Лабораторная работа №3

Задача:

- Написать приложение/веб-приложение, иллюстрирующее работу базовых растровых алгоритмов: − пошаговый алгоритм; − алгоритм Брезенхема;
- На проверку сдаются: exe, который должен работать на любом ПК под Windows XP /веб-приложение, выложенное в общий доступ; исходный код; сопроводительная документация.

Требования и критерии оценки:

- Корректность работы методов (проиллюстрировать один пример сопроводительными вычислениями) - 50 баллов.
- Краткий отчет с указанием временных характеристик реализованных алгоритмов - 20 баллов.
- Дружелюбный и удобный интерфейс (масштаб; обязателен вывод системы координат, осей, линий сетки, подписей; пояснить, каким образом целочисленные координаты привязаны к дискретной сетке) - 30 баллов.

Использованные средства разработки:

- Windows Forms .NET, C#
- Библиотека OxyPlot (реализация плоскости Oxy)

Ход работы:

- Создание интерфейса пользователя с помощью встроенного конструктора;
- Реализация методов своего варианта:
    - Пошаговый алгоритм
    - Алгоритм Брезенхема

Выводы:

В ходе выполнения работы я:

- Создал приложение, позволяющее применять к картинкам разричные методы фильтрациимоделях.
- Углубил знания в Windows Forms .NET, C#

- Поработал с системой контроля версий Git

Разбор алгоритмов:

Пошаговый алгоритм

```
public static List<Point> DrawLinePointByPoint(Point p1, Point p2)
        {
            int dx = p2.X - p1.X;
            int dy = p2.Y - p1.Y;

            int x = p1.X;
            int y = p1.Y;

            List<Point> points = new List<Point>();

            if (Math.Abs(dx) > Math.Abs(dy))
            {
                bool steep = Math.Abs(dy) > Math.Abs(dx);
                if (steep)
                {
                    Swap(ref x, ref y);
                    Swap(ref p1);
                    Swap(ref p2);
                    dx = p2.X - p1.X;
                    dy = p2.Y - p1.Y;
```

```
            }
            if (p1.X > p2.X)
            {
                SwapX(ref p1, ref p2);
                SwapY(ref p1, ref p2);
                dx = p2.X - p1.X;
                dy = p2.Y - p1.Y;
            }
            int d = 2 * Math.Abs(dy) - Math.Abs(dx);
            int yStep = p1.Y > p2.Y ? -1 : 1;
            while (x <= p2.X)
            {
                points.Add(steep ? new Point(y, x) : new Point(x,
y));
                if (d > 0)
                {
                    y += yStep;
                    d -= 2 * Math.Abs(dx);
                }
                x += 1;
                d += 2 * Math.Abs(dy);
            }
        }
        else
```

```
        {
            bool steep = Math.Abs(dx) > Math.Abs(dy);
            if (steep)
            {
                Swap(ref x, ref y);
                Swap(ref p1);
                Swap(ref p2);
                dx = p2.X - p1.X;
                dy = p2.Y - p1.Y;
            }
            if (p1.Y > p2.Y)
            {
                SwapY(ref p1, ref p2);
                SwapX(ref p1, ref p2);
                dy = p2.Y - p1.Y;
                dx = p2.X - p1.X;
            }
            int d = 2 * Math.Abs(dx) - Math.Abs(dy);
            int xStep = p1.X > p2.X ? -1 : 1;
            while (y <= p2.Y)
            {
                points.Add(steep ? new Point(y, x) : new Point(x,
y));
                if (d > 0)
```

```
            {
                x += xStep;
                d -= 2 * Math.Abs(dy);
            }
            y += 1;
            d += 2 * Math.Abs(dx);
        }
    }

    return points;
}
```

- Он используется для приближенного нахождения значения функции на следующем шаге, исходя из текущего значения и значения производной функции.
- Алгоритм разбивает интервал времени на небольшие шаги и вычисляет приближенные значения функции на каждом шаге, используя текущее значение и производную.
- Чем меньше шаг, тем более точное приближение получается, но вычислительная нагрузка может быть высокой.
- Алгоритм пошагового широко применяется в численных методах решения дифференциальных уравнений и других задачах, где требуется приближенное численное решение.

Алгоритм Брезенхема:

- Он основан на идее использования только целочисленных вычислений и приближенных значений для рисования линии между двумя точками на растре.

- Алгоритм определяет, какие пиксели должны быть включены в рисуемую линию, исходя из углового коэффициента наклона и расстояния между двумя точками.
- Он выбирает пиксели с наименьшей ошибкой при приближенном рисовании линии, чтобы достичь наилучшего приближения и сохранить прямолинейность и плавность.

```
public static List<Point> DrawLineBresenham(Point p1, Point p2)
    {
        int dx = Math.Abs(p2.X - p1.X);
        int dy = Math.Abs(p2.Y - p1.Y);

        bool steep = dy > dx;

        if (steep)
        {
            Swap(ref p1);
            Swap(ref p2);
            dx = Math.Abs(p2.X - p1.X);
            dy = Math.Abs(p2.Y - p1.Y);
        }

        if (p1.X > p2.X)
        {
            SwapX(ref p1, ref p2);
            SwapY(ref p1, ref p2);
```

```
    }

    int yStep = p1.Y < p2.Y ? 1 : -1;
    int d = 2 * dy - dx;
    int y = p1.Y;

    List<Point> points = new List<Point>();
    for (int x = p1.X; x <= p2.X; x++)
    {
        points.Add(steep ? new Point(y, x) : new Point(x, y));

        if (d > 0)
        {
            y += yStep;
            d -= 2 * dx;
        }
        d += 2 * dy;
    }

    return points;
}
```