**Учреждение образования**
**«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**
**ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»**

**Кафедра интеллектуальных информационных технологий**

# Отчёт
# по курсу «Графический интерфейс интеллектуальных систем»



| | |
|---|---|
| **Выполнил студент группы 121701:** | **Дятлов Е.А.** |
| **Проверила:** | **Жмырко А.В.** |



**МИНСК**
**2023**

**Тема:** Генерация отрезков.

**Цель: Разработка элементарного графического редактора, который будет реализовывать построение отрезков с помощью трех алгоритмов: алгоритма ЦДА, целочисленного алгоритма Брезенхема и алгоритма Ву. Редактор должен предоставлять возможность выбора алгоритма через панель инструментов и отображать пошаговое решение на дискретной сетке в отладочном режиме.**

**Теоретические сведения:**

**Алгоритм ЦДА (Цифрового дифференциального анализатора)** используется для генерации отрезков и основывается на простом линейном интерполировании между двумя точками.
**Целочисленный алгоритм Брезенхема** разбивает отрезок на последовательность пикселей, приближая его к линии с наименьшим смещением. Это позволяет избежать использования операций с плавающей точкой.
**Алгоритм Ву**, также известный как алгоритм Ву-Сям Цяо, использует аппроксимацию градиента для сглаживания линий и предотвращения эффекта "лестницы".

**Алгоритм Цифрового Дифференциального Анализатора (ЦДА)** является одним из простейших алгоритмов для построения отрезков. Его схема выглядит следующим образом:

1. Получить начальную и конечную точки отрезка (x0, y0) и (x1, y1).
2. Вычислить разность координат по оси X и оси Y: dx = x1 - x0, dy = y1 - y0.
3. Определить максимальную разность по модулю: steps = max(|dx|, |dy|).
4. Вычислить приращения по осям X и Y: x_increment = dx / steps, y_increment = dy / steps.
5. Инициализировать текущие координаты текущей точки отрезка: x = x0, y = y0.
6. Для каждого шага от 0 до steps:
7. Нарисовать точку (x, y).
8. Обновить текущие координаты: x = x + x_increment, y = y + y_increment.
9. Алгоритм ЦДА основан на идее инкрементального приращения по координатам X и Y, чтобы на каждом шаге перейти к следующей точке на отрезке. Он работает с вещественными числами и не требует округления.

Время выполнения алгоритма **ЦДА** зависит от длины отрезка и может быть оценено как O(steps), где steps - максимальная разность по модулю между координатами X и Y начальной и конечной точек отрезка.

Схема алгоритма **Брезенхема (Bresenham)** для построения отрезков выглядит следующим образом:

1. Получить начальную и конечную точки отрезка (x0, y0) и (x1, y1).
2. Вычислить разность координат по оси X и оси Y: dx = x1 - x0, dy = y1 - y0.
3. Определить знаки разностей: sx = sign(dx), sy = sign(dy).
4. Преобразовать разности в положительные числа: dx = abs(dx), dy = abs(dy).
5. Если dx > dy, то установить переменную e = dx/2, иначе установить e = -dy/2.
6. Инициализировать текущие координаты текущей точки отрезка: x = x0, y = y0.
7. Для каждого шага от 0 до max(dx, dy):

8. Нарисовать точку (x, y).
9. Обновить переменную e:
10. Если dx > dy, то увеличить e на dy, иначе увеличить x на sx.
11. Если e >= dx, то увеличить y на sy и уменьшить e на dx.
12. Нарисовать последнюю точку (x1, y1).

**Алгоритм Брезенхема** основан на использовании целочисленных операций и минимизации вычислений с плавающей запятой. Он определяет, какой пиксель выбрать на каждом шаге с помощью использования переменной e, которая отслеживает ошибку округления.

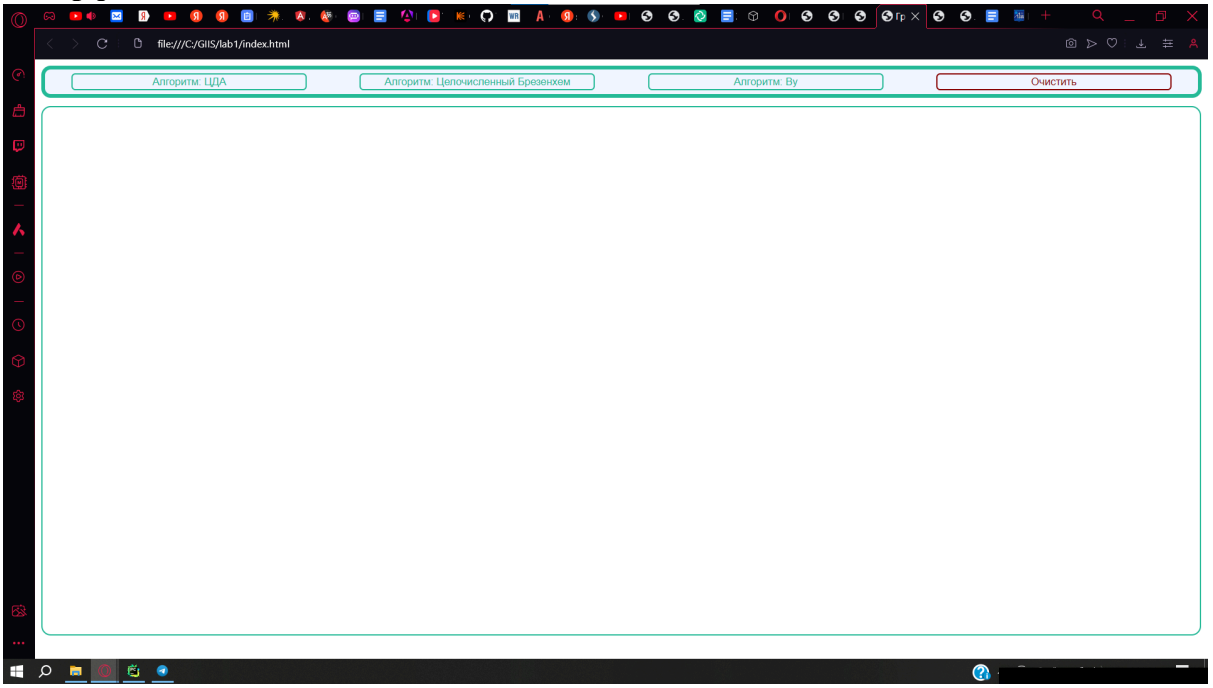
Алгоритмическая оценка **алгоритма Брезенхема**: O(|dx| + |dy|), где dx - разница между конечной и начальной координатой x, dy - разница между конечной и начальной координатой y. Алгоритм работает эффективно и требует только целочисленных вычислений, что делает его быстрым и подходящим для использования в графических приложениях.

Схема **алгоритма By** выглядит следующим образом:

1. Вычислить целочисленные координаты начальной и конечной точек отрезка (x1, y1) и (x2, y2).
2. Вычислить разницу между координатами x и y: dx = x2 - x1, dy = y2 - y1.
3. Если |dy| > |dx|, поменять местами значения x и y.
4. Если x2 < x1, поменять местами значения x1 и x2, y1 и y2.
5. Вычислить значение изменения координаты y для каждого шага: gradient = dy / dx.
6. Инициализировать переменные x и y со значениями начальной точки (x1, y1).
7. Вычислить целочисленную часть значения y и нарисовать пиксель с координатами (x, int(y)).
8. Для каждого шага:
9. Инкрементировать значение x на 1.
10. Вычислить десятичную часть значения y и увеличить ее на значение градиента.
11. Вычислить целочисленную часть значения y и нарисовать пиксель с координатами (x, int(y)).
12. Вычислить целочисленную часть значения y + 1 и нарисовать пиксель с координатами (x, int(y) + 1).
13. Алгоритмическая оценка алгоритма By: O(|x2 - x1|), где x1 и x2 - начальная и конечная координаты x. Алгоритм By также работает эффективно и обеспечивает сглаживание линий без использования операций с плавающей точкой.

Сравнение результатов алгоритма **By** с алгоритмом **Брезенхема** позволяет заметить, что алгоритм By создает более гладкие и сглаженные линии без эффекта "лестницы", в то время как алгоритм Брезенхема имеет более четкие и ступенчатые линии.

**Интерфейс:**




**Листинг кода:**

**index.html**

```
<!DOCTYPE html>
<html>
<head>
   <title>Графический редактор</title>
   <style>
       #canvas {
           cursor: pointer;
           border: 2px solid #25B99A;
           border-radius: 12px;
           color: #25B99A;
       }

       #toolbar {
           background: aliceblue;
           margin-bottom: 10px;
           border: 5px solid #25B99A;
           border-radius: 12px;
           display: flex;
           justify-content: space-around;
       }

       .algorithm_button {
           transition-duration: 0.5s;
           margin: 5px;
           border:2px solid #25B99A;
           color: #25B99A;
           background-color: aliceblue;
           border-radius: 5px;
           width: 300px;
```

```
        }

        .algorithm_button:hover{
            border:2px solid aliceblue;
            color: aliceblue;
            background-color: #25B99A;
        }

        .delete_button{
            transition-duration: 0.5s;
            margin: 5px;
            border:2px solid darkred;
            color: darkred;
            background-color: aliceblue;
            border-radius: 5px;
            width: 300px;
        }

        .delete_button:hover{
            border:2px solid aliceblue;
            color: aliceblue;
            background-color: darkred;
        }
    </style>
</head>
<body>
<div id="toolbar">
    <button class="algorithm_button"
onclick="setAlgorithm('dda')">Алгоритм: ЦДА</button>
    <button class="algorithm_button"
onclick="setAlgorithm('bresenham')">Алгоритм: Целочисленный
Брезенхем</button>
    <button class="algorithm_button"
onclick="setAlgorithm('wu')">Алгоритм: Ву</button>
    <button class="delete_button"
onclick="clearCanvas()">Очистить</button>
</div>
<canvas id="canvas" width="1200" height="1200"></canvas>

<script src="app.js"></script>
</body>
</html>
```

**app.js**

```
const canvas = document.getElementById('canvas');
canvas.width = 1475;
canvas.height = 670;
document.body.appendChild(canvas);

const ctx = canvas.getContext('2d');
let isDrawing = false;
let lastX = 0;
let lastY = 0;

let mode = 'cda'

function drawLine(startX, startY, endX, endY) {
   if (mode === 'cda'){
       const dx = endX - startX;
       const dy = endY - startY;

       const steps = Math.max(Math.abs(dx), Math.abs(dy));

       const xIncrement = dx / steps;
       const yIncrement = dy / steps;

       let currentX = startX;
       let currentY = startY;

       for (let i = 0; i <= steps; i++) {
           console.log(Math.round(currentX) + ' | ' +
Math.round(currentY));
           ctx.fillRect(Math.round(currentX),
Math.round(currentY), 1, 1);
           currentX += xIncrement;
           currentY += yIncrement;
       }

console.log('-----------------------------------------');
       return;
   }
   if (mode === 'bresenham'){
       const dx = Math.abs(endX - startX);
       const dy = Math.abs(endY - startY);
       const sx = startX < endX ? 1 : -1;
       const sy = startY < endY ? 1 : -1;
       let err = dx - dy;

       while (startX !== endX || startY !== endY) {
           console.log(Math.round(startX) + ' | ' +
Math.round(startY));
           ctx.fillRect(startX, startY, 1, 1);
```

```
            const err2 = err * 2;

            if (err2 > -dy) {
                err -= dy;
                startX += sx;
            }

            if (err2 < dx) {
                err += dx;
                startY += sy;
            }
        }
        console.log(Math.round(endX) + ' | ' +
Math.round(endY));
        ctx.fillRect(endX, endY, 1, 1);

console.log('-----------------------------------------');
        return;
    }
    if (mode === 'wu'){
        let y2 = endY;
        let y1 = startY;
        let x1 = startX;
        let x2 =endX;
        const dx = x2 - x1;
        const dy = y2 - y1;
        const gradient = dy / dx;

        let x = x1;
        let y = y1;
        console.log(Math.round(x) + ' | ' + Math.round(y));
        ctx.fillRect(x, y, 1, 1);

        if (Math.abs(gradient) <= 1) {
            const yEnd = y2 < y1 ? y1 - 1 : y1 + 1;

            let intery = y + gradient;

            for (let x = x1 + 1; x <= x2 - 1; x++) {
                ctx.fillRect(x, Math.floor(intery), 1, 1);
                ctx.fillRect(x, Math.floor(intery) + 1, 1, 1);
                console.log(Math.round(x) + ' | ' +
Math.round(intery));
                intery += gradient;
            }
            console.log(Math.round(x2) + ' | ' +
Math.round(yEnd));
            ctx.fillRect(x2, yEnd, 1, 1);

console.log('-----------------------------------------');
        } else {
```

```
            const xEnd = x2 < x1 ? x1 - 1 : x1 + 1;

            let interx = x + (1 / gradient);

            for (let y = y1 + 1; y <= y2 - 1; y++) {
                ctx.fillRect(Math.floor(interx), y, 1, 1);
                ctx.fillRect(Math.floor(interx) + 1, y, 1, 1);
                console.log(Math.round(interx) + ' | ' +
Math.round(y));
                interx += (1 / gradient);
            }
            console.log(Math.round(xEnd) + ' | ' +
Math.round(y2));
            ctx.fillRect(xEnd, y2, 1, 1);

console.log('-----------------------------------------');
        }
    }

    ctx.beginPath();
    ctx.moveTo(startX, startY);
    ctx.lineTo(endX, endY);
    ctx.stroke();
}

let startX = 0;
let startY = 0;
let lines = []

canvas.addEventListener('mousedown', (e) => {
    isDrawing = true;
    startX = e.offsetX;
    startY = e.offsetY;
});

canvas.addEventListener('mousemove', (e) => {
    if (isDrawing) {
        const [currentX, currentY] = [e.offsetX, e.offsetY];
        redrawLines()
        drawLine(startX, startY, currentX, currentY);

    }
});
function redrawLines(){
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    for (let line of lines){
        ctx.moveTo(line.x1, line.y1);
        ctx.lineTo(line.x2, line.y2);
        ctx.stroke();
    }
}
```

```
canvas.addEventListener('mouseup', (e) => {
    drawLine(startX, startY, e.offsetX, e.offsetY);
    let line = new Line();
    line.x1 = startX;
    line.y1 = startY;
    line.x2 = e.offsetX;
    line.y2 = e.offsetY;
    lines.push(line);
    isDrawing = false;
});

function clearCanvas(){
    lines = []
    ctx.clearRect(0, 0, canvas.width, canvas.height);
}

function setAlgorithm(alg){
    mode = alg;
}

class Line{
    constructor() {
        this.x1 = 0;
        this.y1 = 0;
        this.x2 = 0;
        this.y2 = 0;
    }
}
```

**Вывод:**
Редактор предоставляет пользователю гибкость выбора алгоритма построения отрезков и отображение пошагового решения на дискретной сетке в отладочном режиме, что позволяет лучше понять и визуализировать внутренние механизмы каждого алгоритма.

Использование трех различных алгоритмов (ЦДА, целочисленного Брезенхема и Ву) позволяет сравнить их производительность и качество построения отрезков на дискретной сетке. Алгоритм ЦДА является простым и линейным, но может иметь некоторые проблемы с округлением и плавными переходами. Целочисленный алгоритм Брезенхема, основанный на использовании целочисленных операций, обеспечивает точное построение отрезков и может быть более эффективным. Алгоритм Ву предлагает градиентное сглаживание и создание плавных переходов между цветами на линии.

Разработка такого графического редактора требует учета основных принципов каждого алгоритма и их корректной реализации в программном коде. Важно обеспечить возможность выбора алгоритма через панель инструментов, чтобы пользователь мог экспериментировать и сравнивать результаты. Отображение пошагового решения на дискретной сетке в отладочном режиме позволяет лучше понять внутренние детали и работу каждого алгоритма.

Разработка такого графического редактора с поддержкой трех алгоритмов построения отрезков будет полезным инструментом для обучения и исследования алгоритмов графики. Это позволит пользователям изучать различные алгоритмы и их особенности, а также сравнивать их производительность и качество построения отрезков.