|  |  |
| --- | --- |
| 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**

|  |  |
| --- | --- |
| **Тема: Отсечение произвольного многоугольника выпуклым отсекателем (Алгоритм Сазерленда-Ходжмена)**  **Студент** ВоякинА. Я.  **Группа ИУ7-44Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |

Москва.

2020 г.

**Цель работы:**

Изучение и программная реализация алгоритма Сазерленда-Ходжмена отсечения многоугольников.

**Условие задания:**

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

**Описание алгоритма:**

Идея данного алгоритма достаточно проста. На каждом шаге отсечения исходный и промежуточные многоугольники отсекаются последовательно очередной границей отсекателя. Отсечение многоугольника относительно одной прямой не представляет больших затруднений.

На каждом шаге отсечения алгоритм работает со списком вершин многоугольника и в результате также получается список вершин нового многоугольника. Причем все вершины нового многоугольника лежат по видимую сторону очередной границы отсекателя. Каждое ребро многоугольника отсекается независимо от других, поэтому для подробного рассмотрения сути алгорита достаточно рассмотреть все возможные комбинации взаимного расположения ребра отсекаемого многоугольника и ребра отсекателя.

Особым образом обрабатывается первая точка многоугольника: для нее требуется определить только видимость.

**Инструкция к программе:**

Рисование отсекателя происходит с помощью правой кнопки мыши.

Соединение первой точки отсекателя и последней происходит при нажатии клавиши Shift.

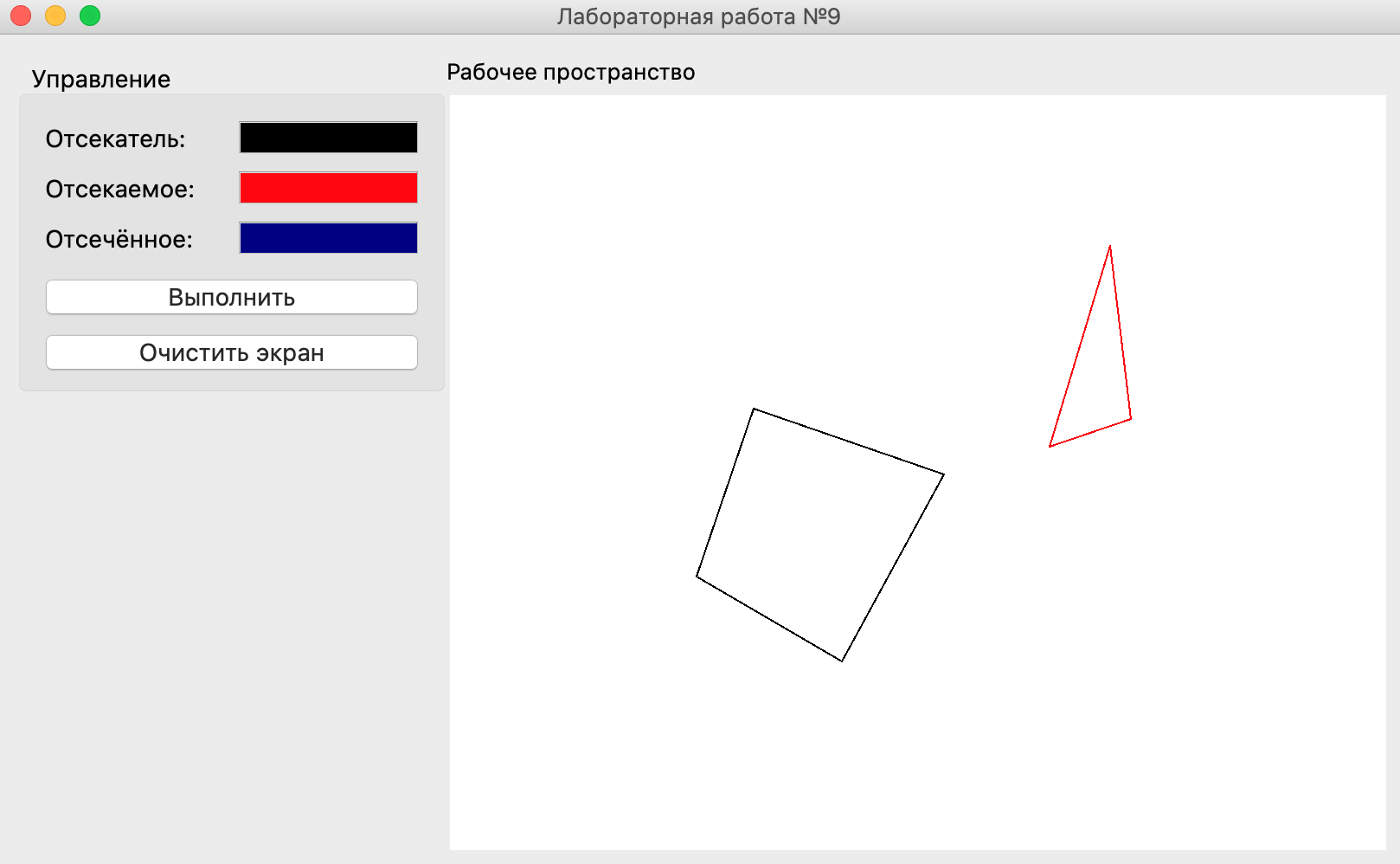
Рисование многоугольников происходит с помощью левой кнопки мыши.

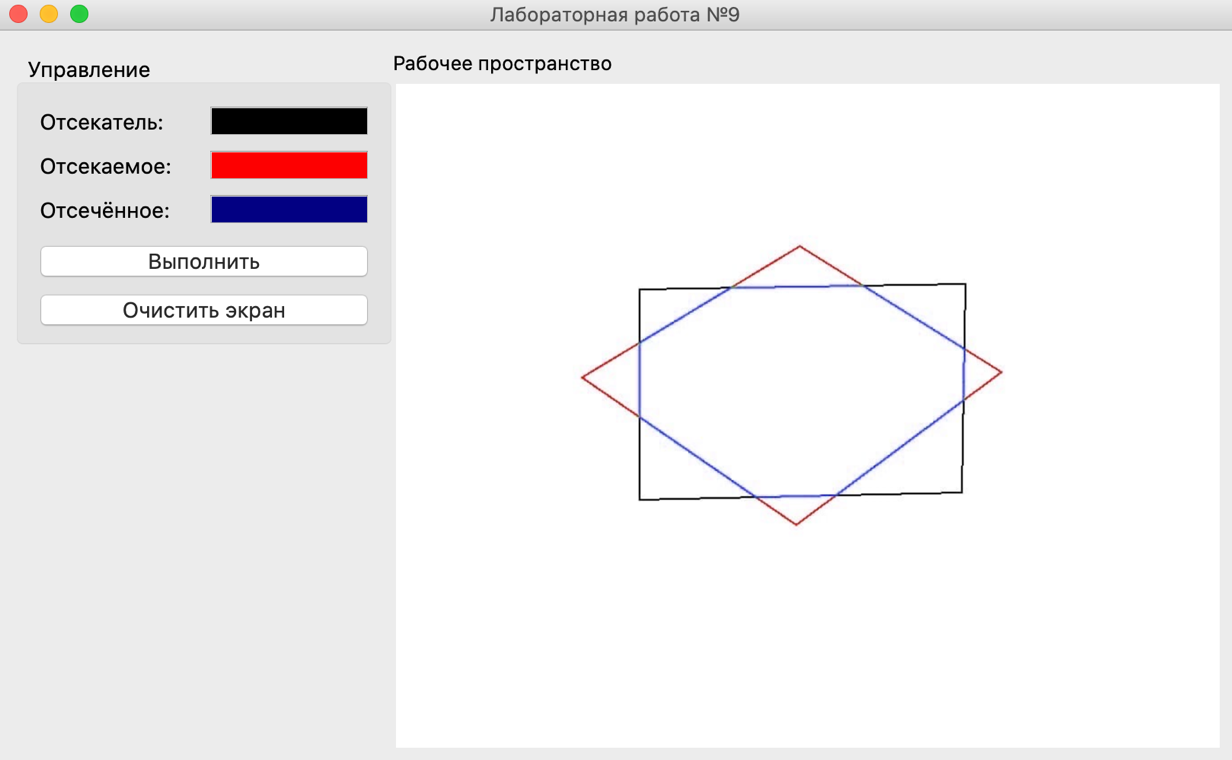
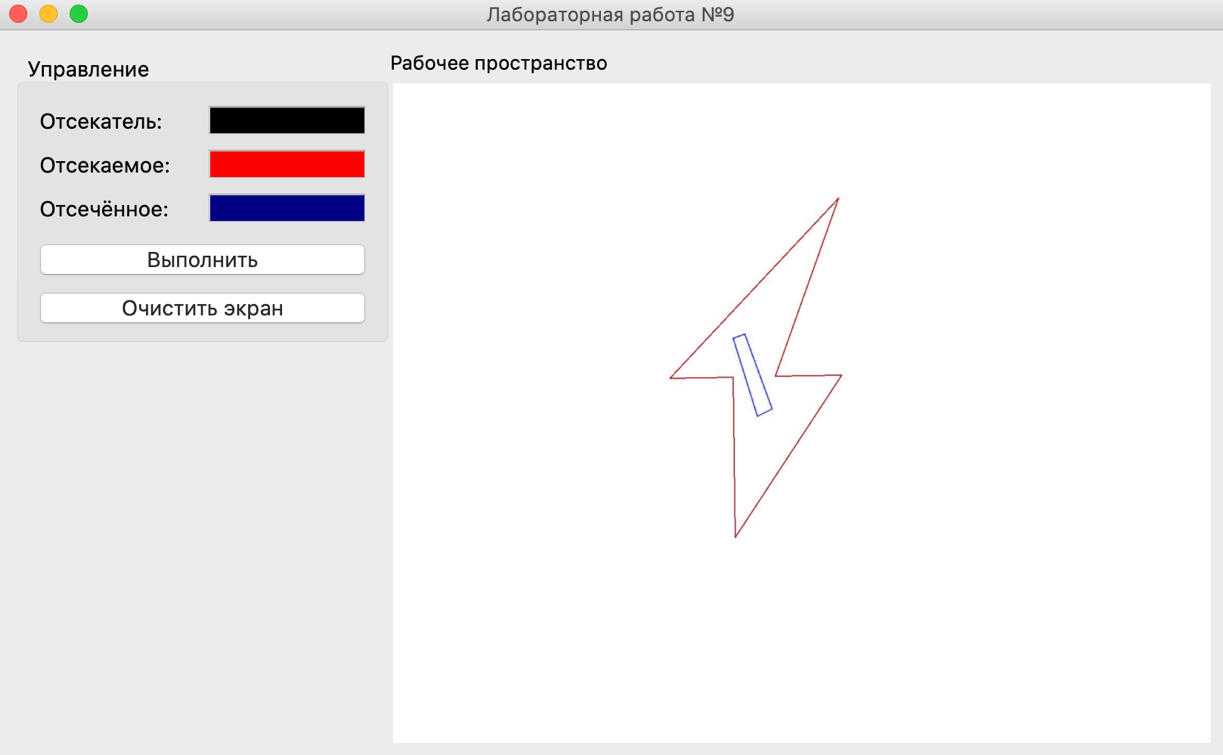
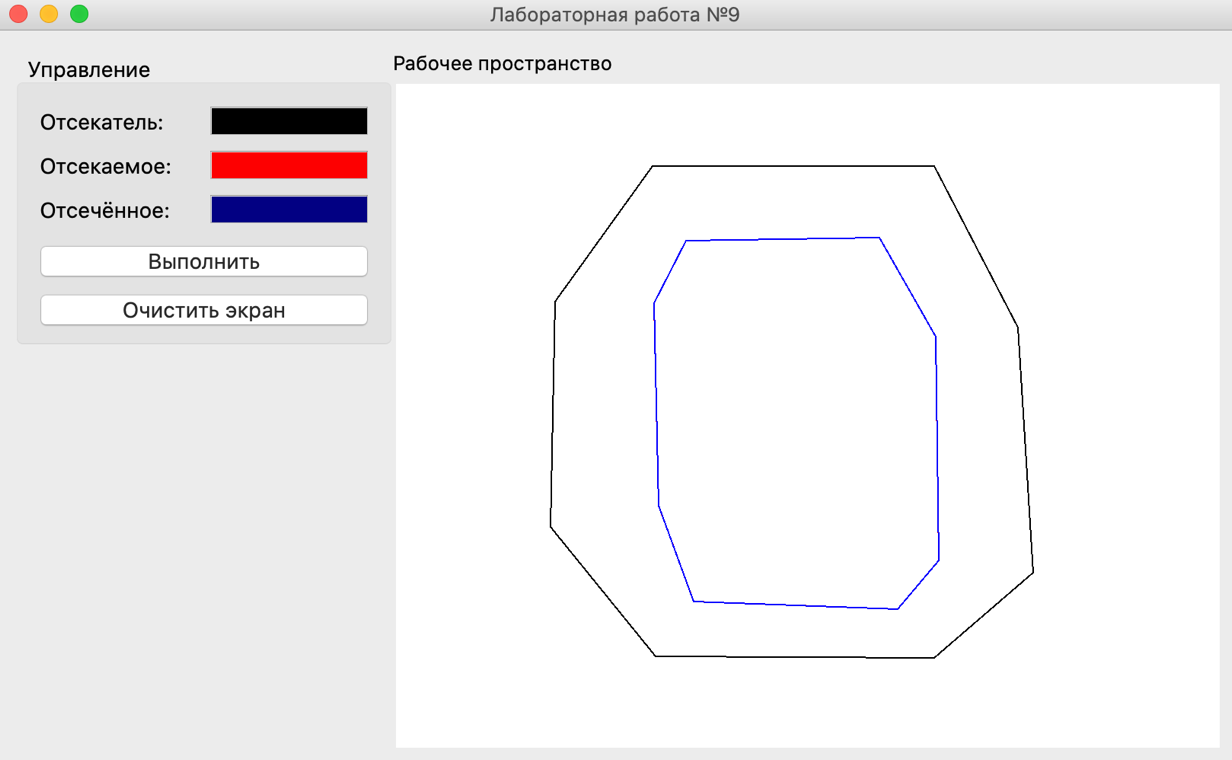
Рисование вертикальных и горизонтальных линий происходит с помощью левой кнопки мыши с зажатыми клавишами «V» и «H» соответственно. Выбор цвета происходит по нажатии на прямоугольник с цветом, который хотите поменять.

**Код алгоритма.**

def visibility(*self*, x, y, w1\_x, w1\_y, w2\_x, w2\_y):  
 buff\_1 = (x - w1\_x) \* (w2\_y - w1\_y)  
 buff\_2 = (y - w1\_y) \* (w2\_x - w1\_x)  
 buff\_1 -= buff\_2  
 return sign(buff\_1)  
  
def is\_crossing(*self*, s\_x, s\_y, buff\_x, buff\_y, w1\_x, w1\_y, w2\_x, w2\_y):  
 vis\_1 = *self*.visibility(s\_x, s\_y, w1\_x, w1\_y, w2\_x, w2\_y)  
 vis\_2 = *self*.visibility(buff\_x, buff\_y, w1\_x, w1\_y, w2\_x, w2\_y)  
 if vis\_1 \* vis\_2 < 0:  
 return True  
 return False  
  
def crossing(*self*, p1\_x, p1\_y, p2\_x, p2\_y, w1\_x, w1\_y, w2\_x, w2\_y):  
 coef = [[p2\_x - p1\_x, w1\_x - w2\_x], [p2\_y - p1\_y, w1\_y - w2\_y]]  
 prav = [w1\_x - p1\_x, w1\_y - p1\_y]  
 coef = inv(coef)  
 param = coef.dot(prav)  
 return p1\_x + (p2\_x - p1\_x) \* param[0], p1\_y + (p2\_x - p1\_x) \* param[0]  
  
def makeCut(*self*):  
 for i in range(0, len(*self*.cutter\_x)):  
 q\_x = []; q\_y = []  
 w1\_x = *self*.cutter\_x[i]; w1\_y = *self*.cutter\_y[i]  
 w2\_x = *self*.cutter\_x[(i + 1) % len(*self*.cutter\_x)]; w2\_y = *self*.cutter\_y[(i + 1) % len(*self*.cutter\_y)]  
 for j in range(0, len(*self*.p\_x)):  
 if j != 0:  
 goto .q1  
 buff\_x = *self*.p\_x[j]; buff\_y = *self*.p\_y[j]  
 goto .q2  
  
 label .q1  
 if not *self*.is\_crossing(s\_x, s\_y, *self*.p\_x[j], *self*.p\_y[j], w1\_x, w1\_y, w2\_x, w2\_y):  
 goto .q2  
 per = *self*.crossing(s\_x, s\_y, *self*.p\_x[j], *self*.p\_y[j], w1\_x, w1\_y, w2\_x, w2\_y)  
 q\_x.append(per[0]); q\_y.append(per[1])  
  
 label .q2  
 s\_x = *self*.p\_x[j]; s\_y = *self*.p\_y[j]  
 if *self*.visibility(s\_x, s\_y, w1\_x, w1\_y, w2\_x, w2\_y) < 0:  
 goto .q3  
 q\_x.append(s\_x); q\_y.append(s\_y)  
  
 label .q3  
 if len(q\_x) == 0:  
 goto .q4  
 if not *self*.is\_crossing(s\_x, s\_y, buff\_x, buff\_y, w1\_x, w1\_y, w2\_x, w2\_y):  
 goto .q4  
 per = *self*.crossing(s\_x, s\_y, buff\_x, buff\_y, w1\_x, w1\_y, w2\_x, w2\_y)  
 q\_x.append(per[0]); q\_y.append(per[1])  
  
 label .q4  
 *self*.p\_x = q\_x; *self*.p\_y = q\_y

**Демонстрация работы программы.**

****

****