|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 № 8**

|  |  |
| --- | --- |
| **Тема «Реализация алгоритма отсечения отрезка произвольным выпуклым отсекателем.(Алгоритм Кируса-Бека)»**  **Дисциплина Компьютерная графика**  **Студент Кузин Антон**  **Группа ИУ7-42Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель Куров А.В.** |  |

Москва.

2020 г.**Цель:**

Изучение и программная реализация алгоритма отсечения отрезка.

**Техническое задание:**

Необходимо обеспечить ввод отсекателя – произвольного многоугольника. Высветить его первым цветом. Также необходимо обеспечить ввод нескольких (до десяти) различных отрезков (высветить их вторым цветом). Отрезки могут иметь произвольное расположение: горизонтальные, вертикальные, имеющие произвольный наклон.

Предусмотреть ввод отрезков, параллельных границе отсекателя.

Ввод осуществлять с помощью мыши и нажатия других клавиш.

Выполнить отсечение отрезков, показав результат третьим цветом. Исходные отрезки не удалять.

**Теоретические сведения:**

В этом алгоритме для определения местоположения точки относительно окна отсечения используется вектор нормали и параметрическая форма задания отрезка. Также для корректной работы алгоритма необходимо, чтобы отсекатель был выпуклым. Чтобы определить, является ли отсекатель выпуклым, анализируются знаки векторных произведений смежных сторон многоугольника, в результате чего возможны такие случаи:

* Знаки всех векторных произведений равны нулю – отсекатель вырождается в прямую.
* Если есть положительные и отрицательные знаки – многоугольник невыпуклый.
* Все знаки неотрицательные (неположительные ) – многоугольник выпуклый, а внутренние нормали направлены влево (вправо) от его контура.

Для определения видимости отрезка производится поиск значения параметра t из уравнения

t = -(Wi nвi) / (D nвi), D ≠0,

где nвi – внутренняя нормаль к i-ому ребру отсекателя, Wi = P1 – fi, P1 – точка отрезка, а fi – точка, лежащая на i-ом ребре отсекателя; D=P2 – P1 – вектор ориентации отрезка.

Если значения t не принадлежат интервалу 0 ≤ t ≤ 1, то их не рассматривают, поскольку они соответствуют точкам, лежащим вне исходного отрезка. Так как можно получить более 2-х решений для t, удовлетворяющих условию, их разбивают на две группы: верхнюю и нижнюю, в зависимости от близости найденной точки пересечения к началу или концу отрезка. Чтобы отрезок был видимым относительно всего отсекателя, он должен быть видим относительно всех ребер отсекателя одновременно. Концам видимой части отрезка будут соответствовать два значения параметра t, одно из которых является максимальным значением из нижней группы tmaxmin, а второе - минимальным из верхней группы tminmax. Найденное значение параметра t для очередной точки пересечения рассматривают в качестве возможного верхнего предела tв, если знаменатель Dnвi < 0; в случае же, когда знаменатель положителен, значение параметра t определяет точку, которую относят к нижней границе видимости tн.

**Практическая часть:**

float crossProduct(const QPointF &v1, const QPointF &v2)

{

return v1.x() \* v2.y() - v2.x() \* v1.y();

}

float dotProduct(const QPointF &v1, const QPointF &v2)

{

return v1.x() \* v2.x() + v1.y() \* v2.y();

}

QPointF perpendicular(const QPointF &vector)

{

return QPointF(-vector.y(), vector.x());

}

void clip(const QLineF &line, const QVector<QLine> &clipper, int direction, QPainter &painter)

{

QPointF d(line.p2() - line.p1());

float tl = 0;

float th = 1;

for (auto edge: clipper)

{

QPointF n = perpendicular(direction \* (edge.p2() - edge.p1()));

QPointF w = line.p1() - edge.p1();

float wScalar = dotProduct(w, n);

float dScalar = dotProduct(d, n);

if (dScalar == 0){

if (wScalar < 0)

return;

} else {

float t = - wScalar / dScalar;

if (dScalar > 0){

if (t > 1)

return;

tl = qMax(tl, t);

}

else{

if (t < 0)

return;

th = qMin(th, t);

}

}

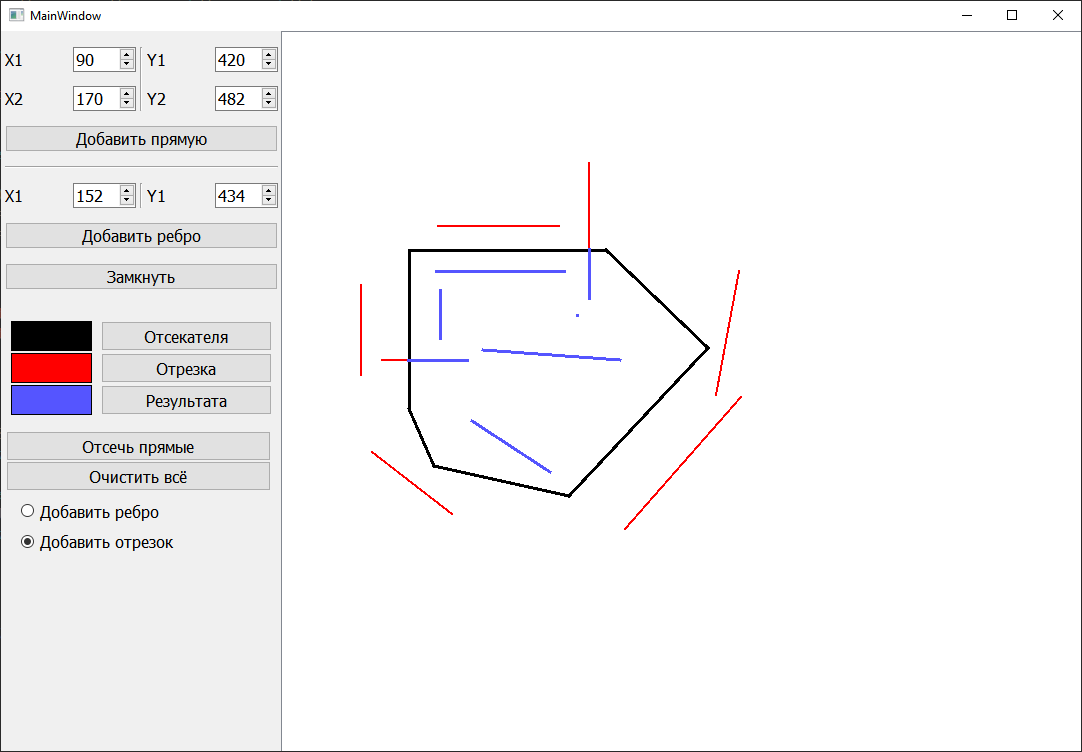
}

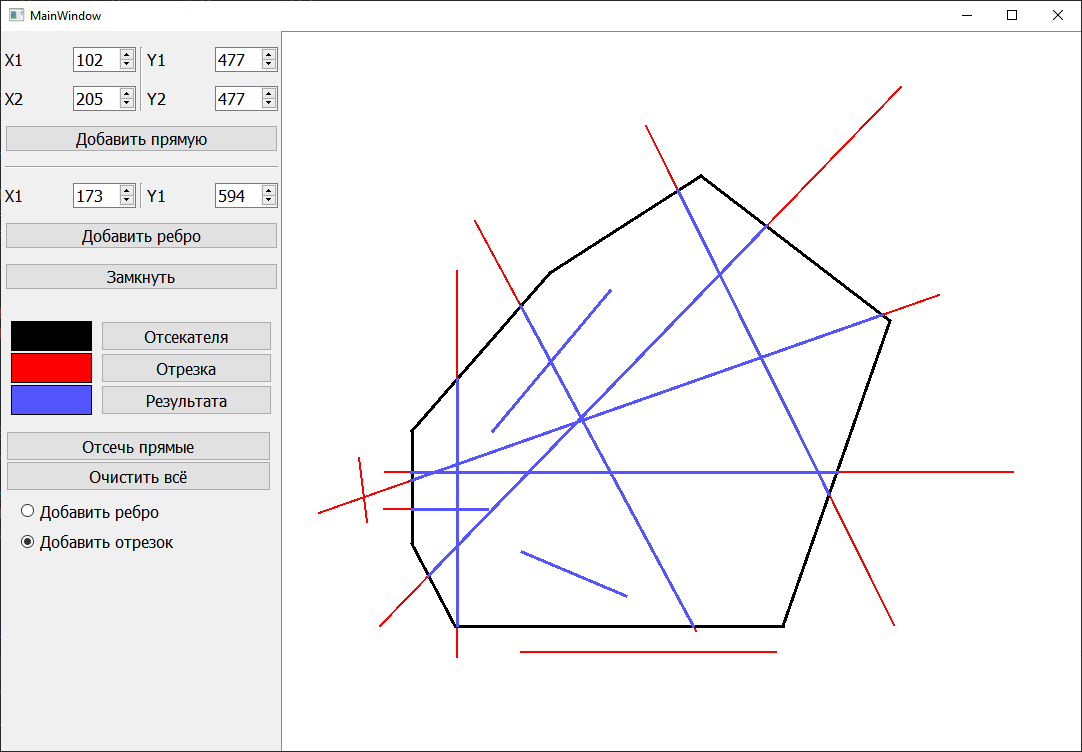
if (tl <= th)

painter.drawLine(line.p1() + d \* tl, line.p1() + d \* th);

}

**Пример работы:**





**Вывод**

В ходе выполнения лабораторной работы был реализован и изучен алгоритм Кируса-Бека и проверена его работа.