|  |  |
| --- | --- |
| 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 № 9**

|  |  |
| --- | --- |
| **Дисциплина Компьютерная графика**  **Тема Программная реализация алгоритма отсечения произвольного многоугольника произвольным выпуклым отсекателем**  **Студент Иванов В.А.**  **Группа ИУ7-42Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель Куров А.В.** |  |

Москва.

2020 г.

**Цель работы**

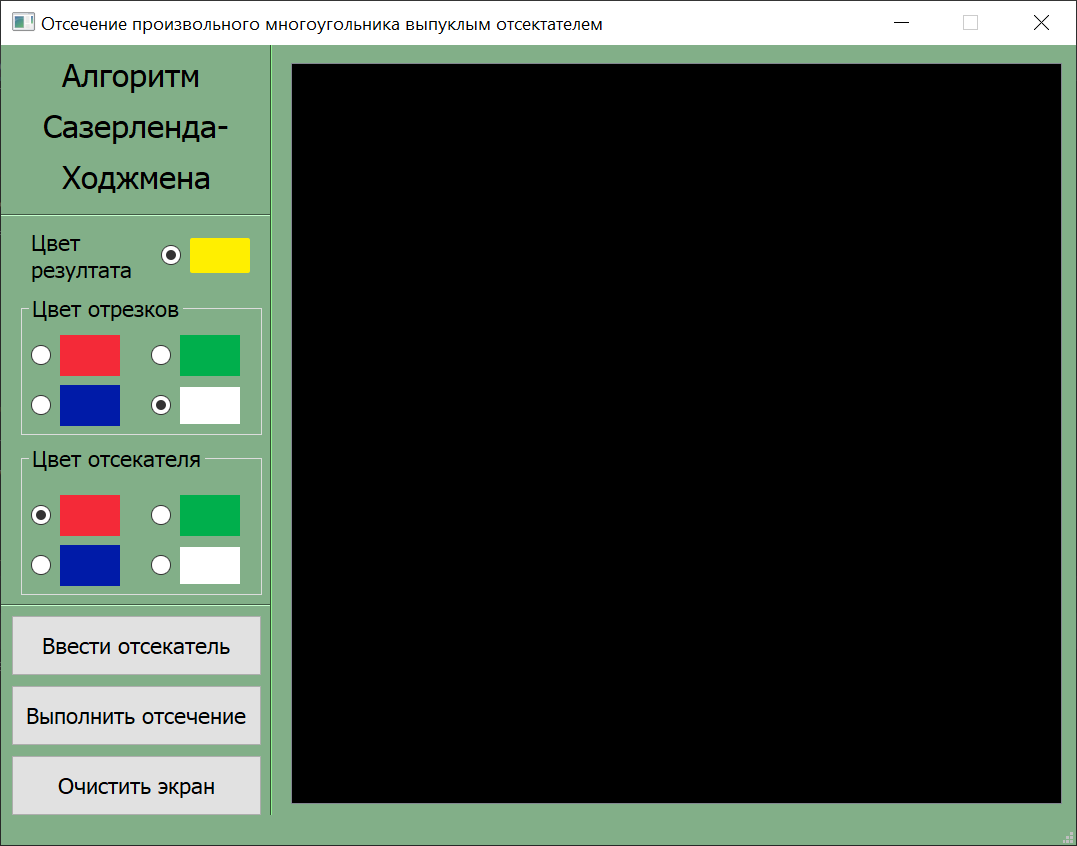
Реализация алгоритма Сазерленда-Ходжмена.

**Описание задания**

1. Реализовать алгоритм Сазерленда-Ходжмена
2. Обеспечить возможность ввода отсекателя и отсекаемой фигуры многоугольником с помощью мыши (в т.ч. и горизонтальных и вертикальных отрезков и рёбер, и вершин, лежащих на стороне отскателя или совпадающих с его вершинами)

**Описание графического интерфейса**

Для использования функционала поставленных задач, был создан графический интерфейс

****

Интерфейс предоставляет возможность:

* Выбора цвета рёбер и отсекателя
* Ввод отсекателя при помощи мыши
* Ввода фигуры при помощи мыши. Нажатие ЛКМ для ввода произвольного отрезка и ПКМ для ввода горизонтального/вертикального отрезка. Нажатие рядом с вершиной или стороной отсекателя скорректирует вводимую точку, чтобы она попала на отсекатель

**Описание и реализация алгоритма**

Задачей данной лабораторной работы служит выполнение отсечения произвольного многоугольника выпуклым отсекателем, используя алгоритм Сазерленда-Ходжмена.

Идея алгоритма состоит в последовательном отсечении фигуры каждой из сторон отсекателя. Получившийся в результате отсечения очередной стороной многоугольник служит объектом отсечения уже для следующей стороны. В итоге, получается многоугольник, все вершины которого являются видимыми относительно всех сторон отсекателя.

При отсечении стороной отсекателя, рассматриваются расположения рёбер многоугольника относительно этой стороны. Для удобства, в алгоритме рассматривается каждая вершина многоугольника P[i] (i = 1..n-1) как конечная точка ребра (p[i-1], p[i]), при этом вершина P[i] рассматривается отдельно. Возможно всего 4 случая расположения ребра относительно стороны отсекателя:

1. Точка P[i-1] невидима, P[i] невидима (полная невидимость)
2. Точка P[i-1] видима, P[i] видима (полная видимость)
3. Точка P[i-1] видима, P[i] невидима (т.е. отрезок “входит” в отсекатель)
4. Точка P[i-1] невидима, P[i] видима (т.е. отрезок “выходит” из отсекателя)

В список вершин результирующего многоугольника требуется занести только те точки из отрезка, которые являются видимыми. Поэтому в 1 и 3 случае точка P[i] не заносится в список, а в случаях 2 и 4 заносится. В случаях 3 и 4 также возникает точка пересечения со стороной отсекателя, её требуется найти и также занести в список, так как она является видимой.

Для первой же точки списка вершин многоугольника (i = 0) требуется определить только видимость, и на основании принять решение о её занесении в список. Также отдельно рассматривается ребро (P[0], P[n-1]), для этого первая вершина в начале обхода запоминается, и после завершения обхода ребро рассматривается отдельно, таким же образом.

В случае, если после очередного отсечения, список вершин оказался пустым, анализ фигуры можно прервать, она считается полностью невидимой.

Определение видимости точки производится за счёт анализа компоненты Z векторного произведения вектора стороны отсекателя на вектор между точкой на этой стороне и анализируемой точкой. В случае, если знак Z совпадёт со знаком, полученным при анализе выпуклости (векторное произведение соседних сторон), то можно сказать, что вектор до точки лежит по одну сторону с внутренней нормалью, что говорит о видимости точки. Точку пересечения можно определить с помощью решения системы для двух уравнений прямых, проходящих через отрезки. Определение факта пересечения стороны с отсекателем можно выполнить, сравнив видимость концов стороны. Пересечение будет в случае, если их видимость будет различной.

**Программная реализация:**

Отсечение многоугольника  
def sutherland\_hodgman\_cut(self):  
 cut\_dir = is\_convex(self.cut\_arr)  
 if not cut\_dir: return False  
   
 np = len(self.polygon\_arr)  
 p = list(point[0] for point in self.polygon\_arr)  
  
 nc = len(self.cut\_arr)  
 c = list(point[0] for point in self.cut\_arr)  
 c.append(c[0])  
  
 f, s = None, None  
 for i in range(nc):  
 nq = 0  
 q = list()  
 for j in range(np):  
 if j == 0:  
 f = p[0]  
 elif is\_cross((s, p[j]), (c[i], c[i+1])):  
 cross\_p = find\_cross((s, p[j]), (c[i], c[i+1]))  
 q.append(cross\_p)  
 nq += 1  
  
 s = p[j]  
 if is\_visible(s, (c[i], c[i+1])) \* cut\_dir >= 0:  
 q.append(s)  
 nq += 1  
  
 if nq != 0 and is\_cross((s, f), (c[i], c[i+1])):  
 cross\_p = find\_cross((s, f), (c[i], c[i + 1]))  
 q.append(cross\_p)  
 nq += 1  
 p = q  
 np = nq  
 if np < 2: break  
  
 for i in range(np):  
 self.draw\_line(p[i-1], p[i])  
 return True

Проверка выпуклости и поиск направления нормалей

def is\_convex(edge\_arr):  
 v1 = edge\_dir(\*edge\_arr[-1])  
 v2 = edge\_dir(\*edge\_arr[0])  
 last\_mlt = vector\_mlt(v2, v1)  
  
 for i in range(1, len(edge\_arr)):  
 v1 = edge\_dir(\*edge\_arr[i - 1])  
 v2 = edge\_dir(\*edge\_arr[i])  
  
 mlt = vector\_mlt(v2, v1)  
 if mlt \* last\_mlt <= 0:  
 return False  
 last\_mlt = mlt  
 return copysign(1, last\_mlt)

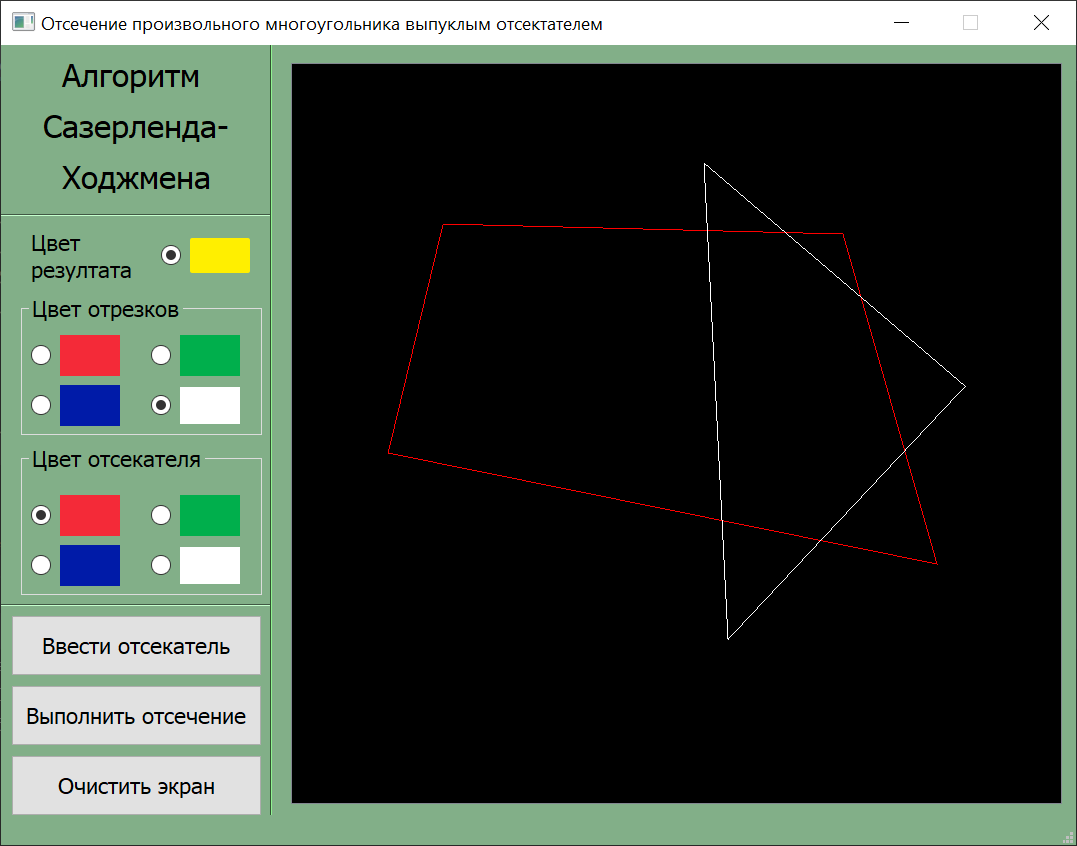
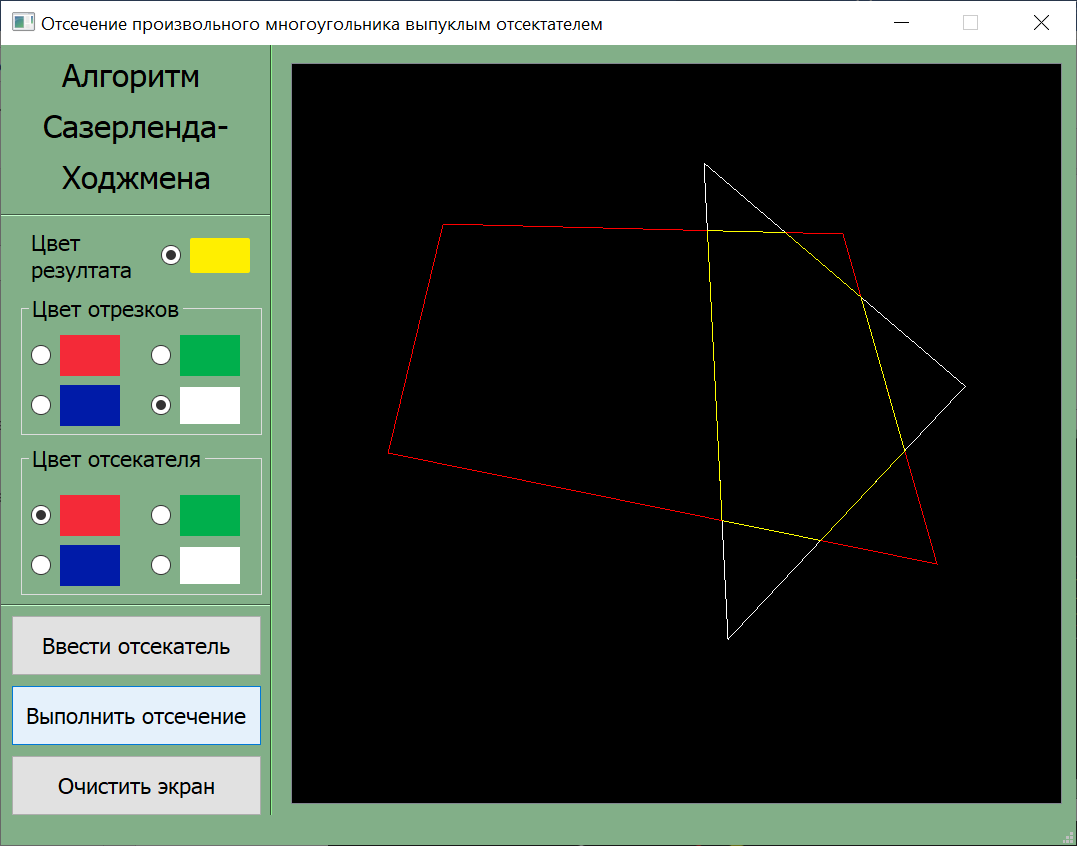
Функция работы с рёбрами и отрезками  
def edge\_dir(p1, p2):  
 x = p2[0] - p1[0]  
 y = p2[1] - p1[1]  
 return [x, y]  
def line\_equation(p1, p2):  
 A = p2[1] - p1[1]  
 B = p1[0] - p2[0]  
 C = -p1[0] \* A - p1[1] \* B  
 return A, B, C

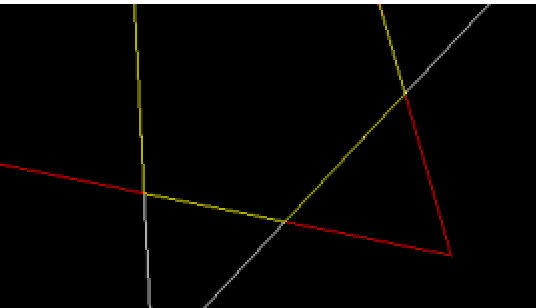
def scalar\_mlt(v1, v2):  
 return v1[0]\*v2[0] + v1[1]\*v2[1]  
def vector\_mlt(v1, v2):  
 return v1[0]\*v2[1] - v1[1]\*v2[0]

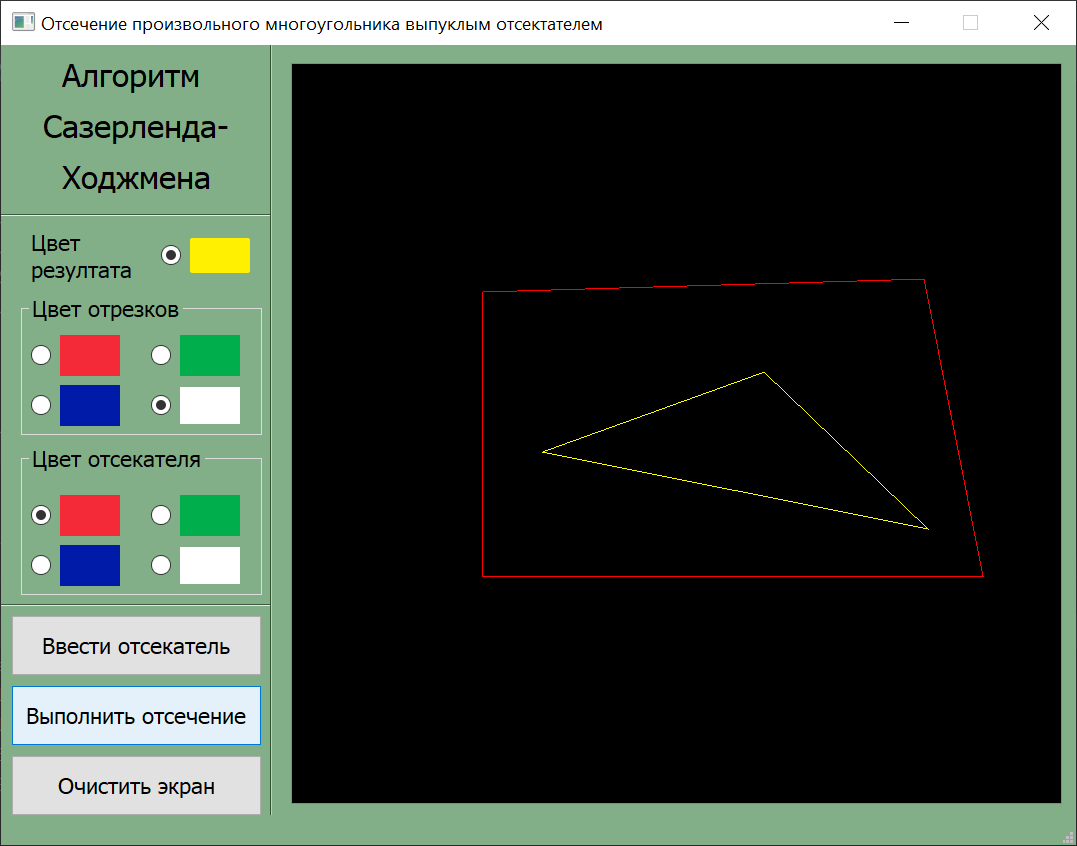
def is\_visible(point, cut):  
 temp1 = (point[0] - cut[0][0]) \* (cut[1][1] - cut[0][1])  
 temp2 = (point[1] - cut[0][1]) \* (cut[1][0] - cut[0][0])  
 return temp1 - temp2  
  
def is\_cross(seg, cut):  
 view0 = is\_visible(seg[0], cut)  
 view1 = is\_visible(seg[1], cut)  
 return view0\*view1 <= 0  
  
def matrix\_det(a, b, c, d):  
 return a\*d - b\*c  
def find\_cross(edge1, edge2):  
 A1, B1, C1 = line\_equation(\*edge1)  
 A2, B2, C2 = line\_equation(\*edge2)  
  
 det = matrix\_det(A2, B2, A1, B1)  
 if det == 0: return edge1[1]  
  
 point = [0, 0]  
 point[0] = -matrix\_det(C2, B2, C1, B1) / det  
 point[1] = -matrix\_det(A2, C2, A1, C1) / det  
 return point

**Визуальные характеристики**

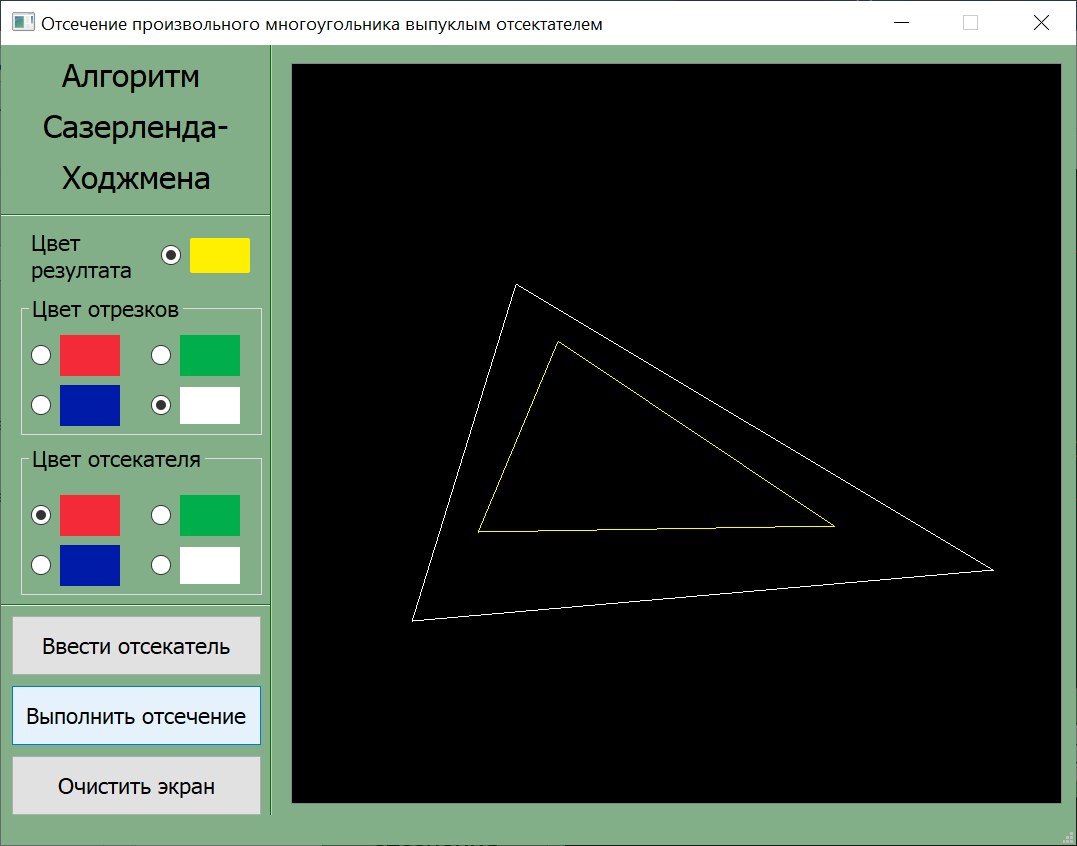
Ниже приведены примеры работы программы в различных ситуациях:

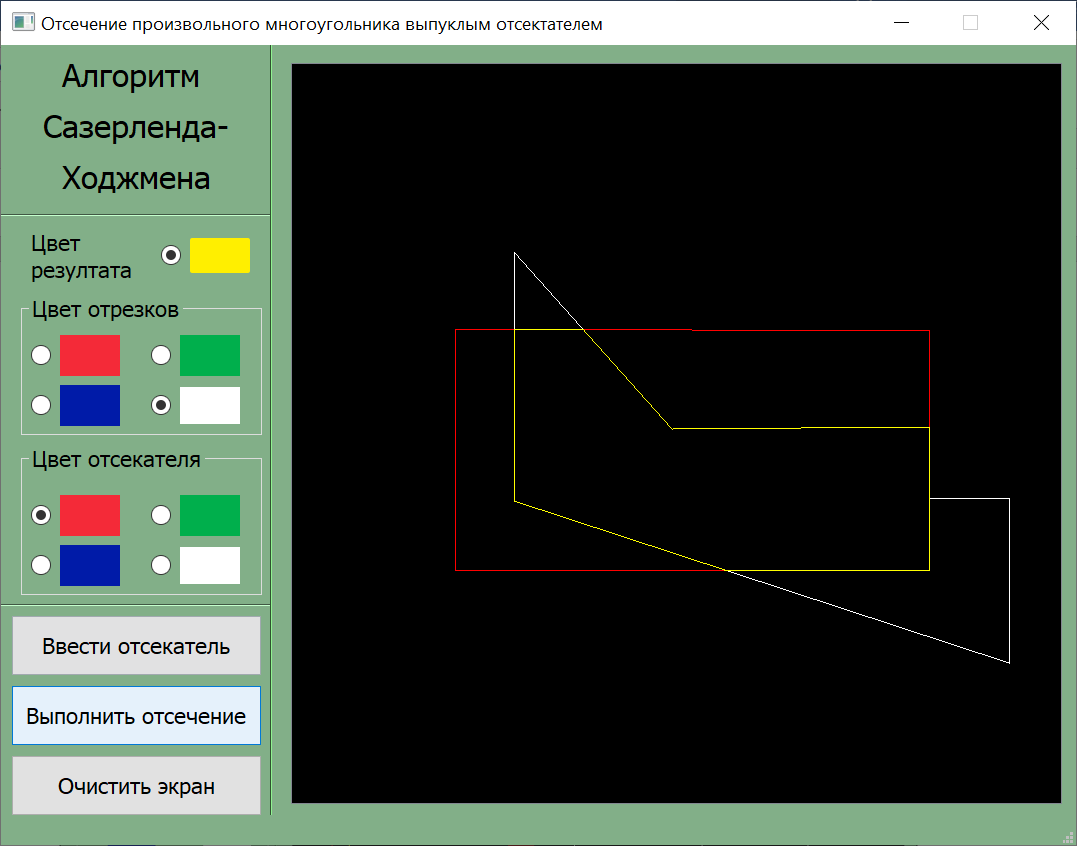
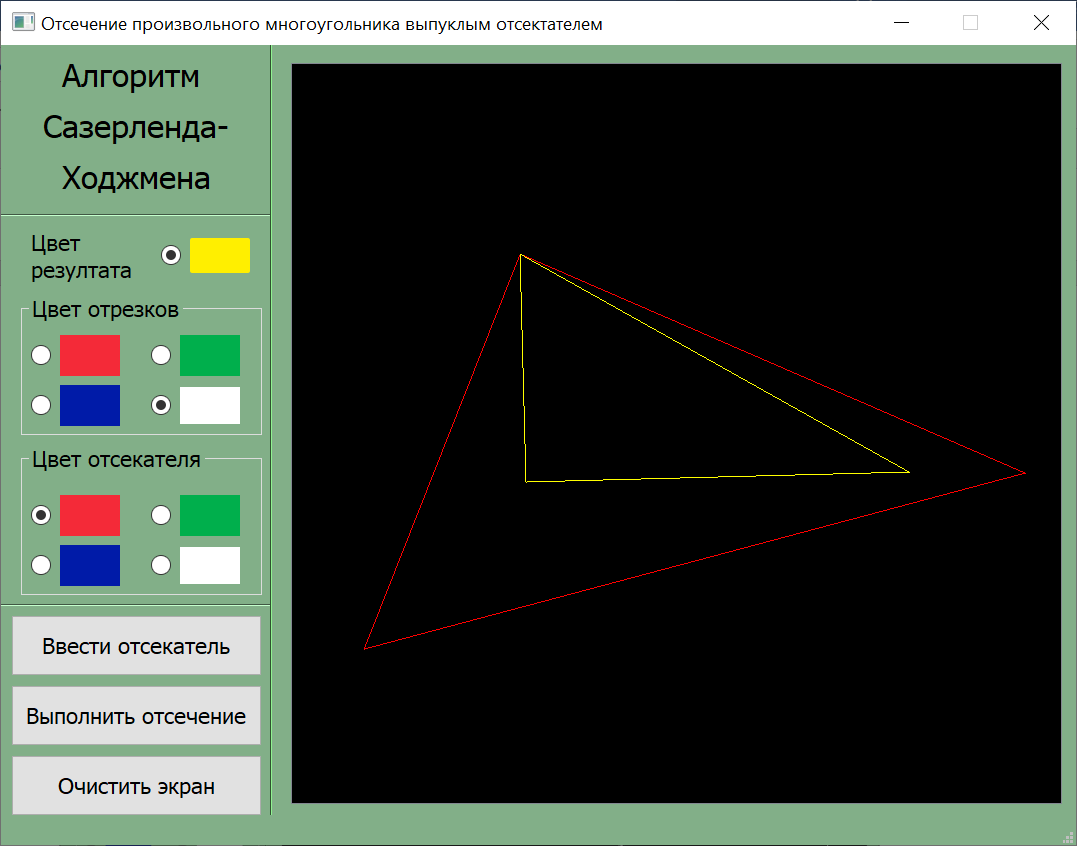
 

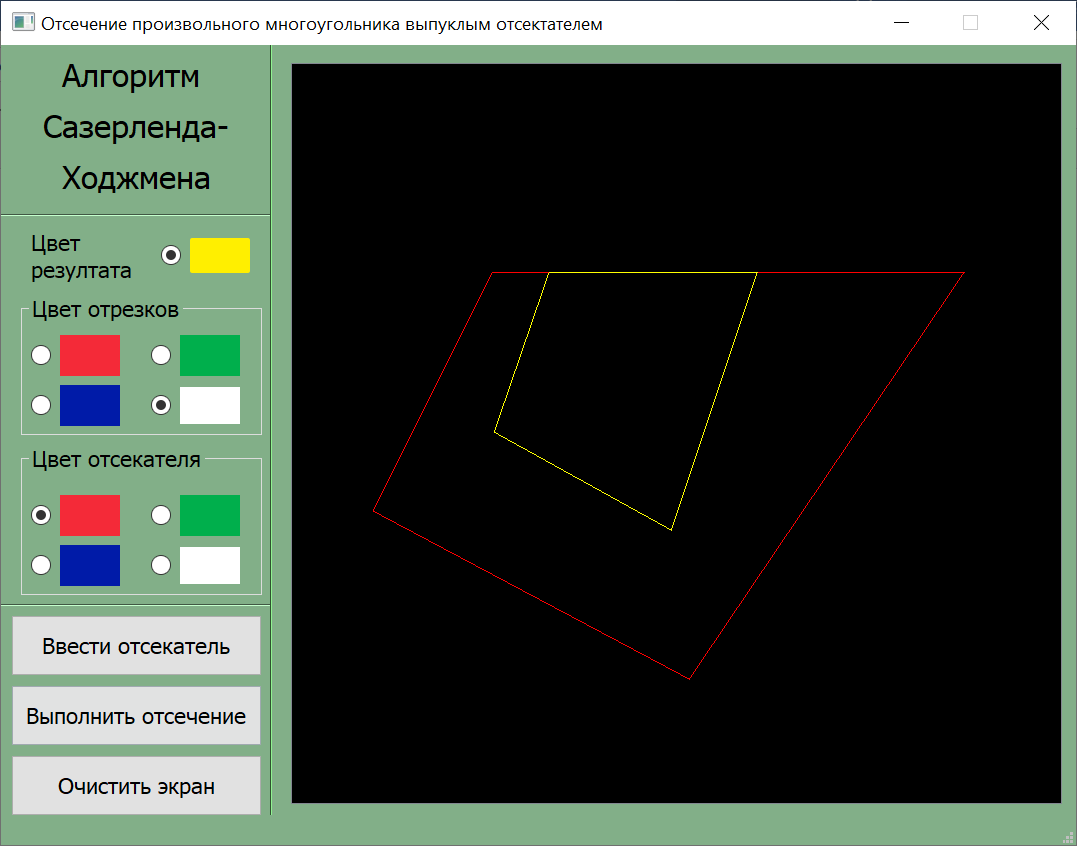
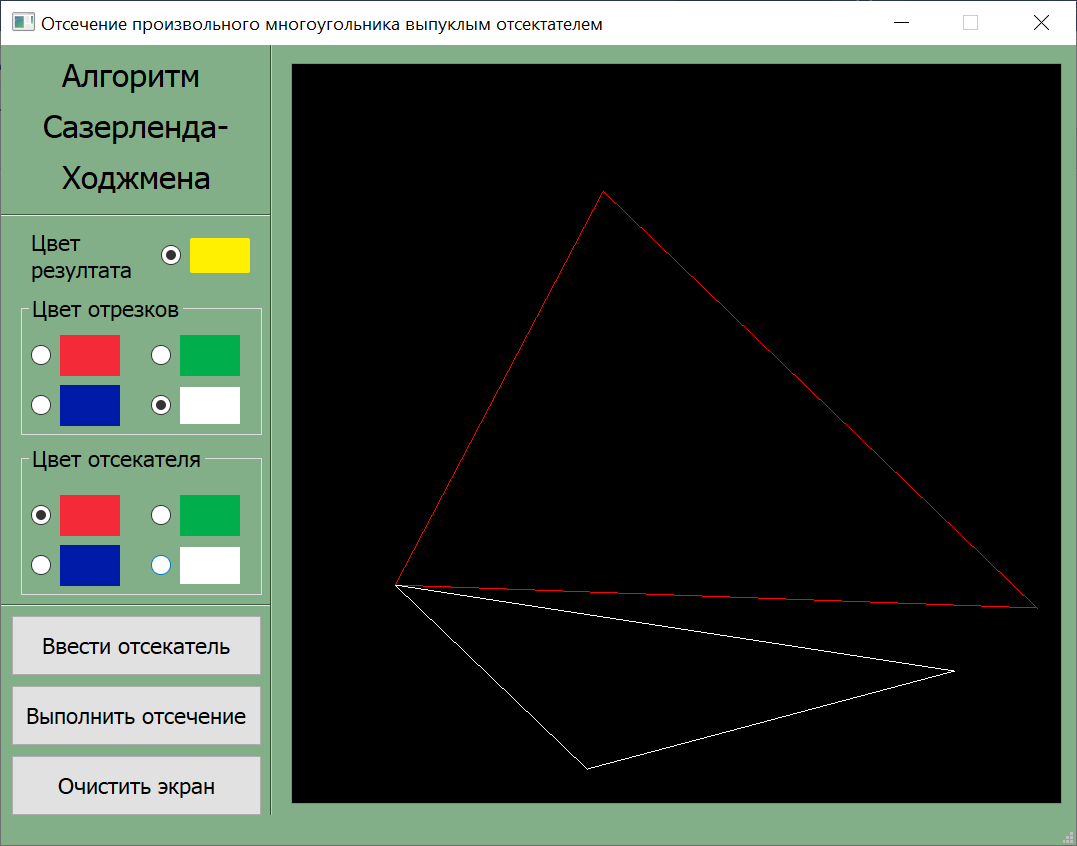




(отсекатель внутри отсекаемой фигуры)



**Заключение**

В ходе лабораторной работы был изучен и реализован алгоритм отсечения многоугольников произвольным выпуклым отсекателем.