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

|  |  |
| --- | --- |
| **Дисциплина Компьютерная графика**  **Тема Программная реализация алгоритма отсечения отрезка регулярным отсекателем**  **9 вариант – Алгоритм разбиения средней точкой**  **Студент Иванов В.А.**  **Группа ИУ7-42Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель Куров А.В.** |  |

Москва.

2020 г.

**Цель работы**

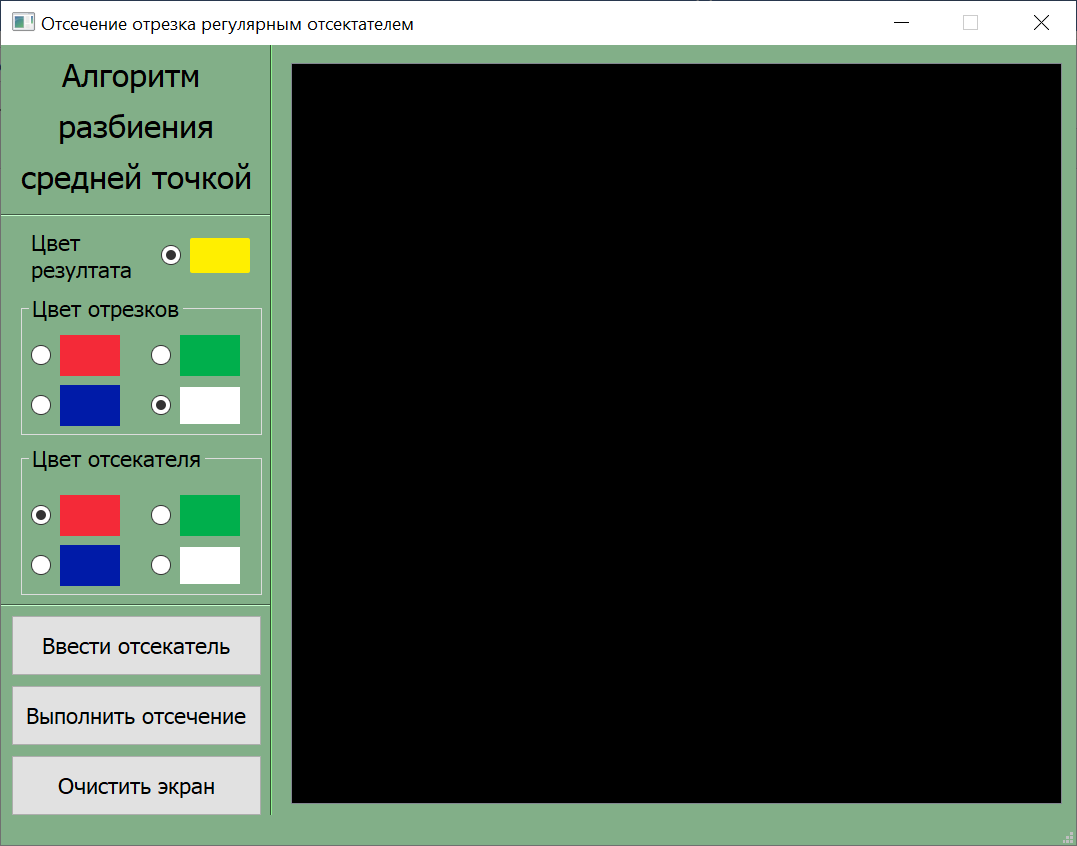
Реализация алгоритма разбиения средней точкой.

**Описание задания**

1. Реализовать алгоритм разбиения средней точкой
2. Обеспечить возможность ввода отсекателя прямоугольником и до 10 отрезков с помощью мыши (в т.ч. и горизонтальных и вертикальных отрезков)

**Описание графического интерфейса**

Для использования функционала поставленных задач, был создан графический интерфейс

****

Интерфейс предоставляет возможность:

* Выбора цвета рёбер и отсекателя
* Ввода отрезков при помощи мыши. Нажатие ЛКМ для ввода произвольного отрезка и ПКМ для ввода горизонтального/вертикального отрезка.
* Ввод отсекателя при помощи мыши

**Описание и реализация алгоритма**

Задачей данной лабораторной работы служит выполнение отсечения множества отрезков регулярным отсекателем, используя алгоритм разбиения средней точкой.

Алгоритм создан как один из способов оптимизации простого алгоритма. Идея алгоритма состоит в том, что точка пересечения находится не аналитически, а численным методом – бинарным поиском, то есть уточнением корня уравнения пересечения отрезка с стороной отсекателя с заданной точностью. Сутью работы алгоритма является поиск для каждого из концов отрезка самой удалённой от него видимой точки. Процесс работы состоит в разбиении отрезка средней точкой, и дальнейшем рассмотрением получившихся отрезков, имеющих частичную видимость. Средняя точка находится как полусумма координат двух точек.

Преимуществом алгоритма над простым аналогом является то, что деление на 2 уровне может быть реализовано с помощью побитового сдвига вправо, что гораздо быстрее обычного деления.

Также, как и в простом алгоритме, за основу берётся анализ кодов точек отрезка. Сами коды показывают положение отрезка относительно каждой из сторон отсекателя.

Для каждого отрезка алгоритм имеет следующие шаги:

1. Крайними точками p1, p2 выбираются концевые точки отрезка.
2. Вычислить коды p1, p2. В случае, если отрезок тривиально полностью видим (т.е. коды обеих точек = 0000), или полностью невидим (т.е. логическое И кодов ≠ 0), алгоритм переходит к пункту 5 или завершает работу соответственно.
3. Поиск видимой точки, наиболее удалённой от p1. В случае, если обе крайние видимые точки уже найдены, выполняется проверка видимости отрезка между этими точками. В случае видимости – переход к пункту 5, иначе – завершает работу.
4. В процессе поиска вычисляется средняя точка pm между текущими p1, p2. В случае, если отрезок pm-p2 невидим, точка pm записывается в p2. Иначе, pm будет записана в p1. Пункт 3 повторяется до тех пор, пока отрезок p1-p2 не выродится в точку.
5. На место точки p1 записывается текущая p2, а на место p2 – начальная точка p1. Алгоритм переходит к пункту 1. За счёт этой перестановки производится поиск уже второй крайней видимой точки, на этот раз наиболее удалённой от другого конца отрезка.
6. Отрисовка видимой части отрезка.

Далее алгоритм продолжается уже для следующего отрезка.

Программная реализация:

В качестве погрешности EPS выбрана величина . Основная функция:

def mid\_point\_cut(self):  
 for seg in self.segment\_arr:  
 p1, p2 = seg  
 i = 1  
 draw\_flag = True  
 while True:  
 t1 = self.point\_code(p1)  
 t2 = self.point\_code(p2)  
  
 sum1 = self.code\_sum(t1)  
 sum2 = self.code\_sum(t2)  
 if sum1 == 0 and sum2 == 0:  
 break  
 if self.code\_and(t1, t2):  
 draw\_flag = False  
 break  
  
 temp = p1  
 if i > 2:  
 draw\_flag = not self.code\_and(t1, t2)  
 break  
  
 while sum2 and self.distance(p1, p2) > EPS:  
 pm = self.mid\_point(p1, p2)  
 t1 = self.point\_code(pm)  
 if self.code\_and(t1, t2): p2 = pm  
 else: p1 = pm  
 p1 = p2  
 p2 = temp  
 i += 1  
  
 if draw\_flag:  
 self.draw\_line(p1, p2)

Вычисление кода точки:

def point\_code(self, point):  
 t = [0] \* 4  
 if point[0] < self.x\_left: t[3] = 1  
 if point[0] > self.x\_right: t[2] = 1  
 if point[1] < self.y\_low: t[1] = 1  
 if point[1] > self.y\_up: t[0] = 1  
 return t

Вычисление суммы и логического И кодов:

def code\_sum(self, t\_code):  
 return sum(t\_code)

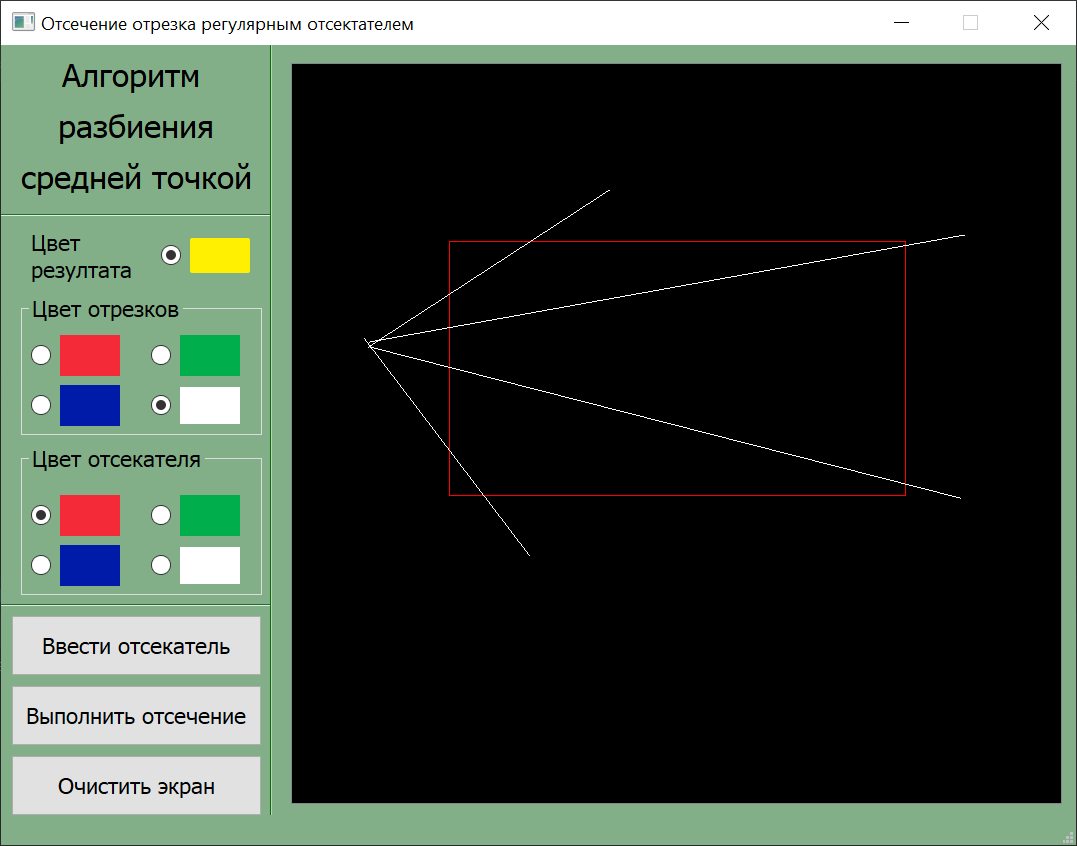
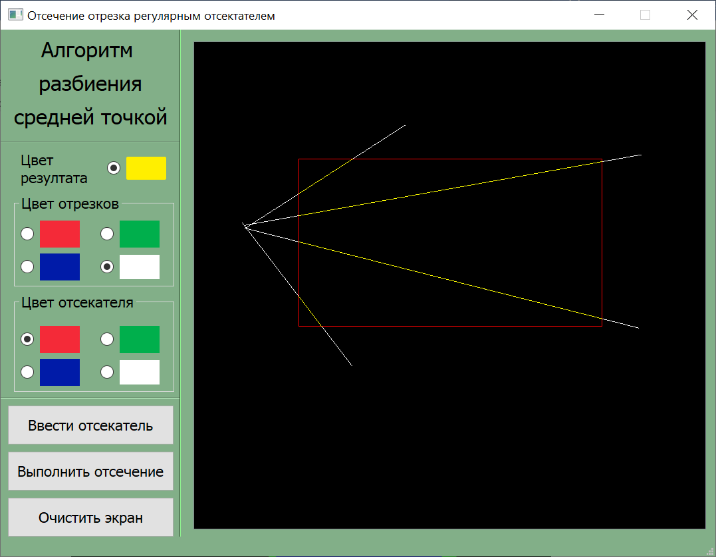
def code\_and(self, t1, t2):  
 res = 0  
 for i in range(4):  
 if t1[i] and t2[i]:  
 res += 1  
 return res

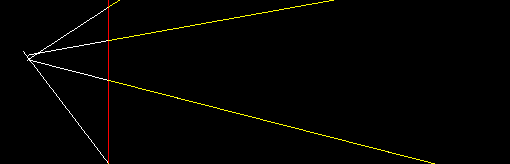
Вычисление средней точки:

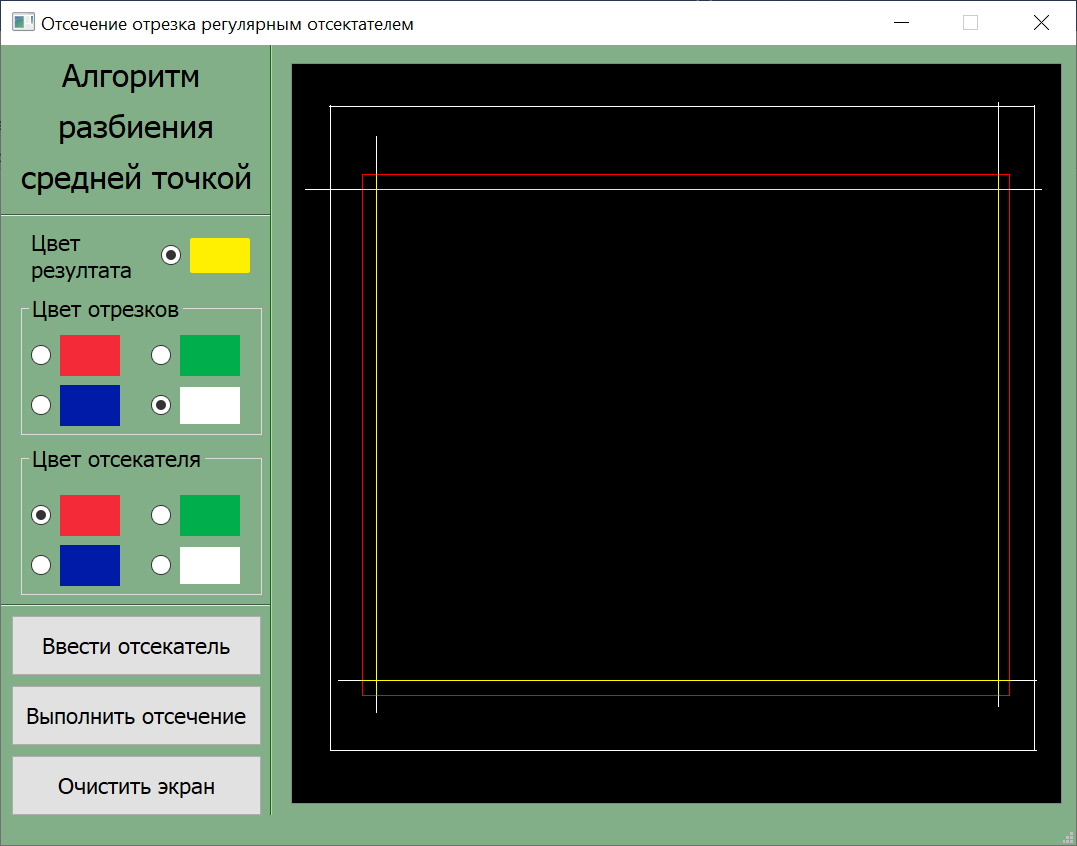
def mid\_point(self, p1, p2):  
 x = (p1[0] + p2[0]) / 2  
 y = (p1[1] + p2[1]) / 2  
 return [x, y]

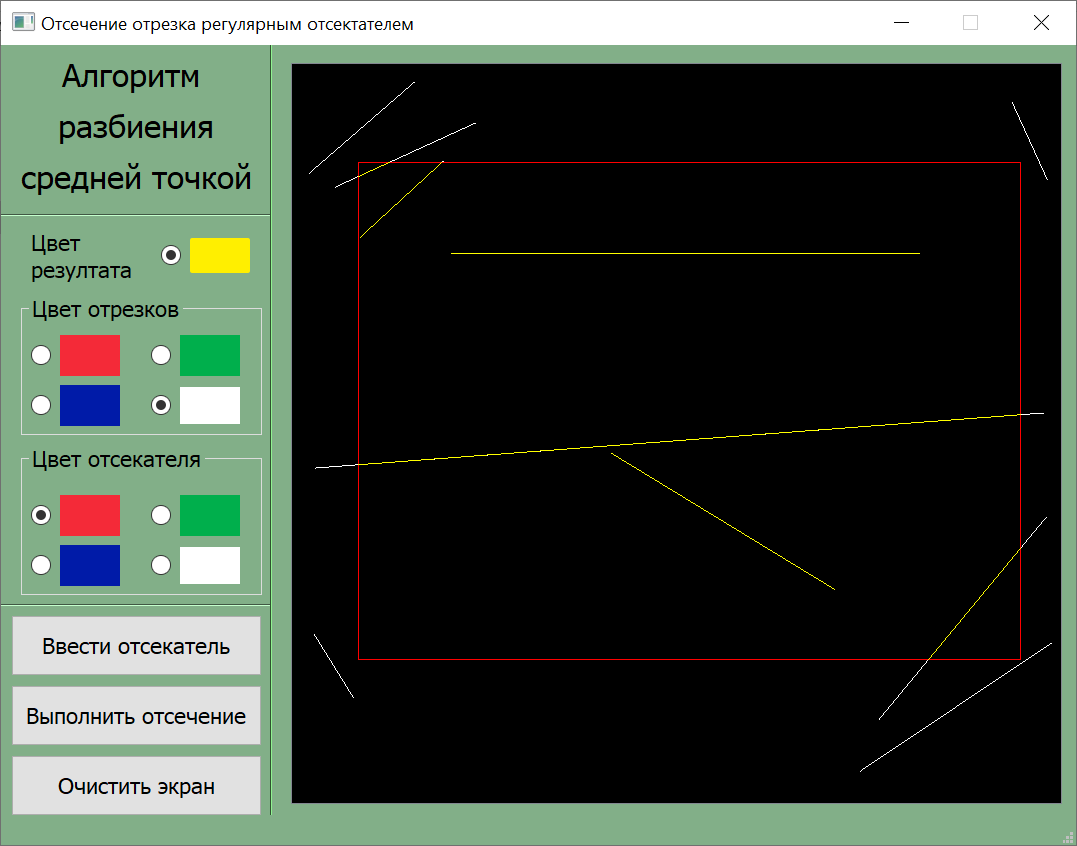
**Визуальные характеристики**

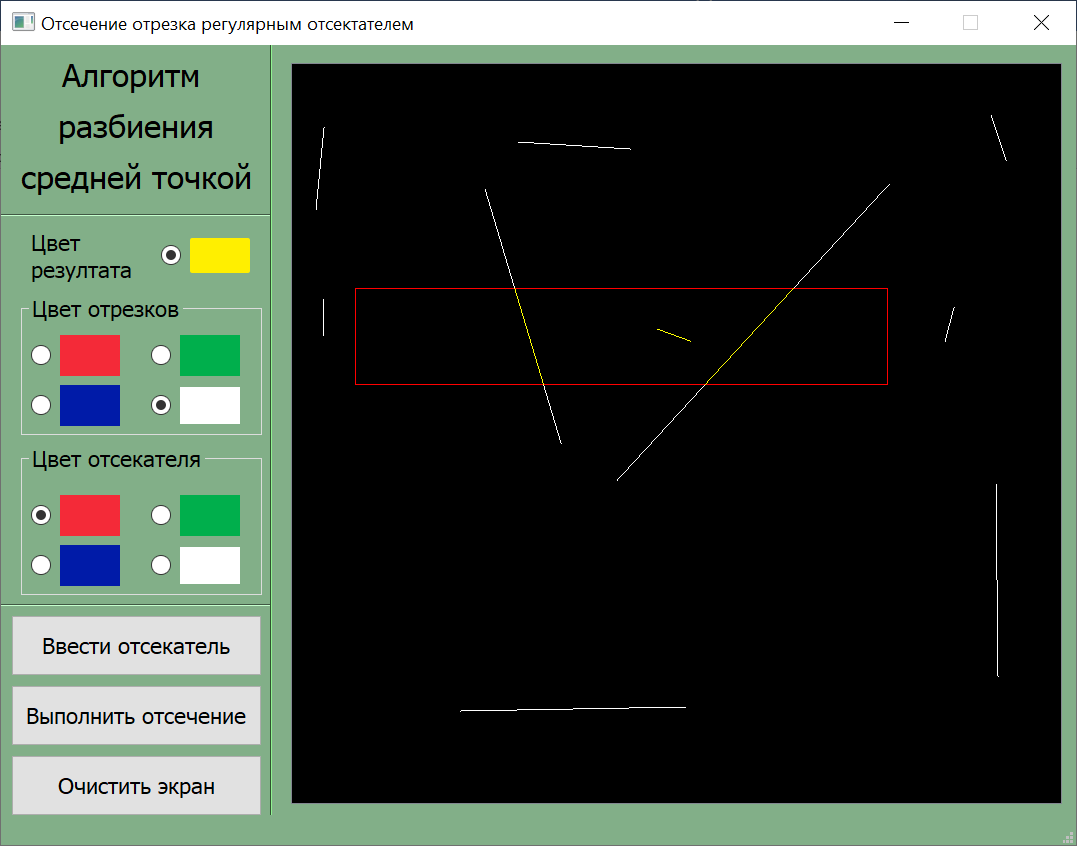
Ниже приведены примеры работы программы в различных ситуациях:









**Заключение**

В ходе лабораторной работы были изучены различные алгоритмы отсечения отрезков регулярным отсекателем и был реализован алгоритм разбиения средней точки.