|  |  |
| --- | --- |
| 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 № 7**

|  |  |
| --- | --- |
| **Тема «Реализация алгоритма отсечения отрезка регулярным отсекателем»**  **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:**

Алгоритм отсечения (простой, Сазерленда-Коэна, деления отрезка пополам) выбирается в соответствии с номером студента в списке группы.

Необходимо обеспечить ввод регулярного отсекателя - прямоугольника. 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.

Ввод осуществлять с помощью мыши и нажатия других клавиш.

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

**Теоретические сведения:**

Отсечение - это операция удаления изображения за пределами выделенной области, называемой окном. При отсечении на плоскости в качестве стандартного отсекателя рассматривается прямоугольник со сторонами, параллельными осям координат. Помимо отсекателя должны быть также известны геометрические характеристики изображенных объектов. В результате отсечения должны получиться геометрические характеристики объектов, остающихся в пределах окна отсечения в результате выполнения рассматриваемой операции.

**Простой алгоритм определения видимости**

В ходе выполнения данного алгоритма, сначала проверяется полная видимость или невидимость отрезка. Для этого формируются коды концов отрезка – 4-х битные коды, биты которых заполняются по следующему правилу:

T1 =1, если точка лежит левее окна, и 0 в противном случае;

T2 =1, если точка лежит правее окна, и 0 в противном случае;

T3 =1, если точка лежит ниже окна, и 0 в противном случае;

T4 =1, если точка лежит выше окна, и 0 в противном случае.

Точка лежит внутри отсекателя, если все биты ее кода - нулевые. Если побитное логическое произведение кодов концов отрезка

не равно нулю, то отрезок является полностью невидимым и его можно отбросить.

Если отрезок не является полностью видимым, то далее ищутся точки пересечения отрезка с границами отсекателя. При этом сначала анализируется возможность пересечения отрезка с очередной границей отсекателя. Если пересечение возможно, то находятся координаты точки пересечения, после чего проводится анализ ее корректности. Под корректностью понимается принадлежность найденной точки ребру отсекателя, а не его продолжению. В случае корректности найденного пересечения точка заносится в результат и ищется вторая точка пересечения отрезка с границами отсекателя. В противном случае, происходит проверяется пересечение со следующей стороной отсекателя. Если не было найдено ни одного корректного пересечения, то отрезок является невидимым.

**Практическая часть:**

int setupCodes(int xl, int xr, int yb, int yt, const QPoint &point)

{

int result = 0;

if (point.x() < xl)

result |= 1;

if (point.x() > xr)

result |= 1 << 1;

if (point.y() < yb)

result |= 1 << 2;

if (point.y() > yt)

result |= 1 << 3;

return result;

}

int find\_intersection(int xl, int xr, int yb, int yt, QLine &line, QPoint &p)

{

float m = 1e30;

float x, y;

if (line.x2() - line.x1() != 0){

m = (line.y2() - line.y1()) / (float)(line.x2() - line.x1());

if (xl >= p.x()){

y = m \* (xl - p.x()) + p.y();

if ((yt >= y) && (y >= yb)){

p.setX(xl);

p.setY(qRound(y));

return 0;

}

}

if (xr <= p.x()){

y = m \* (xr - p.x()) + p.y();

if ((yt >= y) && (y >= yb)){

p.setX(xr);

p.setY(qRound(y));

return 0;

}

}

if (m == 0)

return -2;

}

if (p.y() >= yt){

x = (1 / m) \* (yt - p.y()) + p.x();

if ((xl <= x) && (x <= xr)){

p.setX(qRound(x));

p.setY(yt);

return 0;

}

}

if (yb < p.y())

return -1;

x = (1 / m) \* (yb - p.y()) + p.x();

if ((xl <= x) && (x <= xr)){

p.setX(qRound(x));

p.setY(yb);

return 0;

}

return -1;

}

void cut\_off(int xl, int xr, int yb, int yt, QLine &line, canvas\_t &canvas)

{

int t1 = setupCodes(xl, xr, yb, yt, line.p1());

int t2 = setupCodes(xl, xr, yb, yt, line.p2());

int flag = 0;

if (!t1 && !t2){

draw\_line(canvas, line);

return;

}

if (t1 & t2)

return;

QPoint p;

QPoint res[2];

int number = 0;

if (!t1){

res[0] = line.p1();

p = line.p2();

number = 2;

flag = find\_intersection(xl, xr, yb, yt, line, p);

if (flag == -2)

return;

}

else if (!t2){

res[0] = line.p2();

p = line.p1();

number = 2;

flag = find\_intersection(xl, xr, yb, yt, line, p);

if (flag == -2)

return;

}

while (number <= 2){

if (number)

res[number - 1] = p;

number++;

if (number > 2)

break;

if (number == 1)

p = line.p1();

else

p = line.p2();

flag = find\_intersection(xl, xr, yb, yt, line, p);

if (flag == -2)

return;

else if (flag == -1)

break;

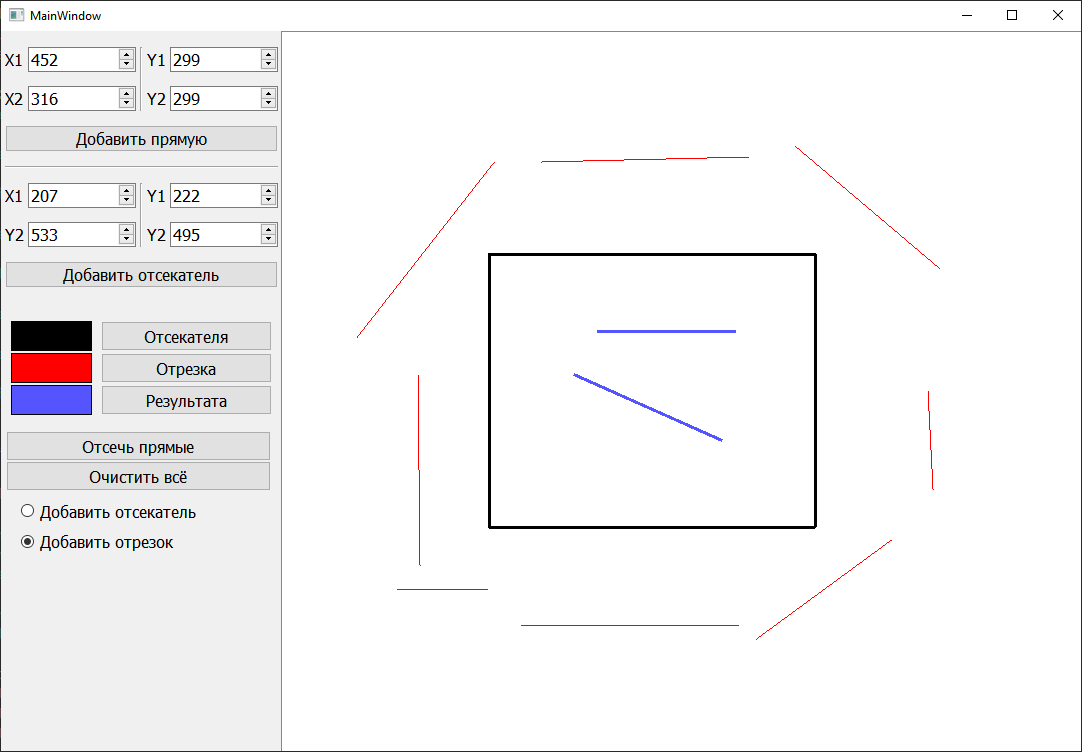
}

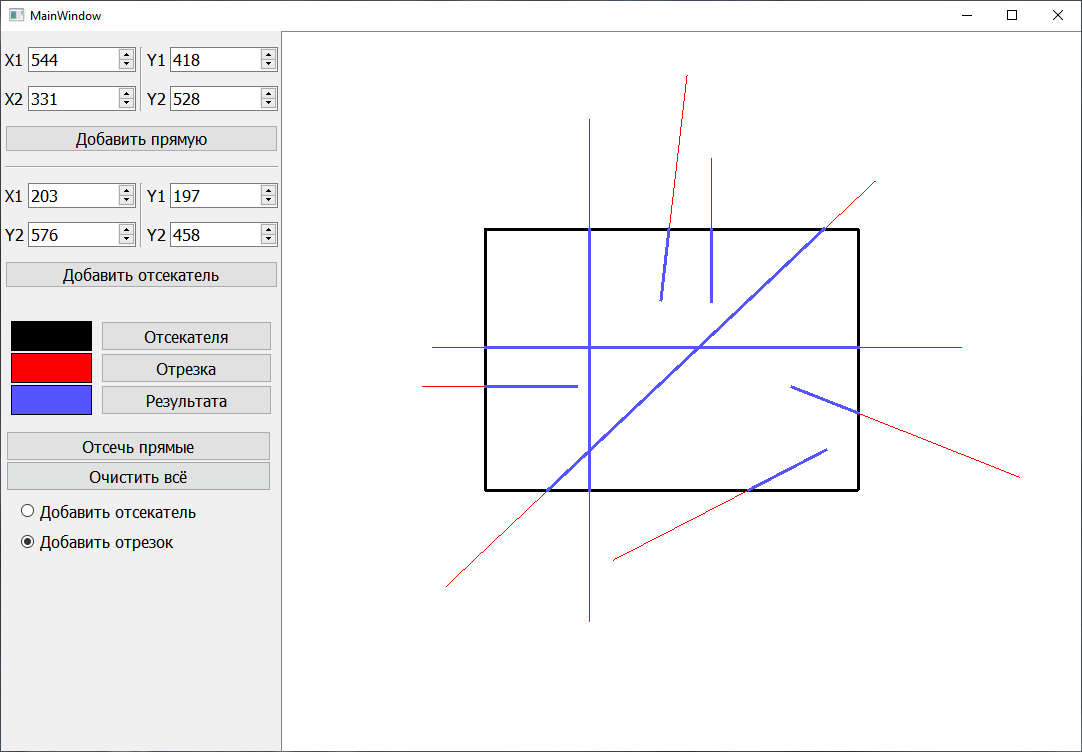
if (!flag)

draw\_line(canvas, QLine(res[0], res[1]));

}

**Пример работы:**





**Вывод**

Был реализован простой алгоритм отсечения отрезка и проверена его работа.