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

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

Москва.

2020 г.

**Цель работы**

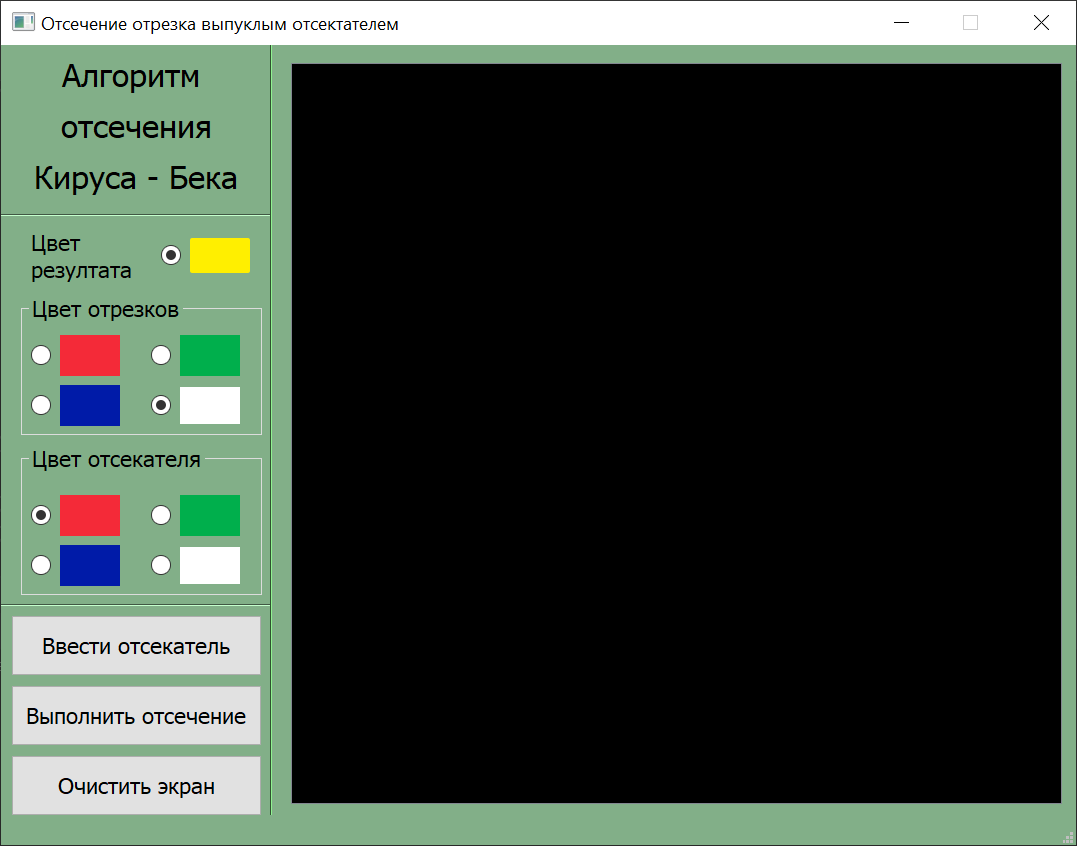
Реализация алгоритма Кируса-Бека.

**Описание задания**

1. Реализовать алгоритма Кируса-Бека
2. Обеспечить возможность ввода отсекателя многоугольником и до 10 отрезков с помощью мыши (в т.ч. и горизонтальных и вертикальных отрезков и рёбер)

**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 произвольным выпуклым отсекателем, используя алгоритм Кируса-Бека.

Идея алгоритма состоит в том, отсечение производится за счёт внутренних нормалей к каждой стороне отсекателя и свойств скалярного произведения. В самом начале обработки отрезков выполняется проверка условия выпуклости отсекателя. Производится оно проверкой условия равенства знаков всех векторных произведений директрис смежных сторон. В случае, если одно из произведений равно 0, можно сделать вывод о том, что отсекатель вырожден в отрезок, а если знак любых двух векторных произведений не совпадают, то отсекатель является невыпуклым. В обоих случаях отсечение отрезков производится не будет.

Далее производится нахождение внутренних нормалей для каждой стороны отсекателя. Нормаль находится с помощью использования уравнения прямой, проходящей через сторону. В случае, если в проверке выпуклости произведения имели отрицательный знак, нормали к сторонам будут внешними, и их следует инвертировать. В случае положительного знака нормали уже будут направлены внутрь отсекателя.

Следующим этапом начинается обработка каждого из отрезков. Поиск точек пересечения со сторонами производится с помощью отношений t0 и t1, соответствующих “нижней” и “верхней” точке пересечения соответственно. Нижняя точка находится ближе к первой точке отрезка (t=0), а верхняя ближе ко второй(t=1). Сами t находятся из решения уравнения скалярного произведения нормали стороны на вектор, от точки на отрезке с отношением t до точки стороны. В случае, когда t соответствует точке пересечения, это произведение будет равно 0. Для решения этого уравнения используется директриса отрезка и вектор от края стороны до края отрезка. Изначально считается, что отрезок полностью видим, т.е. t0=0 и t1=1

Для очередного пересечения отрезка со стороной отсекателя производится анализ скалярного произведения директрисы и нормали стороны. В случае, если оно положительно, считается, что пересечение относится к t0, а при отрицательном случае, к t1. Если произведение нулевое, то это значит, что отрезок параллелен частично совпал со стороной отсекателя. В этом случае точка пересечения не может быть определена, и если скалярное произведение нормали и вектора между стороной и отрезком не менее 0, то данная сторона пропускается, так как точки отсечения с этим отрезком могут быть установлены сторонами, смежными данной. Иначе, можно сразу сделать вывод о полной невидимости и завершить алгоритм.

Так как в выпуклом отсекателе может быть лишь две точки пересечения с отрезком, из всего множества точек, относящихся к нижнему и верхнему пределу, действительными пересечениями является лишь максимальная и минимальная точки по t соответственно. При этом, если для нижней границы t оказался >1, или для верхней <0, то также делается вывод о полной невидимости отрезка, и алгоритм завершается. Прочие значения t, лежащие вне интервала [0, 1] изначально игнорируются, так как не являются действительными пересечениями.

В случае t0 <= t1 производится отрисовка отрезка, так как обратная ситуация может возникнуть только у невидимого отрезка.

Далее алгоритм продолжается уже для следующего отрезка.

**Программная реализация:**

Отсечение массива отрезков

def cut\_segments(self):  
 dir = is\_convex(self.cut\_arr)  
 if not dir: return False  
 n\_arr = find\_normals(self.cut\_arr, dir == -1)  
  
 for segment in self.segment\_arr:  
 self.cyrus\_beck\_cut(segment, n\_arr)  
 return True

Отсечение отрезка  
def cyrus\_beck\_cut(self, segment, n\_arr):  
 t0, t1 = 0, 1  
 d = edge\_dir(\*segment)  
 for i in range(len(self.cut\_arr)):  
 w = edge\_dir(self.cut\_arr[i][0], segment[0])  
  
 d\_mlt = scalar\_mlt(d, n\_arr[i])  
 w\_mlt = scalar\_mlt(w, n\_arr[i])  
 if d\_mlt == 0:  
 if w\_mlt < 0: return  
 else: continue  
  
 t = -w\_mlt / d\_mlt  
 if d\_mlt > 0:  
 if t > 1: return  
 t0 = max(t, t0)  
 else:  
 if t < 0: return  
 t1 = min(t, t1)  
 if t0 <= t1:  
 p0 = edge\_ratio(segment[0], d, t0)  
 p1 = edge\_ratio(segment[0], d, t1)  
 self.draw\_line(p0, p1)

Поиск внутренних нормалей

def find\_normals(edge\_arr, is\_invert):  
 normal\_arr = [None] \* len(edge\_arr)  
 for i in range(len(edge\_arr)):  
 normal\_arr[i] = edge\_normal(\*edge\_arr[i])  
 if is\_invert:  
 normal\_arr[i] = neg\_vector(normal\_arr[i])  
 return normal\_arr

Проверка выпуклости и поиск направления нормалей

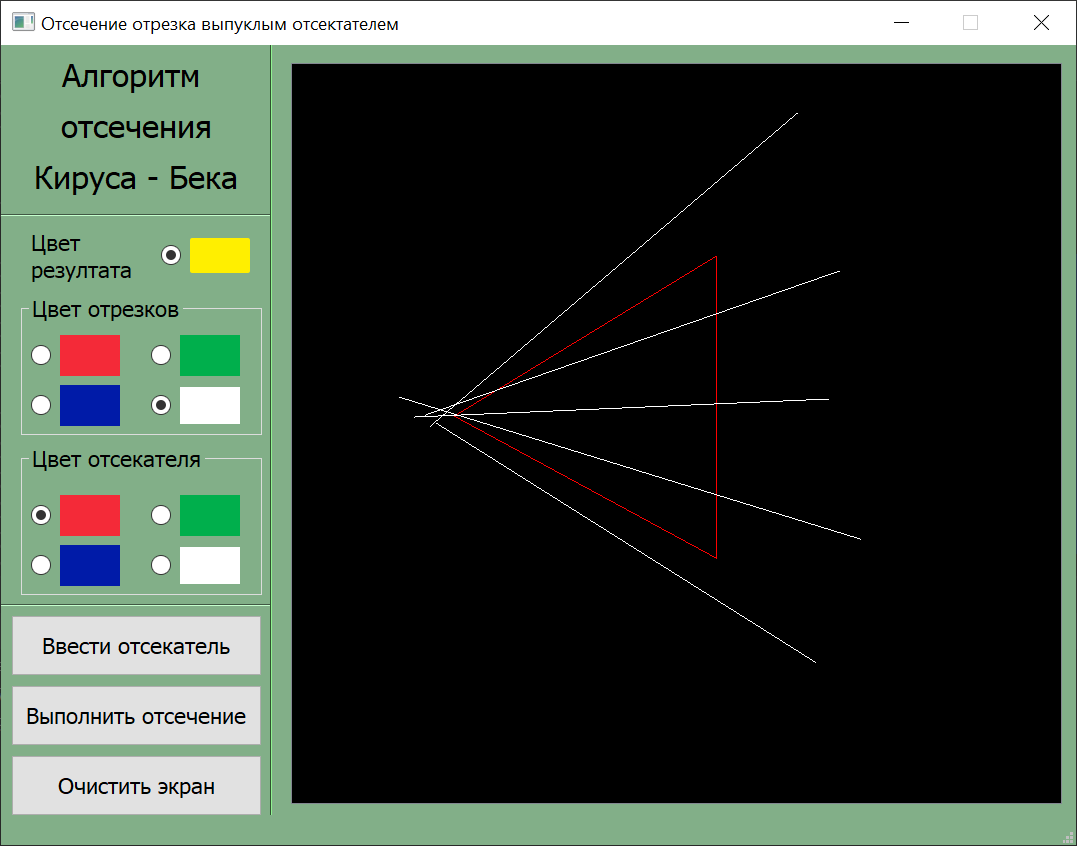
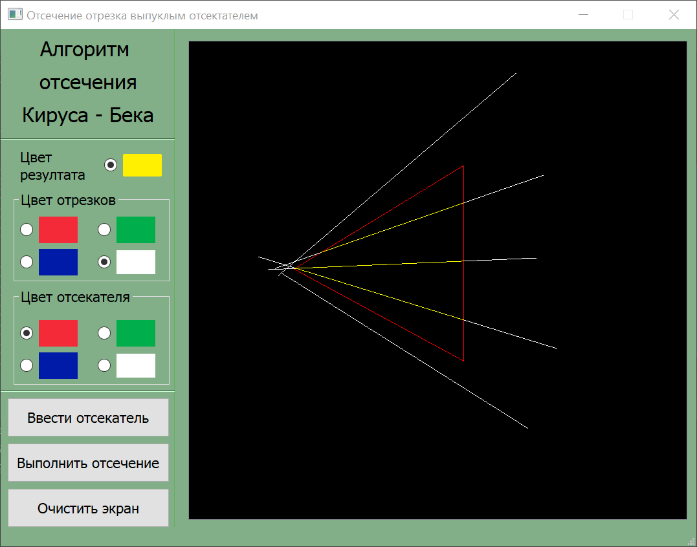
def is\_convex(edge\_arr):  
 v1 = edge\_dir(\*edge\_arr[-1])  
 v2 = edge\_dir(\*edge\_arr[0])  
 last\_mlt = vector\_mlt(v2, v1)  
  
 for i in range(1, len(edge\_arr)):  
 v1 = edge\_dir(\*edge\_arr[i - 1])  
 v2 = edge\_dir(\*edge\_arr[i])  
  
 mlt = vector\_mlt(v2, v1)  
 if mlt \* last\_mlt <=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

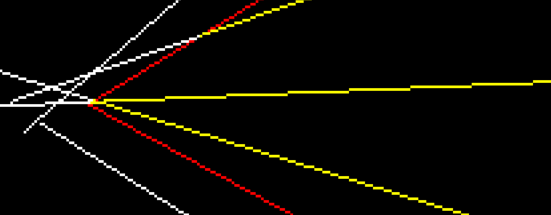
Функция работы с рёбрами и отрезками

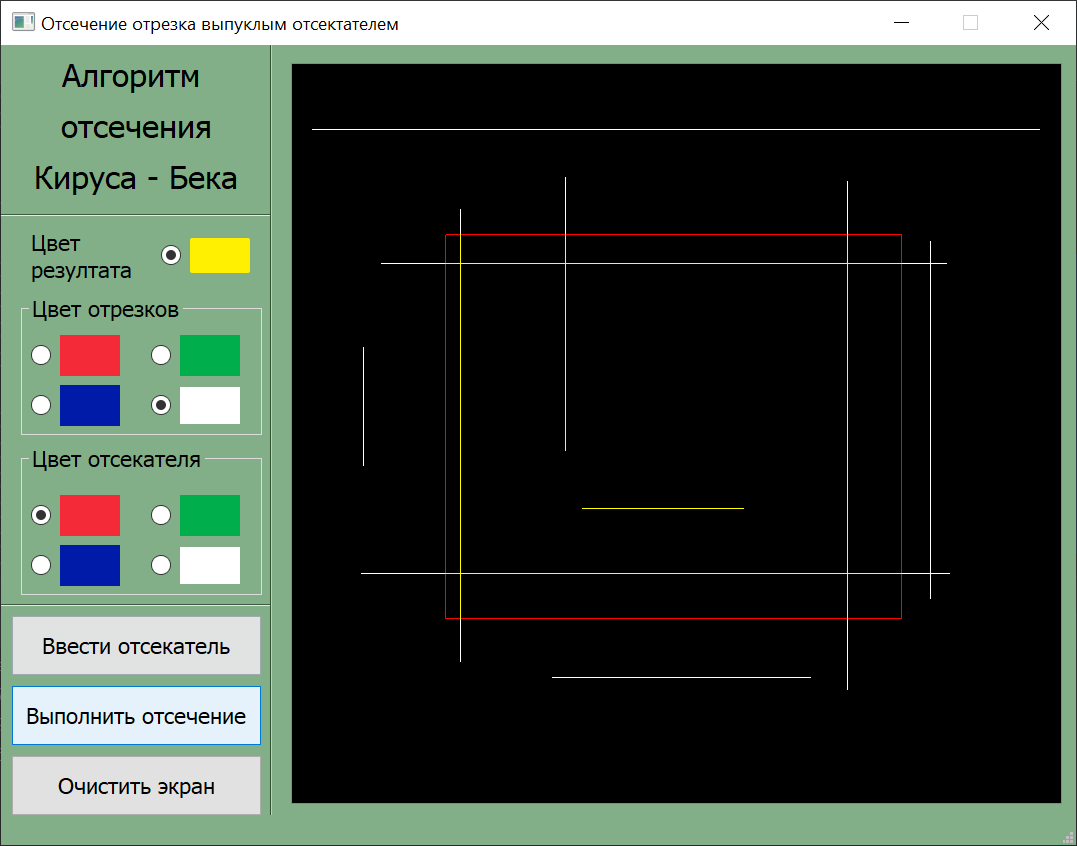
def edge\_mid(p1, p2):  
 p = [0, 0]  
 p[0] = (p1[0] + p2[0]) / 2  
 p[1] = (p1[1] + p2[1]) / 2  
 return p  
def edge\_dir(p1, p2):  
 x = p2[0] - p1[0]  
 y = p2[1] - p1[1]  
 return [x, y]  
def edge\_normal(p1, p2):  
 x = p2[1] - p1[1]  
 y = p1[0] - p2[0]  
 return [x, y]  
def edge\_ratio(p0, d, t):  
 p = [0, 0]  
 p[0] = p0[0] + d[0]\*t  
 p[1] = p0[1] + d[1]\*t  
 return p  
  
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 neg\_vector(v):  
 return [-v[0], -v[1]]

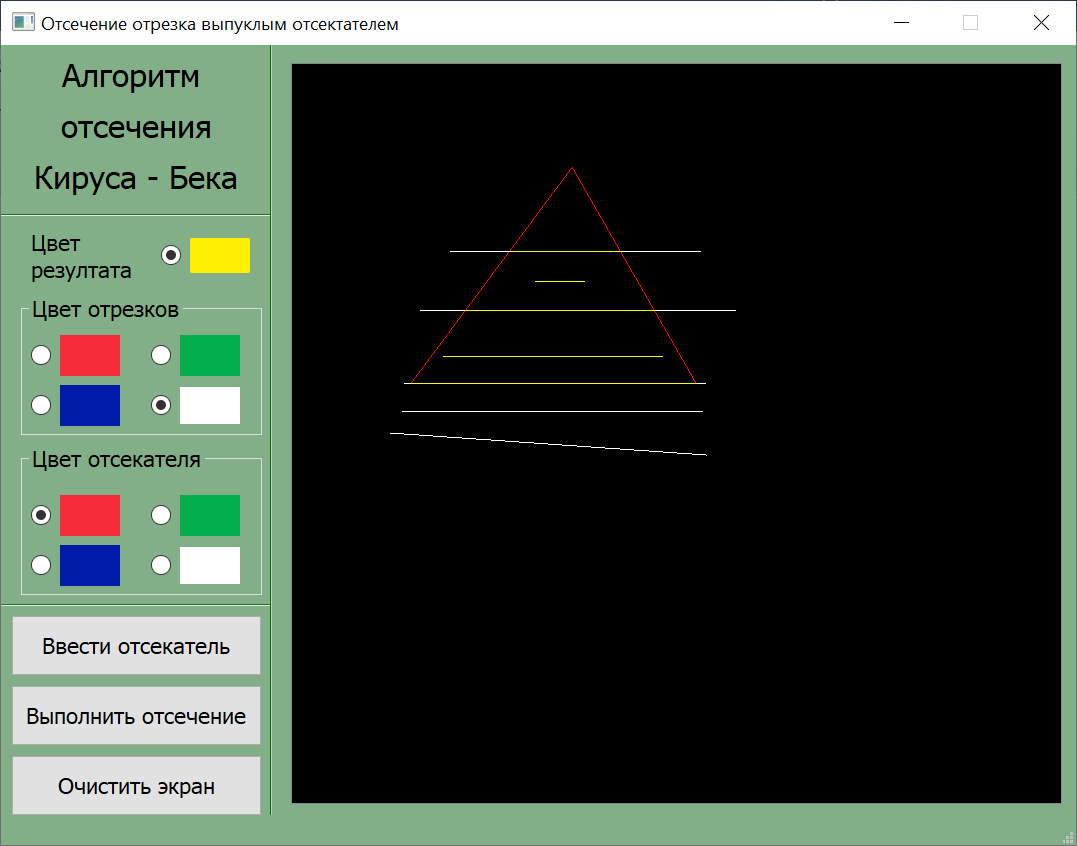
**Визуальные характеристики**

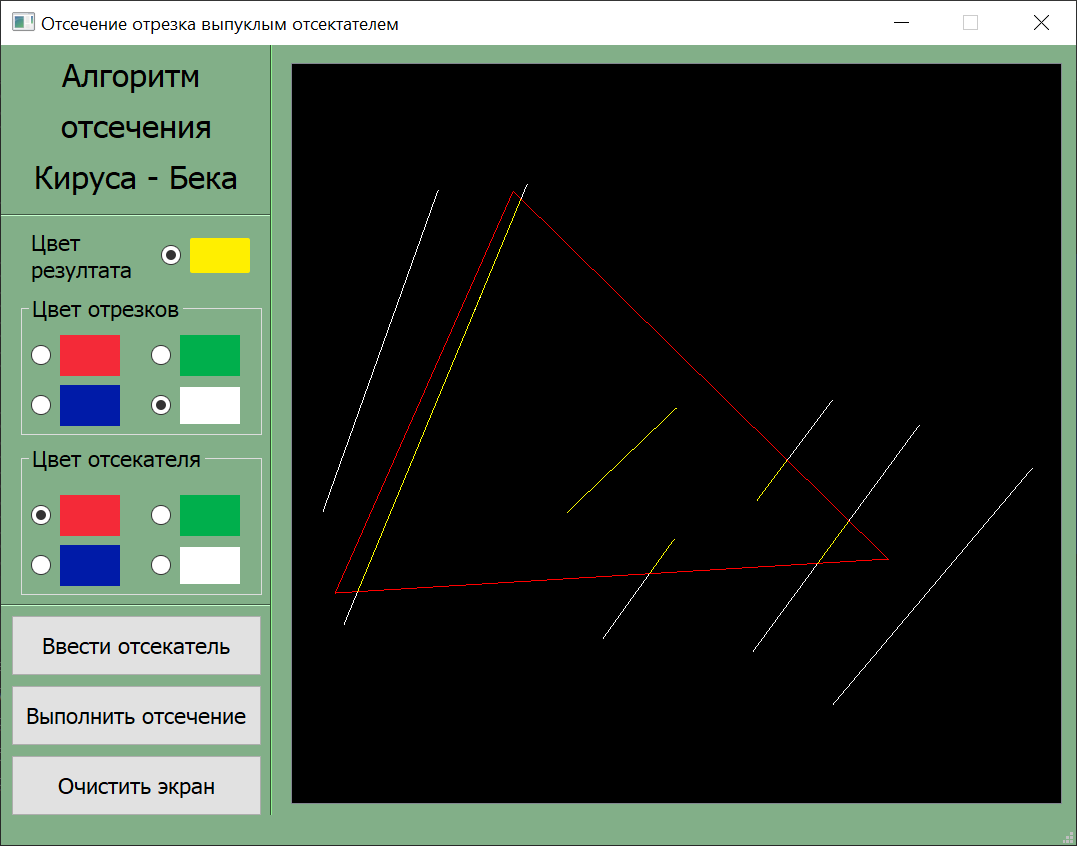
Ниже приведены примеры работы программы в различных ситуациях:









**Заключение**

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