МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра вычислительной техники

ОТЧЁТ
по лабораторной работе №7
«Алгоритмы отсечения отрезков»

по дисциплине: «Компьютерная графика»

Выполнил(а):
Факультет: АВТФ
Группа: АВТ-418
Студенты:
Юрков Владислав
Куриленко Платон

Проверил(а):

Надежницкая
Вероника Анатольевна

«16» декабря 2025 г.

«___» _____ 2025

_____
(оценка, подпись)

Новосибирск
2025

Цель работы: изучить алгоритмы отсечения отрезков [1].

1. Теоретическая часть

1.1. Постановка задачи

Отсечение отрезков – процесс определения видимой части отрезка относительно заданной области (окна отсечения). Отрезки могут быть: полностью видимые, полностью невидимые, частично видимые.

1.2. Изучаемые алгоритмы

1.2.1. Алгоритм Коэна-Сазерленда

Принцип работы:

- Пространство делится на 9 областей с помощью 4-битного кода.
- Каждый бит кода указывает положение точки относительно окна:
  - 1 бит – левее окна;
  - 2 бит – правее окна;
  - 3 бит – ниже окна;
  - 4 бит – выше окна.

Логика работы:

- Если коды обоих концов = 0000 => отрезок полностью видим.
- Если побитовое И кодов ≠ 0 => отрезок полностью невидим.
- Иначе => отрезок частично видим, требуется вычисление пересечений.

1.2.2. FC-алгоритм

Принцип работы:

- Кодирует весь отрезок целиком.
- 81 возможный вариант расположения отрезка.

- Быстрое определение видимости без вычисления пересечений для большинства случаев.

Логика работы:

- Пространство делится на 9 областей, каждой присваивается код от 1 до 9:
  - 1 (1001) 2 (1000) 3 (1010) – верхний ряд;
  - 4 (0001) 5 (0000) 6 (0010) – центральный ряд;
  - 7 (0101) 8 (0100) 9 (0110) – нижний ряд

1.2.3. Алгоритм Кируса-Бека

Принцип работы:

- Используется параметрическое представление отрезков.
- Работает с произвольными выпуклыми многоугольниками.
- Основан на анализе скалярных произведений с нормалями.

Преимущества:

- Работает с произвольными выпуклыми областями.
- Более точный и универсальный.
- Эффективен для сложных окон отсечения.

2. Практическая часть

2.1. Структура программы

Программа реализует визуализацию обоих алгоритмов отсечения с возможностью интерактивного создания отрезков и окон отсечения.

2.2. Реализованные функции

2.2.2. cohenSutherlandClip() – реализация алгоритма Коэна-Сазерленда

2.2.2. cyrusBeckClip() – реализация алгоритма Кируса-Бека

Задание:

1. Постройте окно отсечения.
2. Постройте отрезок, пересекающий окно отсечения.
3. Произведите отсечение отрезка с помощью любого алгоритма.
4. Выделите другим цветом отсекаемую часть отрезка.
5. Повторите п.п. 2 и 3 для полностью видимых и невидимых отрезков.
6. Сравните эффективность различных алгоритмов отсечения.
7. Чем отличаются алгоритмы Коэна-Сазерленда и FC-алгоритм от алгоритма Кируса-Бека?

Результат выполнения программы:

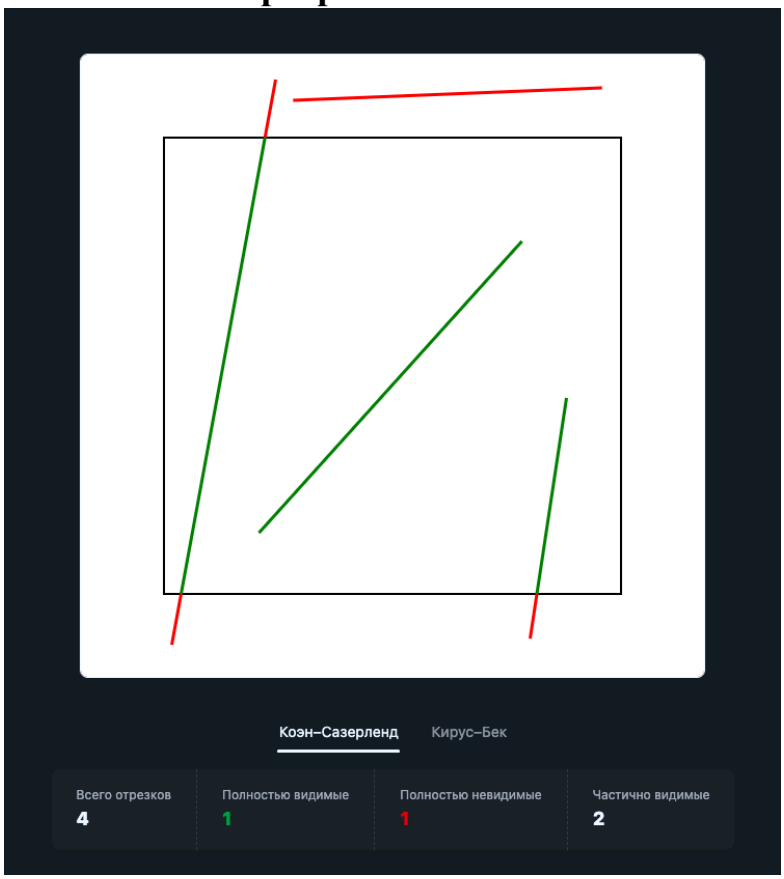

Рисунок 1 Алгоритм Коэна-Сазерленда

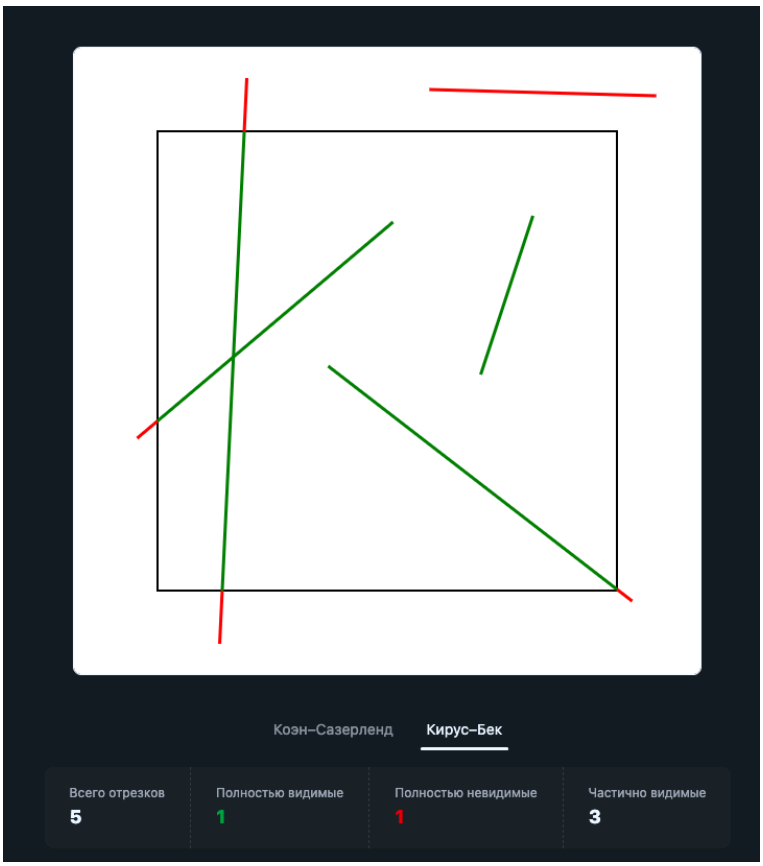

Рисунок 2 Алгоритм Кируса-Бека

3. Сравнительный анализ алгоритмов

3.1. Алгоритм Коэна-Сазерленда

Преимущества:

- Простая реализация.
- Быстрая отбраковка невидимых отрезков.
- Эффективен для прямоугольных окон.

Недостатки:

- Требует вычислений пересечений для частично видимых линий.

3.2. FC-алгоритм

Преимущества:

- Максимальная скорость для большинства случаев.
- Минимальное количество вычислений.
- Предварительная классификация всех вариантов.

Недостатки:

- Сложность реализации из-за большого количества случаев.
- Ограничен прямоугольными окнами.

3.3. Алгоритм Кируса-Бека

Преимущества:

- Универсальность (работает с любыми выпуклыми многоугольниками).
- Высокая точность вычислений.

Недостатки:

- Вычислительная сложность.
- Требует проверки выпуклости многоугольника.

Вывод

В ходе выполнения лабораторной работы были успешно изучены и реализованы три алгоритма отсечения отрезков: алгоритм Коэна-Сазерленда, FC-алгоритм и алгоритм Кирса-Бека. Проведённое исследование позволило провести сравнительный анализ их эффективности и определить оптимальные области применения каждого из алгоритмов.

FC-алгоритм продемонстрировал наивысшую производительность при работе с прямоугольными окнами отсечения. Его преимущество заключается в использовании предварительно классифицированных кодов областей, что позволяет мгновенно определять видимость отрезков для большинства случаев без выполнения вычислительно сложных операций определения пересечений.

Алгоритм Коэна-Сазерленда показал себя как сбалансированное решение, сочетающее относительную простоту реализации с достаточно высокой эффективностью. Его логика работы, основанная на поэтапном анализе кодов концевых точек отрезка, обеспечивает надёжное определение видимости при умеренных вычислительных затратах.

Алгоритм Кируса-Бека подтвердил свою уникальность при работе с произвольными выпуклыми многоугольниками в качестве окон отсечения. Несмотря на более высокую вычислительную сложность, связанную с использованием параметрического представления и векторных операций, данный алгоритм незаменим в случаях, когда требуется отсечение относительно сложных геометрических областей.

Выбор конкретного алгоритма для практического применения должен осуществляться на основе комплексного анализа требований задач. Для прямоугольных окон отсечения приоритет следует отдавать FC-алгоритму, для задач, требующих простоты реализации – алгоритму Коэна-Сазерленда, а для работы со сложными выпуклыми областями – алгоритму Кируса-Бека.

ПРИЛОЖЕНИЕ

```typescript
function cohenSutherlandClip(line: LineSegment, window: ClipWindow): ClippedLineResult {
    const INSIDE = 0;
    const LEFT = 1;
    const RIGHT = 2;
    const BOTTOM = 4;
    const TOP = 8;

    function computeOutCode(x: number, y: number): number {
        let code = INSIDE;
        if (x < window.xmin) code |= LEFT;
        else if (x > window.xmax) code |= RIGHT;
        if (y < window.ymin) code |= BOTTOM;
        else if (y > window.ymax) code |= TOP;
        return code;
    }

    let x0 = line.ax;
    let y0 = line.ay;
    let x1 = line.bx;
    let y1 = line.by;

    let outCode0 = computeOutCode(x0, y0);
    let outCode1 = computeOutCode(x1, y1);
    let accept = false;

    while (true) {
        if (!(outCode0 | outCode1)) {
            accept = true;
            break;
        }
        if (outCode0 & outCode1) {
            break;
        }

        const outCodeOut = outCode0 ? outCode0 : outCode1;
        let x = 0;
        let y = 0;

        if (outCodeOut & TOP) {
            x = x0 + (x1 - x0) * (window.ymax - y0) / (y1 - y0);
            y = window.ymax;
        }
        else if (outCodeOut & BOTTOM) {
            x = x0 + (x1 - x0) * (window.ymin - y0) / (y1 - y0);
            y = window.ymin;
        }
        else if (outCodeOut & RIGHT) {
            y = y0 + (y1 - y0) * (window.xmax - x0) / (x1 - x0);
            x = window.xmax;
        }
        else if (outCodeOut & LEFT) {
            y = y0 + (y1 - y0) * (window.xmin - x0) / (x1 - x0);
            x = window.xmin;
        }

        if (outCodeOut === outCode0) {
            x0 = x;
            y0 = y;
            outCode0 = computeOutCode(x0, y0);
        }
        else {
            x1 = x;
            y1 = y;
            outCode1 = computeOutCode(x1, y1);
        }
    }

    const original: LineSegment = { ax: line.ax, ay: line.ay, bx: line.bx, by: line.by };

    if (!accept) {
        return {
            original,
            visibleParts: [],
            invisibleParts: [original],
        };
    }

    const clipped: LineSegment = { ax: x0, ay: y0, bx: x1, by: y1 };

    const invisibleParts: LineSegment[] = [];
    if (x0 !== line.ax || y0 !== line.ay) {
        invisibleParts.push({ ax: line.ax, ay: line.ay, bx: x0, by: y0 });
    }
    if (x1 !== line.bx || y1 !== line.by) {
        invisibleParts.push({ ax: x1, ay: y1, bx: line.bx, by: line.by });
    }

    return {
        original,
        visibleParts: [clipped],
        invisibleParts,
    };
}
```

```typescript
function cyrusBeckClip(line: LineSegment, window: ClipWindow): ClippedLineResult {
    const x0 = line.ax;
    const y0 = line.ay;
    const x1 = line.bx;
    const y1 = line.by;

    const dx = x1 - x0;
    const dy = y1 - y0;

    let t0 = 0;
    let t1 = 1;

    function clipTest(p: number, q: number): boolean {
        if (p === 0) {
            return q >= 0;
        }
        const r = q / p;
        if (p < 0) {
            if (r > t1) return false;
            if (r > t0) t0 = r;
        }
        else if (p > 0) {
            if (r < t0) return false;
            if (r < t1) t1 = r;
        }
        return true;
    }

    if (
        clipTest(-dx, x0 - window.xmin)
        && clipTest(dx, window.xmax - x0)
        && clipTest(-dy, y0 - window.ymin)
        && clipTest(dy, window.ymax - y0)
    ) {
        const nx0 = x0 + t0 * dx;
        const ny0 = y0 + t0 * dy;
        const nx1 = x0 + t1 * dx;
        const ny1 = y0 + t1 * dy;

        const visible: LineSegment = { ax: nx0, ay: ny0, bx: nx1, by: ny1 };
        const original: LineSegment = { ax: line.ax, ay: line.ay, bx: line.bx, by: line.by };
        const invisibleParts: LineSegment[] = [];

        if (t0 > 0) {
            invisibleParts.push({ ax: line.ax, ay: line.ay, bx: nx0, by: ny0 });
        }
        if (t1 < 1) {
            invisibleParts.push({ ax: nx1, ay: ny1, bx: line.bx, by: line.by });
        }

        return {
            original,
            visibleParts: [visible],
            invisibleParts,
        };
    }

    return {
        original: { ax: line.ax, ay: line.ay, bx: line.bx, by: line.by },
        visibleParts: [],
        invisibleParts: [
            { ax: line.ax, ay: line.ay, bx: line.bx, by: line.by },
        ],
    };
}
```