция

800x800



Выбрать цвет отрезка

Рисовать фоновым цветом

Метод ЦДА

Метод Брезенхема (float)

☒ Метод Брезенхема (int)

Брезенхем (Сглаживание)

Метод ВУ

Библиотечный метод

Начальная точка (x y):

Конечная точка (x y):

Нарисовать отрезок

Длина пучка:

Шаг изменения угла:

Нарисовать пучок

Цвет линии:

Показать временные характеристики

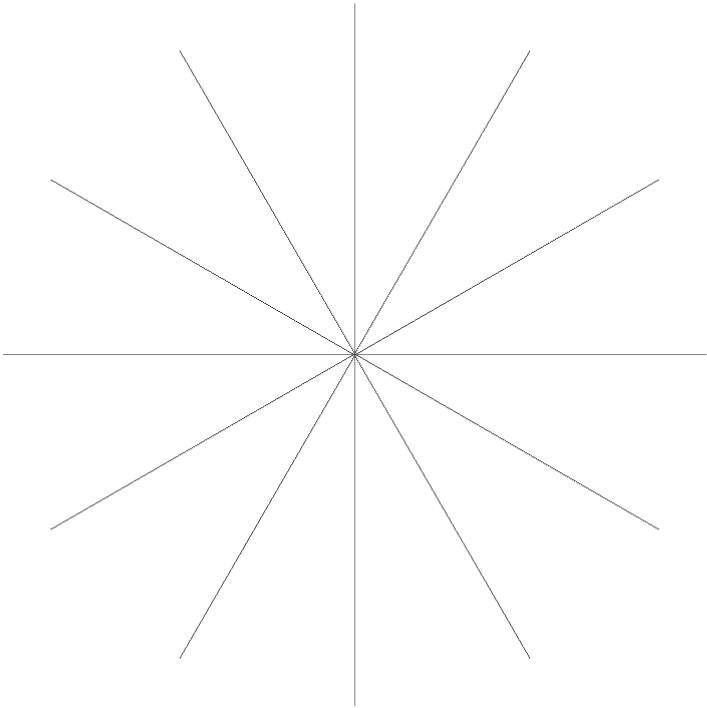
Стереть всё

Брезенхем (сглаживание)

Лабораторная работа №3

Инструкция

800x800



Выбрать цвет отрезка

Рисовать фоновым цветом

Метод ЦДА

Метод Брезенхема (float)

Метод Брезенхема (int)

Брезенхем (Сглаживание)

Метод ВУ

Библиотечный метод

Начальная точка (x y):

Конечная точка (x y):

Нарисовать отрезок

Длина пучка:

Шаг изменения угла:

Нарисовать пучок

Цвет линии:

Показать временные характеристики

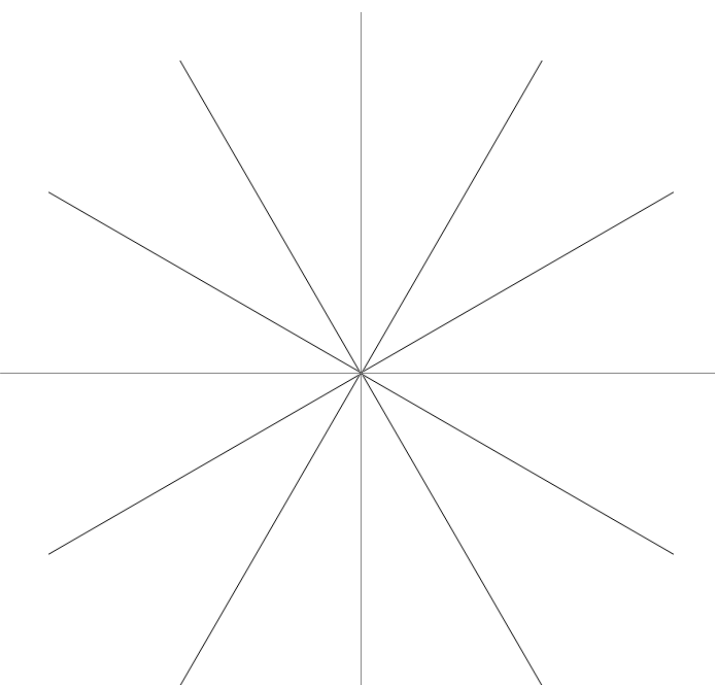
Стереть всё

ВУ

Лабораторная работа №3

Инструкция

800x800



Выбрать цвет отрезка

Рисовать фоновым цветом

Метод ЦДА

Метод Брезенхема (float)

Метод Брезенхема (int)

Брезенхем (Сглаживание)

Метод ВУ

Библиотечный метод

Начальная точка (x y):

Конечная точка (x y):

Нарисовать отрезок

Длина пучка:

Шаг изменения угла:

Нарисовать пучок

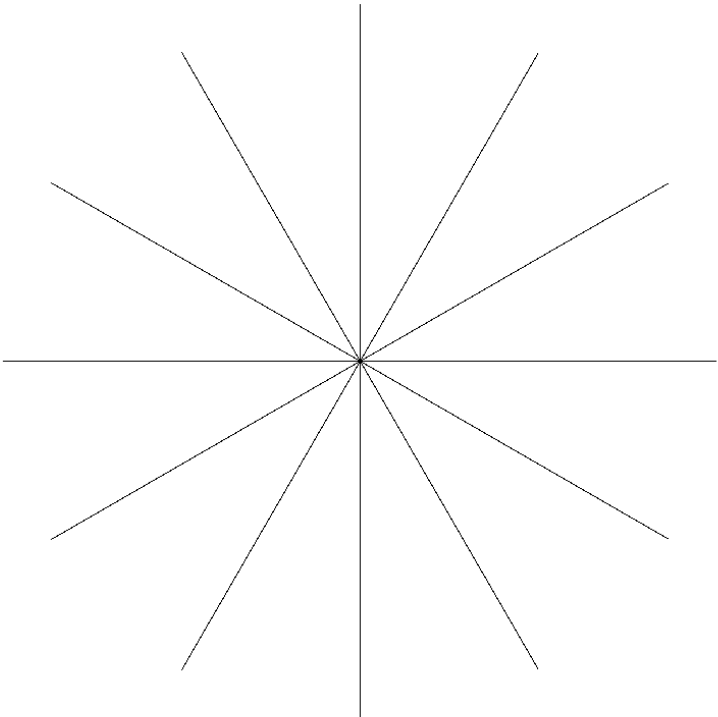
Цвет линии:

Показать временные характеристики

Библиотечный:

Инструкция

800x800



Выбрать цвет отрезка

Рисовать фоновым цветом

Метод ЦДА

Метод Брезенхема (float)

Метод Брезенхема (int)

Брезенхем (Сглаживание)

Метод ВУ

Библиотечный метод

Начальная точка (x y):

Конечная точка (x y):

Нарисовать отрезок

Длина пучка:

Шаг изменения угла:

Нарисовать пучок

Цвет линии:

Показать временные характеристики

Стереть всё

Временные характеристики:

