МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет автоматики и вычислительной техники
Кафедра автоматики

# ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине: «Компьютерная графика»
Генерация отрезков

Выполнили:
Студенты гр. АВТ-418
Юрков Владислав Юрьевич
Куриленко Платон Семенович
«___» _____ 2025 г.

Проверил:
Надежницкая В. А.

_____
(оценка, подпись)

Новосибирск
2025

**Цель работы:** Изучить и реализовать два классических алгоритма растрового построения отрезка: несимметричный ЦДА и Брезенхема

**Задание:**

1. Задайте любым способом начало и конец отрезка.
2. Используя разные цвета, постройте отрезки с помощью методов несимметричного ЦДА и Брезенхема. Между построением отрезков выдержать паузу.
3. Нарисуйте другим цветом «настоящий» отрезок.
4. Сравните алгоритмы ЦДА и Брезенхема. Приведите примеры их сходства и различия.

## Теоретические основы:

Используются два классических алгоритма растеризации отрезка - несимметричный ЦДА и Брезенхем - для дискретизации идеальной прямой на пиксельной решетке максимально близко к геометрической линии и последующего сравнения точности выбора пикселей.

**Несимметричный ЦДА:**

Уравнение прямой с шагом по «длинной» оси и вещественными приращениями dx/length и dy/length для равномерного продвижения и округления до ближайшего пикселя.

**Алгоритм Брезенхема:**

Тест серединной точки через «ошибку» e: накапливается отклонение и при достижении порога делается шаг по «короткой» оси.

## Практическая часть:

Реализовано сравнение ЦДА и Брезенхема с помощью WEB-технологий: каждый алгоритм рисует один и тот же отрезок разным цветом.

**Используемые технологии**

Typescript, Vue, Nuxt, Tailwind CSS, canvas, Daysi UI

**Организация кода**

- Nuxt архитектура с SSR для просмотра всех лабораторных работ
- Серверные компоненты для реализации подсветки синтаксиса и интеграции кода в компоненты для отрисовки

## Сравнение алгоритмов ЦДА и Брезенхема

Визуальные различия. При наложении отрезков, построенных ЦДА и Брезенхемом (рис. 3), часть пикселей различается из-за разных правил округления: ЦДА выбирает пиксель через округление накопленных вещественных координат (рис. 1), а Брезенхем - через знак целочисленной «ошибки» и тест серединной точки (рис. 2). Эти расхождения чаще проявляются на пограничных участках, где идеальная прямая почти поровну «делит» соседние пиксели, поэтому выбор может отличаться на единичный пиксель вдоль короткой оси.

Влияние разрешения. На высоком разрешении различия становятся едва заметными: относительный вклад одного «несовпавшего» пикселя на фоне общей длины отрезка мал, и визуальная ломаная почти совпадает. Чем длиннее отрезок и выше разрешение, тем меньше видна разница в выборе отдельных клеток.

Производительность. Алгоритм Брезенхема использует только целочисленные операции (инкременты/декременты и сравнение с нулём), избегая делений и вещественной арифметики. Несимметричный ЦДА опирается на вещественные приращения и округление, что теоретически медленнее. Брезенхем остаётся эталоном по эффективности.

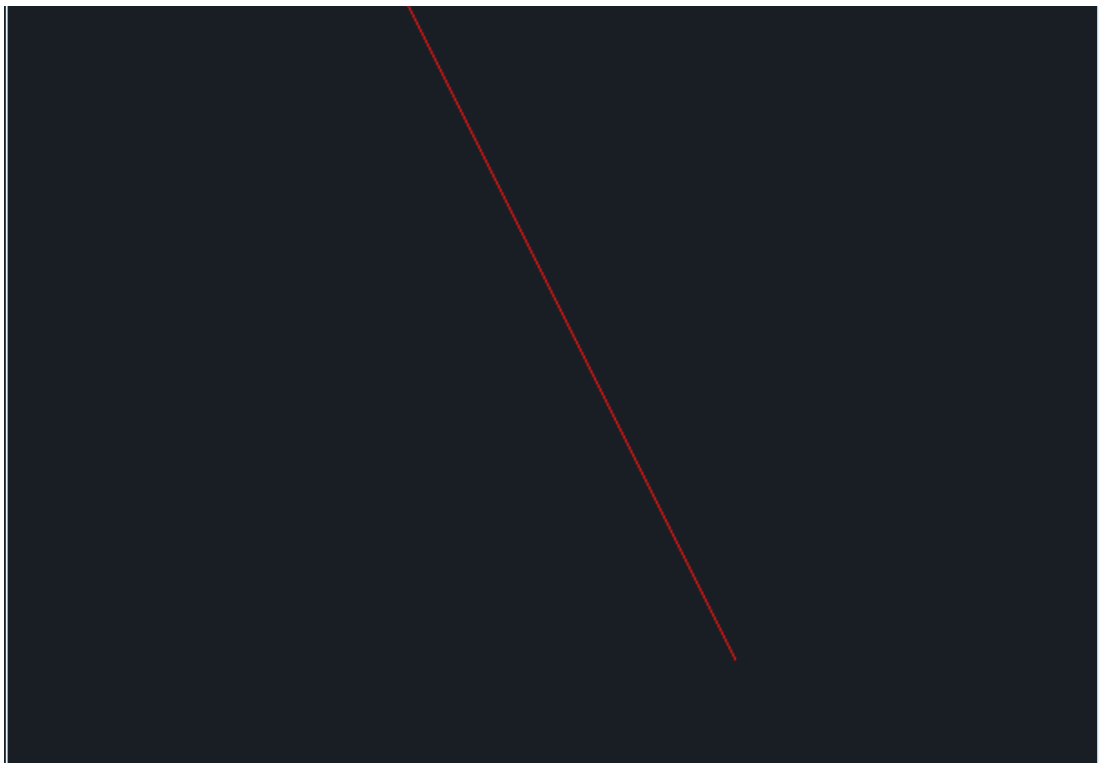

Рисунок 1 - Отрезок, построенный несимметричным ЦДА

Рисунок 2 - Отрезок, построенный алгоритмом Брезенхэма

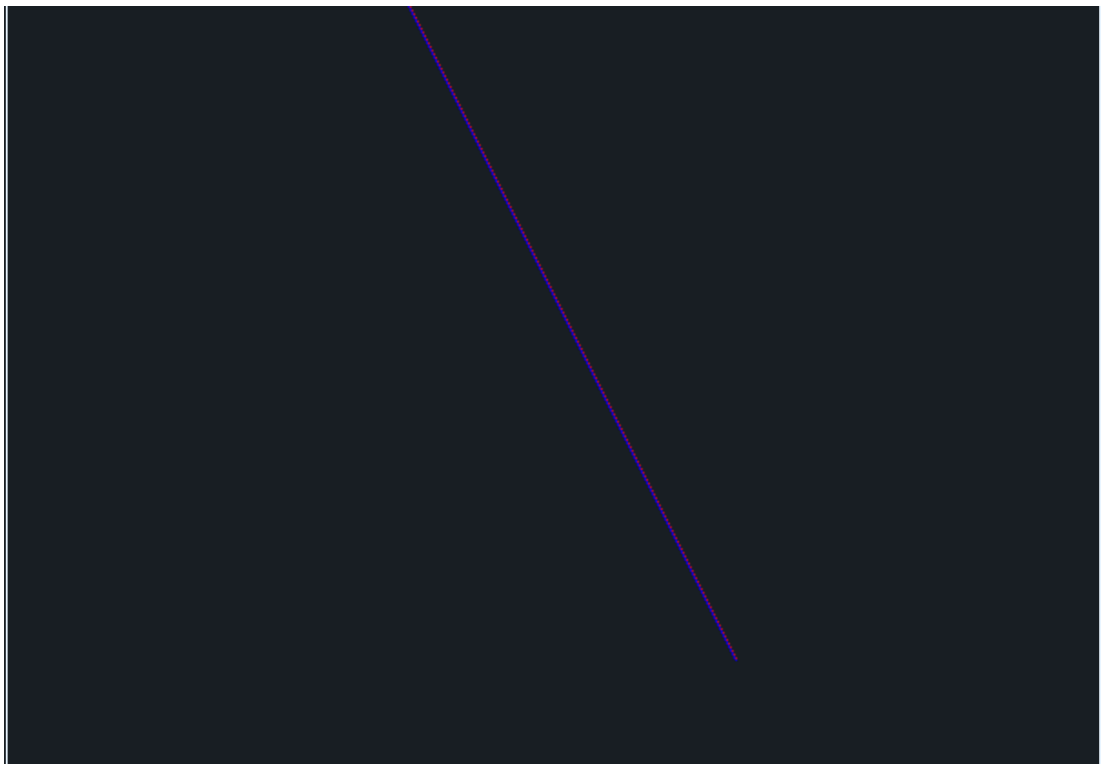

Рисунок 3 - Наложение ЦДА и Брезенхэма

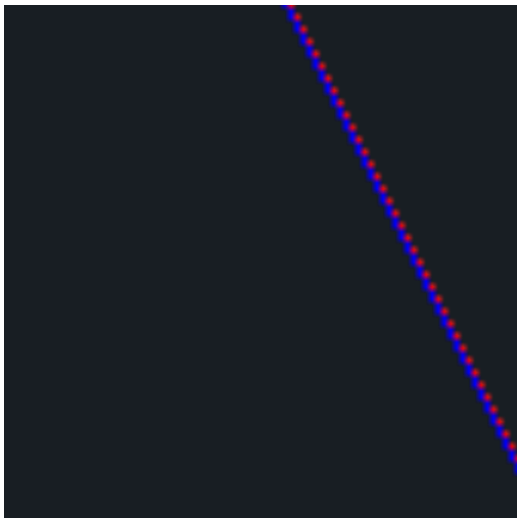

Рисунок 4 - Приближение рисунка 3

**Вывод:** В ходе лабораторной работы были изучены и реализованы два классических алгоритма построения отрезков на растровом дисплее - несимметричный алгоритм цифрового дифференциального анализатора (ЦДА) и алгоритм Брезенхема.

# Приложение

```
const renderCDA = (
    x0: number,
    y0: number,
    x1: number,
    y1: number,
    ctx: CanvasRenderingContext2D,
) => {
    const dx = x1 - x0;
    const dy = y1 - y0;

    const steps = Math.abs(dx) > Math.abs(dy) ? Math.abs(dx) : Math.abs(dy);

    const xInc = dx / steps;
    const yInc = dy / steps;

    let x = x0;
    let y = y0;

    ctx.fillStyle = 'red';

    for (let i = 0; i <= steps; i++) {
        ctx.fillRect(Math.round(x), Math.round(y), 1, 1);
        x += xInc;
        y += yInc;
    }

    ctx.save();
};
```

```

export default function render(
    x0: number,
    y0: number,
    x1: number,
    y1: number,
    ctx: CanvasRenderingContext2D,
) {
    renderCDA(x0, y0, x1, y1, ctx);
    renderBresenham(x0, y0, x1, y1, ctx);
}
```

```

const renderBresenham = (
    x0: number,
    y0: number,
    x1: number,
    y1: number,
    ctx: CanvasRenderingContext2D,
) => {
    const dx = Math.abs(x1 - x0);
    const dy = Math.abs(y1 - y0);
    const sx = x0 < x1 ? 1 : -1;
    const sy = y0 < y1 ? 1 : -1;
    let err = dx - dy;

    ctx.fillStyle = 'blue';

    while (true) {
        ctx.fillRect(x0, y0, 1, 1);

        if (x0 === x1 && y0 === y1) break;

        const e2 = 2 * err;

        if (e2 > -dy) {
            err -= dy;
            x0 += sx;
        }

        if (e2 < dx) {
            err += dx;
            y0 += sy;
        }
    }

    ctx.save();
};
```