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 №9
по дисциплине «Компьютерная графика»

Тема: Реализация алгоритма отсечения произвольного многоугольника выпуклым отсекателем (Алгоритм Сазерленда-Ходжмена).

Студент: Блохин Дмитрий

Группа: ИУ7-42Б

Москва.
2020 г.

Цель работы: изучение и программная реализация алгоритма произвольного многоугольника выпуклым отсекателем (Алгоритм Сазерленда-Ходжмена).

Задание:

-Должна быть разработана программа, позволяющая осуществлять ввод отсекателя, отсекаемого многоугольника и выполнять отсечение многоугольника по границам отсекателя.
-Необходимо обеспечить ввод отсекателя – произвольного многоугольника. Высветить его первым цветом. Также необходимо обеспечить ввод отсекаемого многоугольника (высветить вторым цветом). Должна присутствовать проверка отсекателя на выпуклость. Должен быть предусмотрен ввод вершин многоугольника в произвольных точках ребер отсекателя (включая его вершины)
-Ввод осуществлять с помощью мыши и нажатия других клавиш.
-Выполнить отсечение многоугольника, показав результат третьим цветом. Исходный многоугольник не удалять.

Ход работы:

Алгоритм Сазерленда-Ходжмена позволяет провести отсечение произвольного многоугольника по границам выпуклого отсекателя. Идея алгоритма достаточно проста. На каждом шаге отсечения исходный и промежуточные многоугольники отсекаются последовательно очередной границей отсекателя.

Алгоритм Сазерленда - Ходжмена(алгоритма отсечения произвольного многоугольника выпуклым отсекателем):

Q - массив вершин результирующего многоугольника

1. Ввод исходных данных:
   P - массив вершин исходного многоугольника,
   W - массив вершин отсекателя.
   Для удобства работы алгоритма первая вершина отсекателя заносится в массив W дважды: на первое место и еще раз в конец массива (это сделано потому, что последнее ребро отсекателя образуется последней и первой вершинами многоугольника),
   NP - число вершин исходного многоугольника
   NW - число вершин отсекателя единица

Вершины всех многоугольников перечисляются по часовой стрелке для каждой стороны отсекателя выполнить

2. for i = 1 to NW-1

2.1 установить счетчик вершин результата и обнулить результат NQ=0, Q=0

отсечь каждое ребро многоугольника по данной стороне отсекателя

2.2 for j = 1 to NP

2.2.1 if j <> 1 then 1

Запомнить первую вершину F = Pj

go to 2

Проверить факт пересечения ребром многоугольника стороны отсекателя

call Факт-сеч(S, Pj, Wi, Wi + 1; Признак)

if Признак = нет then 2

Если ребро пересекает сторону отсекателя, вычислить точку пересечения

call Пересечение(S, Pj, Wi, Wi + 1; T сечнения)

Занести точку пересечения в результат

call Выход(Tсечения,NQ,Q)

Изменить начальную точку ребра многоугольника S = Pj

Проверить видимость конечной точки (теперь это S) ребра многоугольника

call Видимость(S, Wi, Wi + 1; S видимость)

if Sвидимость < 0 then 3

Если точка видима то занести ее в результат call Выход(Tсечения,NQ,Q)

3. next j

Обработать замыкающее ребро многоугольника если результат пуст, то перейти к следующей стороне отсекателя

Реализация:

1.Для начала приведу код проверки на выпуклость многоугольника.

Используемые функции для реализации проверки на выпуклость многоугольника:

```
int sign(int x) {
    if (x < 0)
        return -1;

    if (x > 0)
        return 1;

    return 0;
}

int skewProduct(const QPoint &a, const QPoint &b) {
    return a.x() * b.y() - a.y() * b.x();
}

int direction(const QPoint &prev, const QPoint &curr, const QPoint &next) {
    return sign(skewProduct(curr - prev, next - curr));
}
```

Реализация проверки на выпуклость многоугольника

```
int MainWindow::checkConvex(QVector<QPoint> &clipper_vertices, int k) {
    int f = 1;
    int curr_direction = 0;
    QPoint prev_vertex = clipper_vertices.back();
    QPoint curr_vertex = clipper_vertices[0];
    QPoint next_vertex = clipper_vertices[1];
    int prev_direction = direction(prev_vertex, curr_vertex, next_vertex);

    for (int i = 1; i < k && f; ++i)
    {
        prev_vertex = curr_vertex;
        curr_vertex = next_vertex;
        next_vertex = clipper_vertices[(i + 1) % clipper_vertices.size()];
        curr_direction = direction(prev_vertex, curr_vertex, next_vertex);
        if (curr_direction != prev_direction)
            f = 0;
        prev_direction = curr_direction;
    }
    return  f * curr_direction;
}
```

2. Реализация алгоритма Сазерленда - Ходжмена.

Используемые функции для реализации алгоритма Сазерленда - Ходжмена:

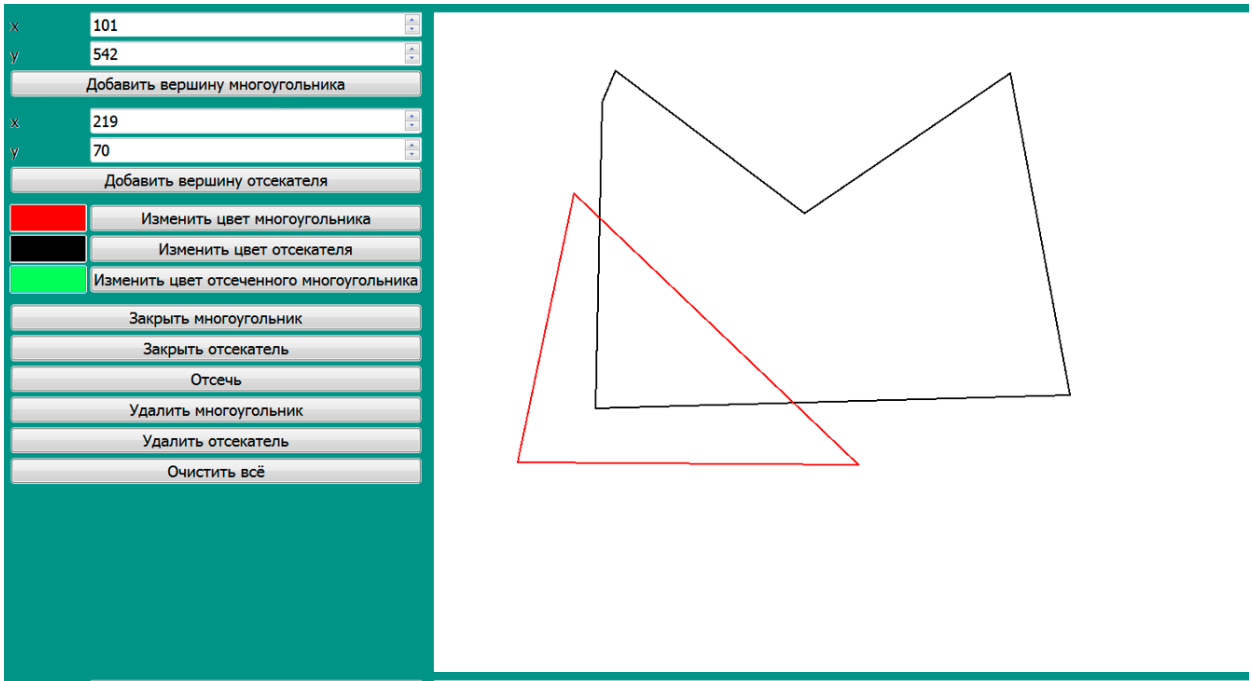
```
int sign(int x) {
    if (x < 0)
        return -1;

    if (x > 0)
        return 1;

    return 0;
}

int skewProduct(const QPoint &a, const QPoint &b) {
    return a.x() * b.y() - a.y() * b.x();
}

bool MainWindow::checkIntersection(const QPoint &sp, const QPoint &ep, const QPoint &p0, const QPoint &p1) {
    return isVisible(sp, p0, p1) * isVisible(ep, p0, p1) <= 0;
}

int MainWindow::isVisible(const QPoint &p, const QPoint &p1, const QPoint &p2) {
    return sign(skewProduct(p - p1, p2 - p1));
}

QPoint MainWindow::intersection(QPoint &p1, QPoint &p2, QPoint &cp1, QPoint &cp2) {
    const int det = skewProduct(p2 - p1, cp1 - cp2);
    const double t = double(skewProduct(cp1 - p1, cp1 - cp2)) / det;
    return p1 + (p2 - p1) * t;
}
```

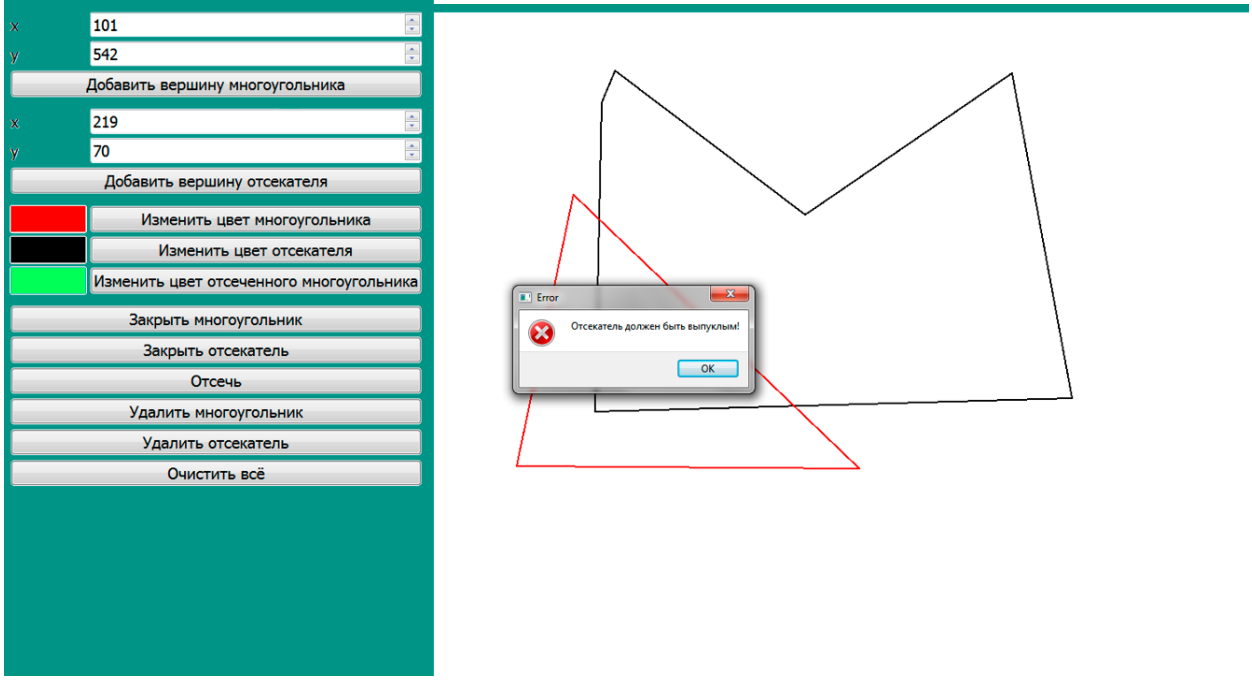
Реализация алгоритма Сазерленда-Ходжмена

```
QVector<QPoint> MainWindow::Sutherland_Hodgman(QVector<QPoint>& polygonVertices, QVector<QPoint>& clipperVertices, int direction) {
    QVector<QPoint> result;
    QVector<QPoint> polygon = polygonVertices;
    QVector<QPoint> clipper = clipperVertices;
    clipper.push_back(clipper.front());
    QPoint first, start;

    // цикл для ребер отсекателя
    for (int i = 0; i < clipper.size() - 1; ++i)
    {
        // цикл для вершин отсекаемого многоугольника
        for (int j = 0; j < polygon.size(); ++j) {
            if (!j)
                first = polygon[j];
            else if (checkIntersection(start, polygon[j], clipper[i], clipper[i + 1]))
                result.push_back(intersection(start, polygon[j], clipper[i], clipper[i + 1]));

            start = polygon[j];
            if (isVisible(start, clipper[i], clipper[i + 1]) * direction < 0)
                result.push_back(start);
        }

        if (!result.empty())
        {
            if (checkIntersection(start, first, clipper[i], clipper[i + 1]))
                result.push_back(intersection(start, first, clipper[i], clipper[i + 1]));
        }

        polygon = result;
        result.clear();
    }

    return polygon;
}
```

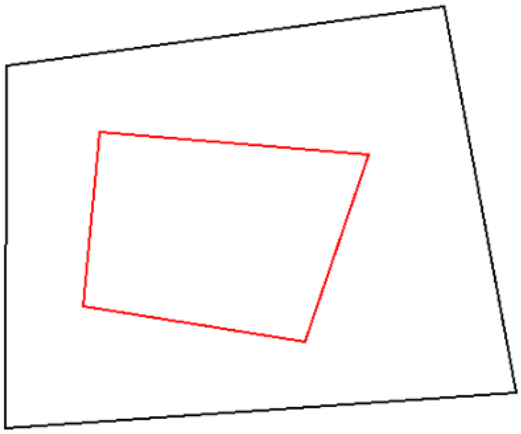
Демонстрация работы программы:

Проверка выпуклости многоугольника:

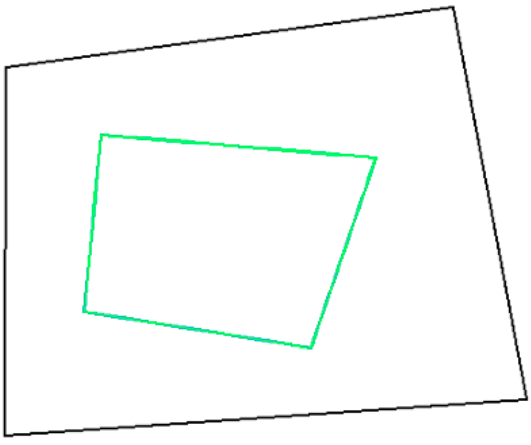




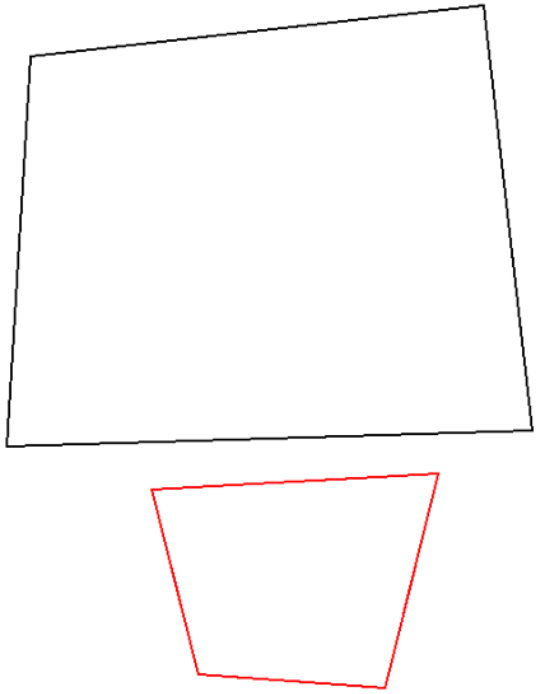
До отсечения:



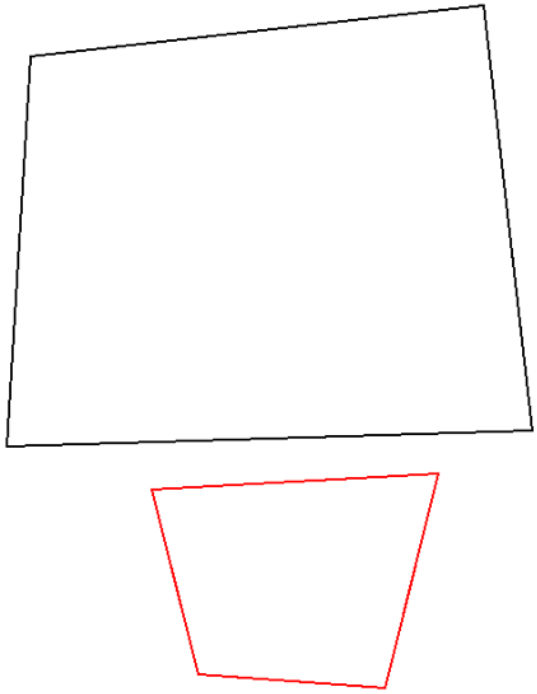
После:



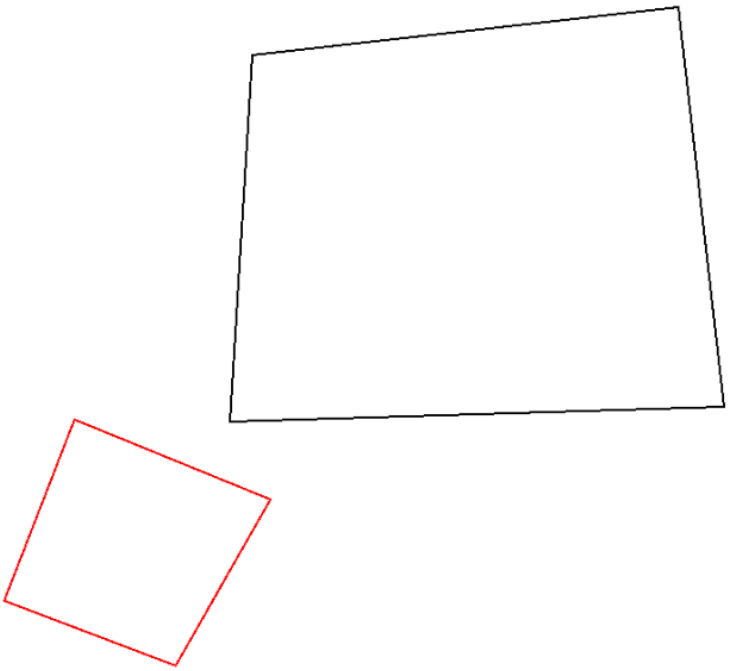
До отсечения:



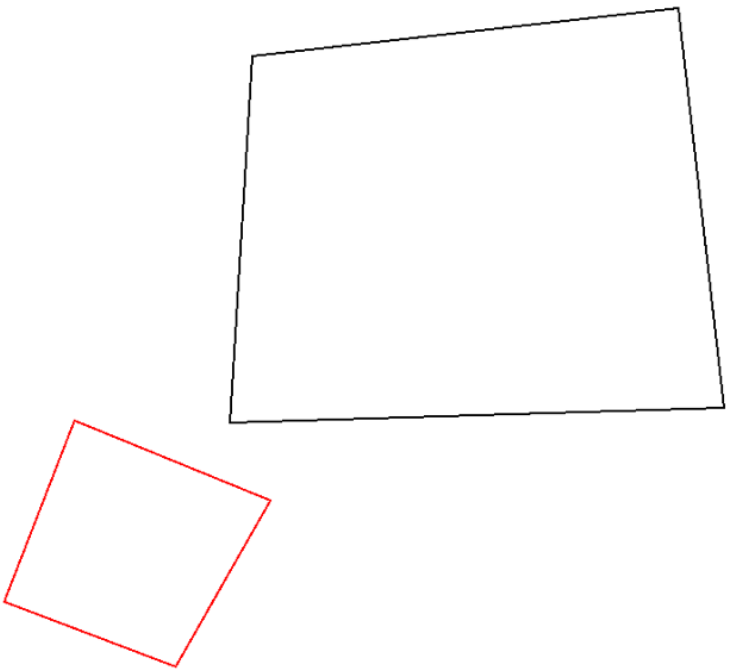
После:



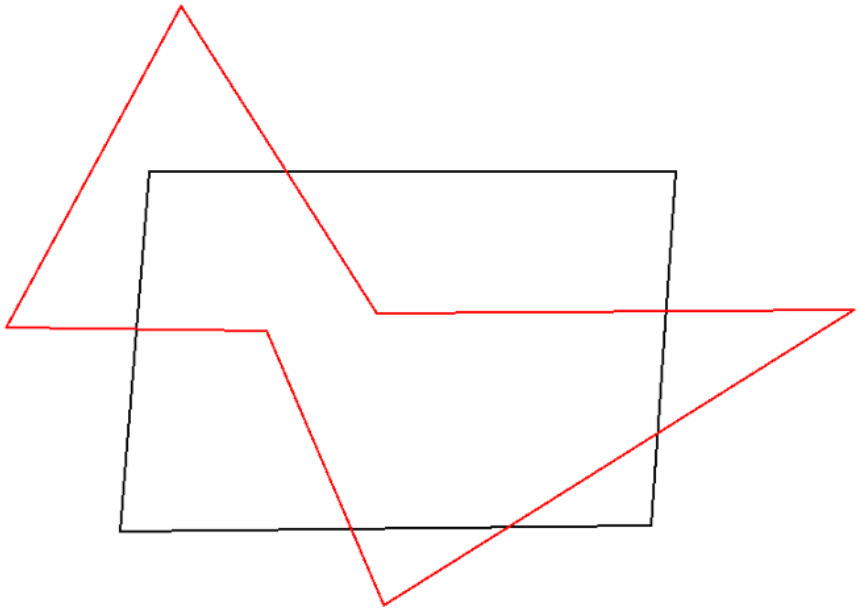
До отсечения:



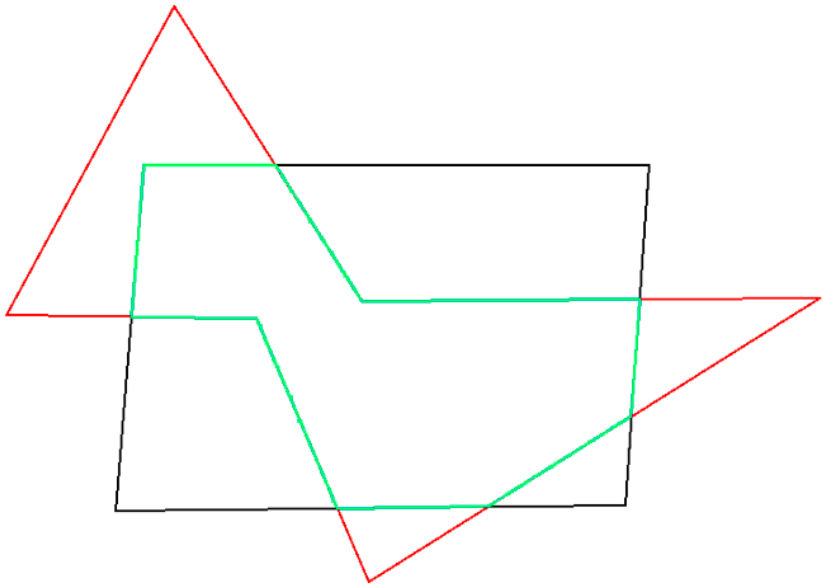
После:



До отсечения:

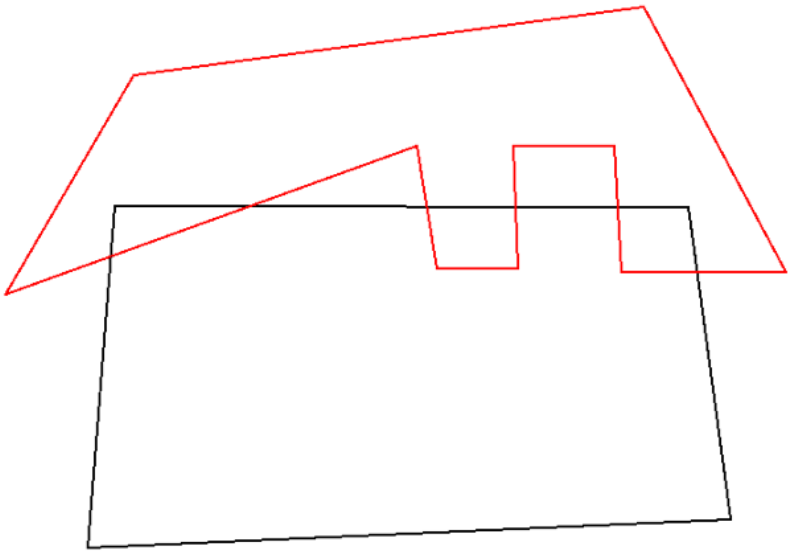


После:



Пример с появлением ложных ребер

До отсечения:



После: