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)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 3

Дисциплина Компьютерная графика

Тема Реализация и исследование алгоритмов построения отрезков

Студент Хетагуров П.К.

Группа ИУ7-45

Оценка (баллы) _____

Преподаватель Куров А. В

Москва.
2020 г.

Цель работы:

Ознакомление с различными способами построения отрезка.

Техническое задание:

1) Рисование отдельных отрезков и сравнение их визуальных характеристик с помощью:
    1) Алгоритма ЦДА
    2) Алгоритма Брезенхема в действительных числах
    3) Алгоритма Брезенхема в целых числах
    4) Алгоритма Брезенхема с устранением ступенчатости
    5) Алгоритма Ву
    6) Стандартным(библиотечным) методом
2) Исследование визуальных характеристик для отрезков, расположенных во всем спектре изменения углов

Дополнительно:

3) Исследование временных характеристик
4) Исследование ступенчатости

Теоретическая часть:

Алгоритм вычерчивания отрезка (разложения отрезка в растр).

Процесс определения пикселов, наилучшим образом аппроксимирующих заданный отрезок, называется разложением в растр.

Алгоритмы разложения в растр имеют пошаговый характер(на очередном шаге высвечиваем пиксель, и производим

вычисления, используемые в следующем шаге)

Общие требования:

1. Отрезок должен выглядеть прямым, начинаться и заканчиваться в заданных точках
2. Яркость (интенсивность) не должна зависеть от длины и угла наклона и должна быть постоянной вдоль всего отрезка
3. Алгоритм должен работать быстро

Практическая часть

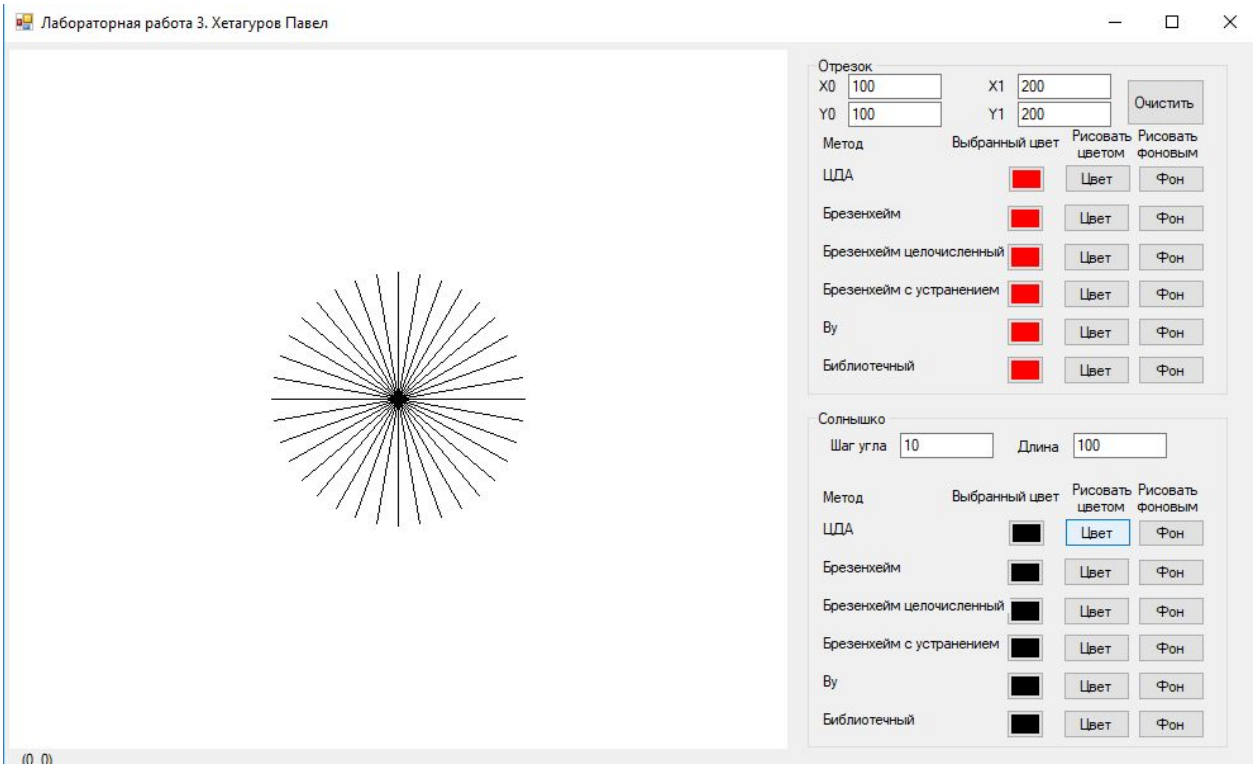
Алгоритм цифрового дифференциального анализатора (ЦДА):

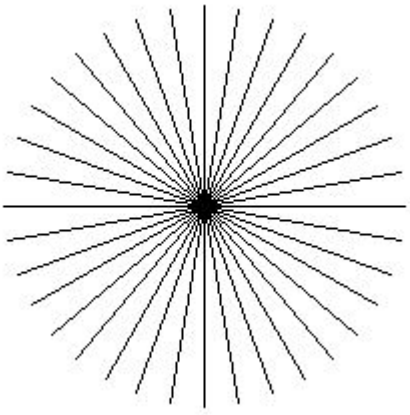
```
        public static void DrawLineDDA(Point firstPoint, Point secondPoint,
Bitmap workBitmap, Color workColor)
        {
            float deltaX = secondPoint.X - firstPoint.X;
            float deltaY = secondPoint.Y - firstPoint.Y;
            int dX = (int)Math.Abs(deltaX);
            int dY = (int)Math.Abs(deltaY);

            int lenght = dX;
            if (dX < dY)
            {
                lenght = dY;
            }

            deltaX = (deltaX / lenght);
            deltaY = (deltaY / lenght);

            float X = firstPoint.X;
            float Y = firstPoint.Y;
```

```
            for (int i = 0; i < lenght + 1; i++)
            {
                workBitmap.SetPixel((int)Math.Round(X), (int)Math.Round(Y),
workColor);
                X += deltaX;
                Y += deltaY;
            }
        }
```

Алгоритм медленный, так как в цикле построения присутствует округление.

Результат выполнения:





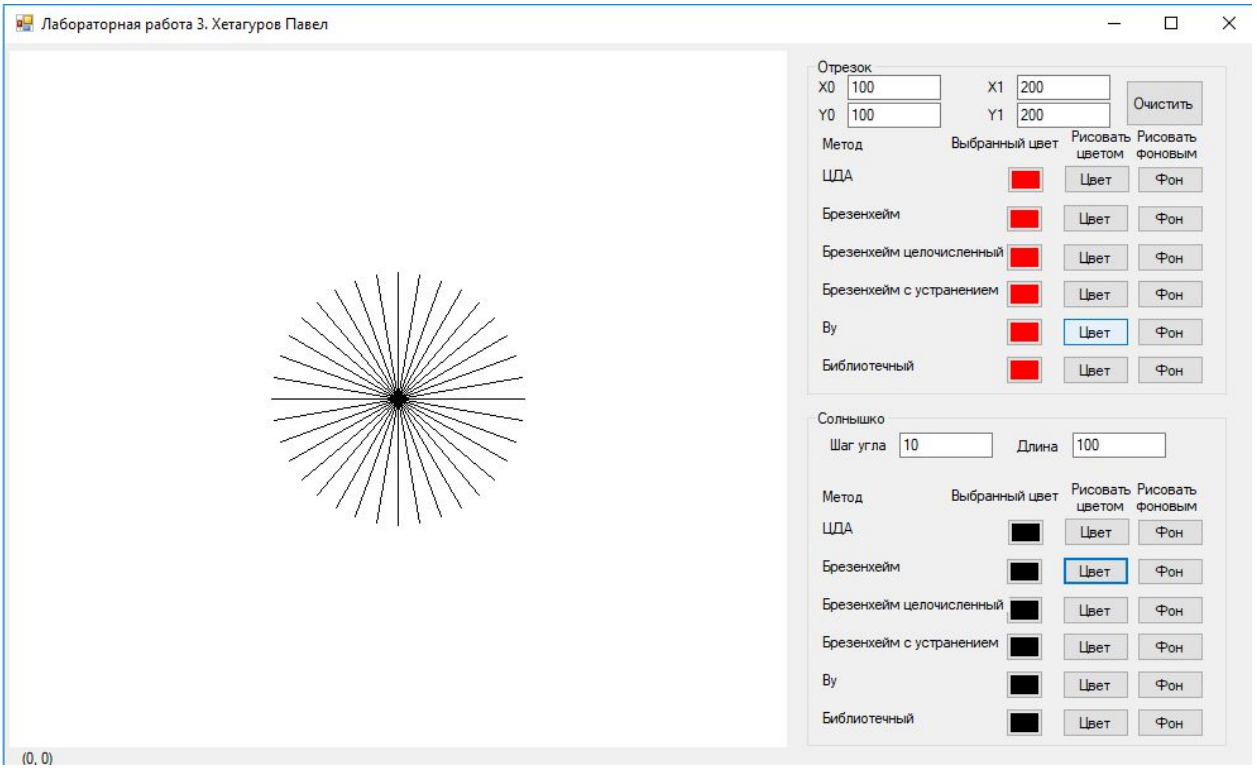
Алгоритм Брезенхема с действ. коэф-ми:

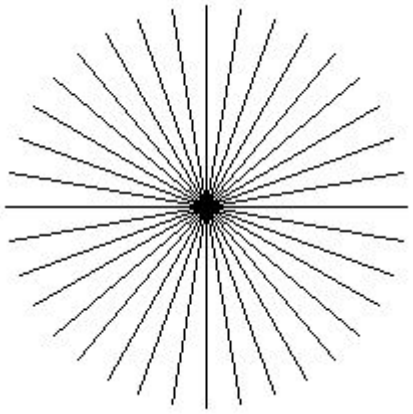
```
        public static void DrawLineBresenham(Point firstPoint, Point secondPoint,
Bitmap workBitmap, Color workColor)
        {
            int X = firstPoint.X;
            int Y = firstPoint.Y;

            int dX = secondPoint.X - firstPoint.X;
            int dY = secondPoint.Y - firstPoint.Y;

            int deltaX = Math.Sign(dX);
            int deltaY = Math.Sign(dY);

            dX = Math.Abs(dX);
            dY = Math.Abs(dY);

            bool isDxGreater = true;
            if (dX < dY)
            {
                isDxGreater = false;
                int temp = dX;
                dX = dY;
                dY = temp;
            }

            float tg = 0;
            if (dX != 0)
            {
                tg = (float)dY / dX;
            }

            float error = tg - (1.0f / 2);

            for (int i = 0; i < dX + 1; i++)
            {
```

```
            workBitmap.SetPixel((int)X, (int)Y, workColor);
            if (error >= 0)
            {
                if (isDxGreater)
                {
                    Y += deltaY;
                }
                else
                {
                    X += deltaX;
                }
                error -= 1;
            }

            if (error < 0)
            {
                if (isDxGreater)
                {
                    X += deltaX;
                }
                else
                {
                    Y += deltaY;
                }
                error += tg;
            }
        }
    }
```

Так как алгоритм оперирует вещественными числами, то могут возникнуть неточности в вычислениях (накопление ошибки) или в сравнении. Также арифметические операции над вещественными числами требуют немного больше времени, а сами вещественные числа - больше памяти, относительно целых

Результат выполнения:





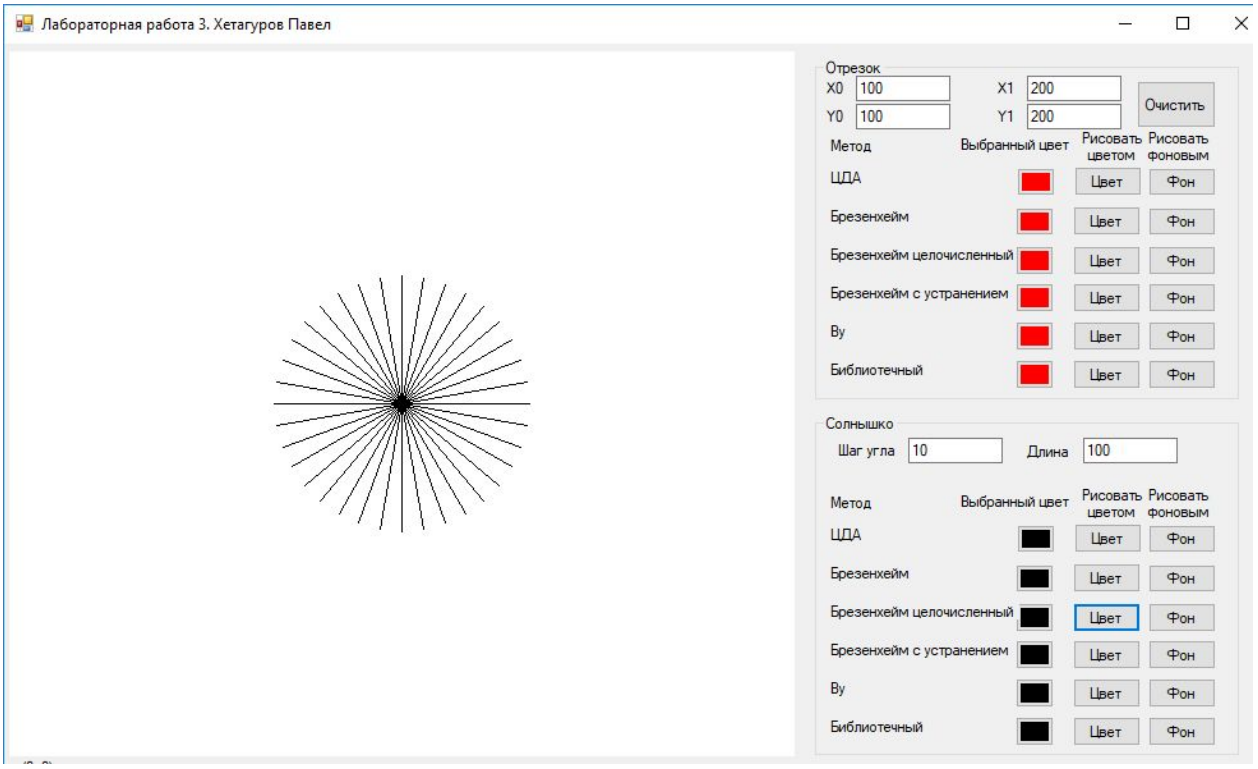
Алгоритм Брезенхема в целых числах:

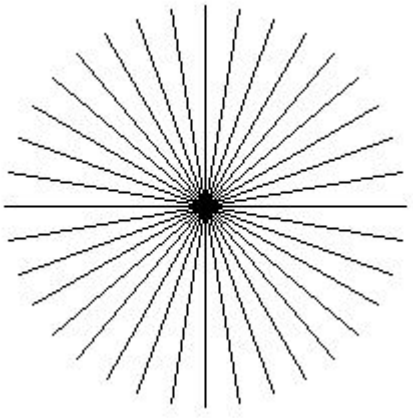
```
        public static void DrawLineBresenhamInt(Point firstPoint, Point
secondPoint, Bitmap workBitmap, Color workColor)
        {
            int X = firstPoint.X;
            int Y = firstPoint.Y;

            int dX = secondPoint.X - firstPoint.X;
            int dY = secondPoint.Y - firstPoint.Y;

            int deltaX = Math.Sign(dX);
            int deltaY = Math.Sign(dY);
```

```
        dX = Math.Abs(dX);
        dY = Math.Abs(dY);

        bool isDxGreater = true;
        if (dX < dY)
        {
            isDxGreater = false;
            int temp = dX;
            dX = dY;
            dY = temp;
        }

        int dotCount = dX + 1;
        int error = 2 * dY - dX;  // Переходим в целые
        dY *= 2;
        dX *= 2;

        for (int i = 0; i < dotCount; i++)
        {
            workBitmap.SetPixel((int)X, (int)Y, workColor);
            if (error >= 0)
            {
                if (isDxGreater)
                {
                    Y += deltaY;
                }
                else
                {
                    X += deltaX;
                }
                error -= dX;
            }

            if (error < 0)
            {
                if (isDxGreater)
                {
                    X += deltaX;
                }
                else
                {
```

```
                    Y += deltaY;
                }
                error += dY;
            }
        }
    }
```

Усовершенствованная версия алгоритма Брезенхема в действительных. Быстрее, отсутствует погрешность.





Алгоритм Брезенхема построения отрезка с устранением ступен-ти:

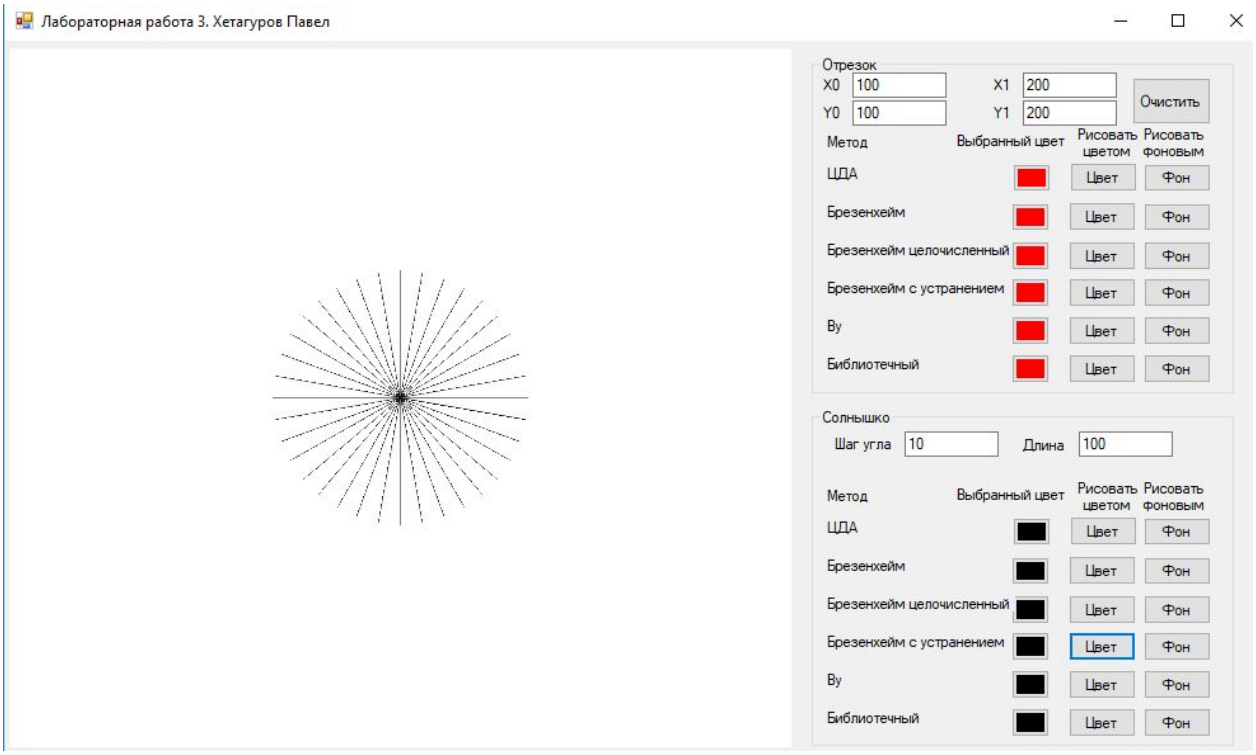
```
            }
```

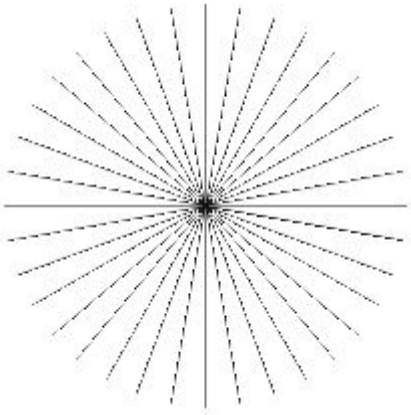
```
        public static void DrawLineBresenhamFlat(Point firstPoint, Point
secondPoint, Bitmap workBitmap, Color workColor)
        {
            int iMax = 255;  // Максимальная интенсивность

            int X = firstPoint.X;
            int Y = firstPoint.Y;

            int dX = secondPoint.X - firstPoint.X;
            int dY = secondPoint.Y - firstPoint.Y;

            int deltaX = Math.Sign(dX);
            int deltaY = Math.Sign(dY);

            dX = Math.Abs(dX);
            dY = Math.Abs(dY);

            bool isDxGreater = true;
            if (dX < dY)
            {
                isDxGreater = false;
                int temp = dX;
                dX = dY;
                dY = temp;
            }

            float tg = 0;
            if (dX != 0)
            {
                tg = (float)dY / dX * iMax;
            }

            float e = iMax / 2;
            float w = iMax - tg;

            workColor = Color.FromArgb((int)(iMax - e), workColor);
            workBitmap.SetPixel((int)X, (int)Y, workColor);
            for (int i = 0; i < dX; i++)
            {
                if (e < w)
                {
                    if (isDxGreater)
```

```
                {
                    X += deltaX;
                }
                else
                {
                    Y += deltaY;
                }
                e += tg;
            }
            else
            {
                X += deltaX;
                Y += deltaY;
                e -= w;
            }

            workColor = Color.FromArgb((int)(255 - e), workColor);
            workBitmap.SetPixel((int)X, (int)Y, workColor);
        }
    }
```

В предыдущих алгоритмах пиксел представлялся как точка, в этом - как квадрат 1х1. Пиксели высвечиваются с интенсивностью, пропорциональной площади части пиксела под отрезком.

Может использоваться для отображения закрашенных многоугольников (плоская сторона - снаружи, ступенчатая - закрашивается)





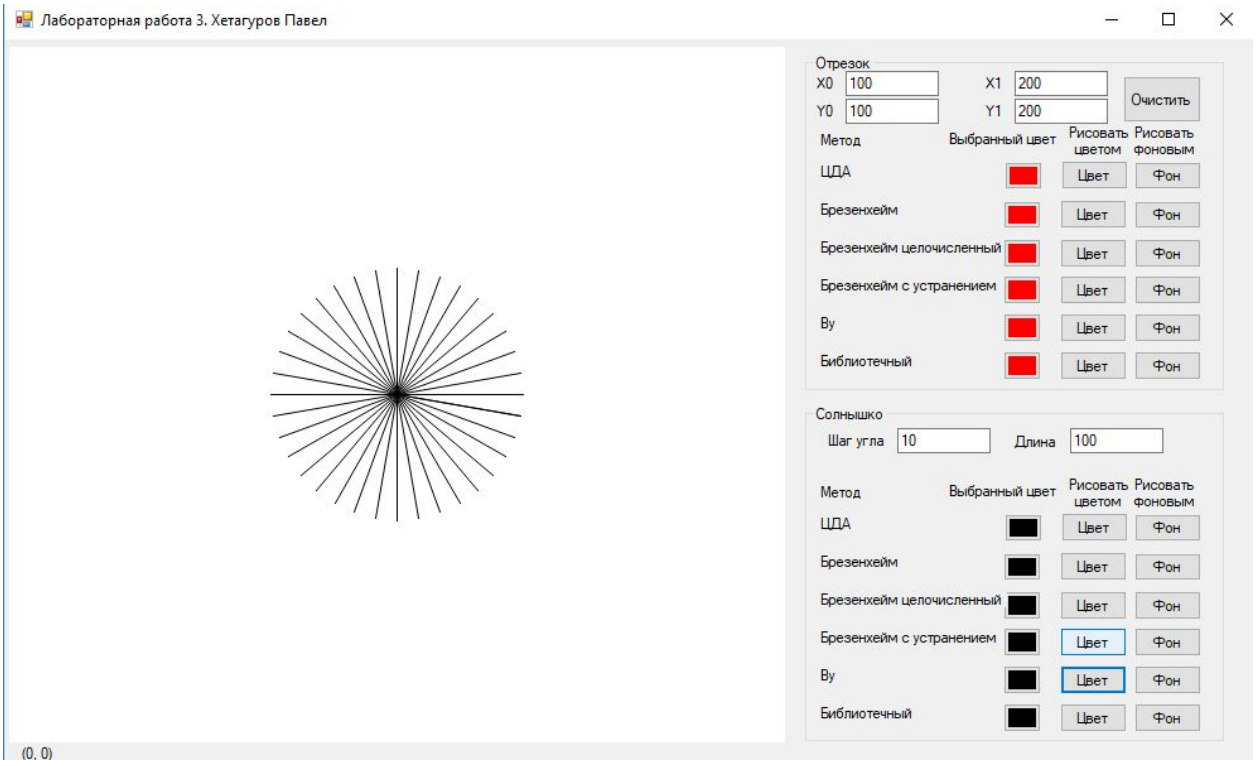
Алгоритм By:

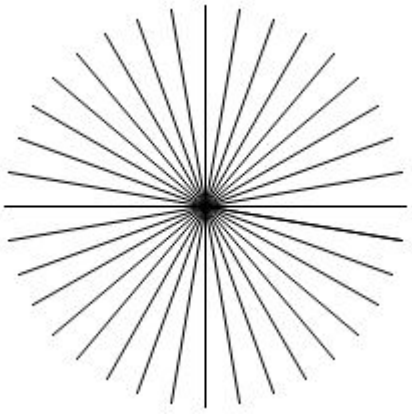
```
        public static void DrawLineVu(Point firstPoint, Point secondPoint, Bitmap
workBitmap, Color workColor)
        {
            int iMax = 255;

            int dX = secondPoint.X - firstPoint.X;
            int dY = secondPoint.Y - firstPoint.Y;

            int firstX = firstPoint.X;
            int firstY = firstPoint.Y;
            int secondX = secondPoint.X;
            int secondY = secondPoint.Y;
```

```
            bool change = false;
            if (Math.Abs(dY) > Math.Abs(dX))
            {
                SwapInt(ref firstX, ref firstY);
                SwapInt(ref secondX, ref secondY);
                SwapInt(ref dX, ref dY);
                change = true;
            }

            if (secondX < firstX)
            {
                SwapInt(ref firstX, ref secondX);
                SwapInt(ref firstY, ref secondY);
            }

            float tg = 0;
            if (dX != 0)
            {
                tg = (float)dY / dX;
            }

            double d1, d2;
            double Y = firstY;

            for (int X = firstX; X <= secondX; X++)
            {
                d1 = iMax * (Y - (int)Y);
                d2 = iMax - d1;

                if (change)
                {
                    workBitmap.SetPixel((int)Y, X, Color.FromArgb((int)(d2),
workColor));
                    workBitmap.SetPixel((int)Y + 1, X, Color.FromArgb((int)(d1),
workColor));
                }
                else
                {
                    workBitmap.SetPixel(X, (int)Y, Color.FromArgb((int)(d2),
workColor));
                    workBitmap.SetPixel(X, (int)Y + 1, Color.FromArgb((int)(d1),
workColor));
```

```
                }

                Y += tg;
            }
        }
```

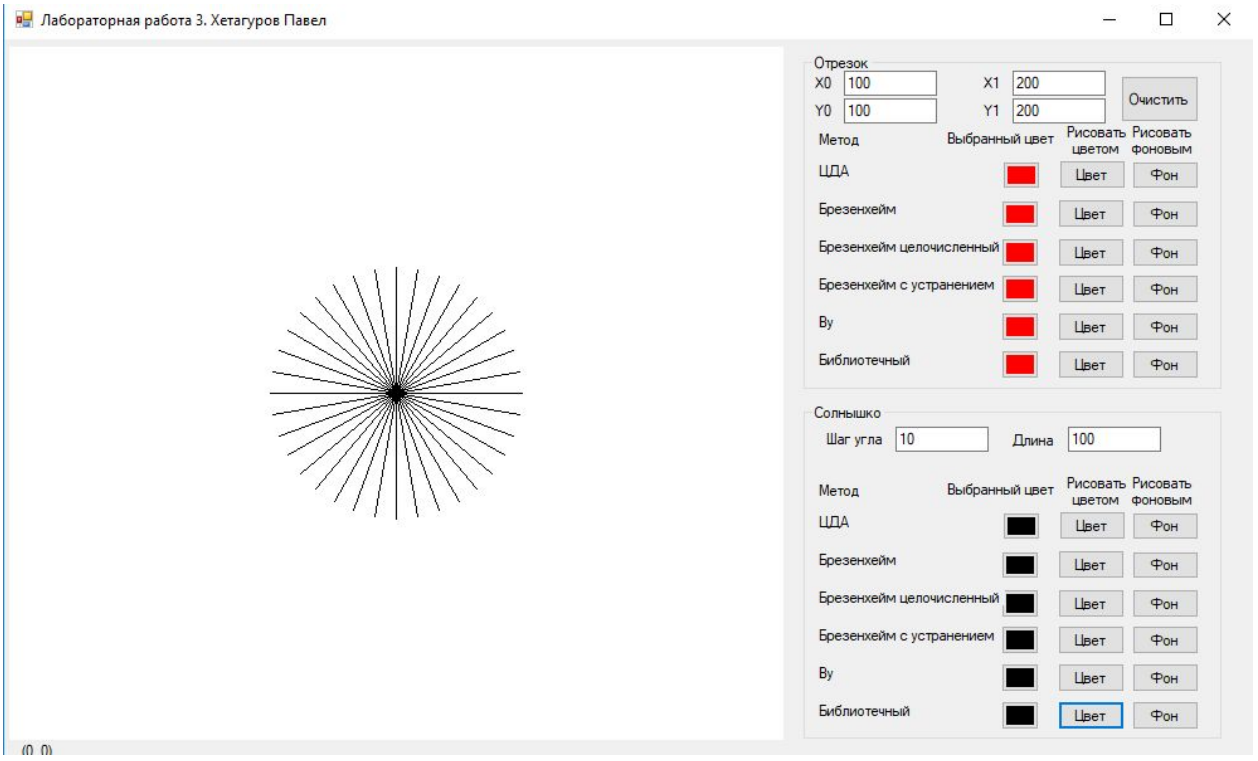
В алгоритме Ву на каждом шаге высвечивается два пиксела и линия получается в два раза толще.

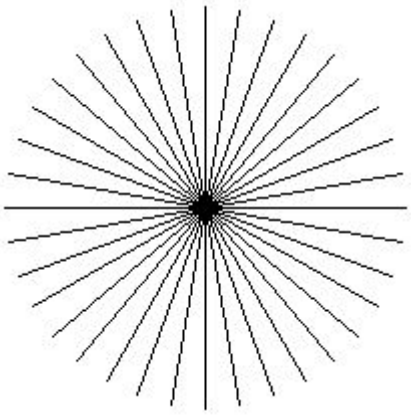




Библиотечный алгоритм:

Скорее всего реализован на основе алгоритма Брезенхема в целых числах.





Вывод:

Все алгоритмы рисуют отрезки, удовлетворяющие основным условиям