|  |  |
| --- | --- |
| **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 \_ Реализация алгоритма отсечения произвольного многоугольника выпуклым отсекателем.**  (Алгоритм Сазерленда-Ходжмена)**.**  **Студент \_Чаушев Александър Красимиров\_\_\_\_\_\_\_\_**  **Группа \_ИУ7-46Б\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_Куров А. В.\_\_\_\_** |  |

Москва.

2020 г.

**Цель работы:** Изучение и программная реализация алгоритма Сазерленда-Ходжмена отсечения многоугольников.

Необходимо обеспечить ввод отсекателя – произвольного многоугольника.

**Входные данные:** Координаты вершин многоугольника, цвет .

**Выходные данные:** Пользовательское меню, содержащее поля ввода и конечннoe изображение. Вывод замера времени выполнения алгоритма( без задержки ).

**Ошибочные ситуации:** Программа прекращается, если хотя бы один из входных данных не корректен.

**Теоретическая часть:**

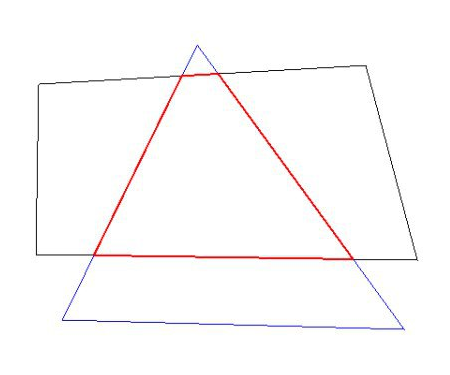
В алгоритме используется стратегия «разделяй и властвуй», которая позволяет решение общей задачи свести к решению ряда простых и похожих подзадач. Примером такой подзадачи является отсечение многоугольника относительно одной отсекающей границы.

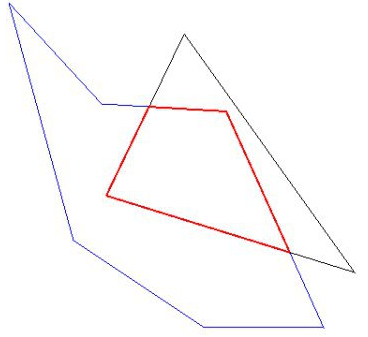
Алгоритм.

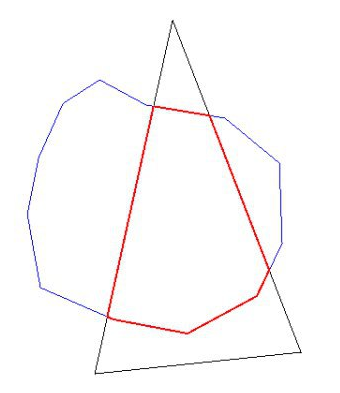
1. Oпределить видимость точки (вершины рёбер)
2. Нахождение точек пересечения. Убедиться в ее наличии
3. Занесение видимых вершин и найденных точек в результирующий список.

Данный алгоритм имеет недостаток — можно столкнуться с ситуацией построения «ложных ребер». Мы работаем с массивом вершин, вершины обходятся последовательно — ложным будет, то ребро, которое обходится два раза.

**Пример Работы:**







**Код Программы:**

def isConvex(edges):  
 # 1. Находим знаки  
 signs = []  
  
 for i in range(1, len(edges) - 1):  
 vo = edges[i - 1] # iая вершина  
 vi = edges[i] # i+1 вершина  
 vn = edges[i + 1] # i+2 вершина и все остальные  
  
 r = vectP(vo, vi, vn) # векторное произведение векторов (i, i+1) и (i, i+2)  
 signs.append(sign(r)) # знак  
  
 vo = edges[len(edges) - 1] # iая вершина  
 vi = edges[0] # i+1 вершина  
 vn = edges[1] # i+2 вершина и все остальные  
  
 r = vectP(vo, vi, vn) # векторное произведение векторов (i, i+1) и (i, i+2)  
 signs.append(sign(r)) # знак  
  
 # 2. Проверяем знаки  
  
 return checkSigns(signs)  
  
def checkSigns(signs: list):  
 '''Returns 0 if its convex else 1 if signs are positive(clockwise) or -1 if they are neg'''  
 print(signs)  
 # counters  
 zeros = 0  
 positive = 0   
 negative = 0  
   
 for i in range(len(signs)):  
 curr = signs[i]  
  
 if curr >= 0:  
 positive += 1  
 if curr <= 0:  
 negative += 1  
 if curr == 0:  
 zeros += 1  
 print("positives {}".format(positive))  
 print("zeros {}".format(zeros))  
  
 if zeros == len(signs):  
 return 0  
 if positive == (len(signs)):  
 return 1  
 if negative == (len(signs)):  
 return -1  
 return 0  
  
def lineClipping(win):  
 if not win.isClipper:  
 msg = QMessageBox()  
 msg.setText("Ошибка!")  
 msg.setInformativeText("Отсекатель не задан!")  
 msg.setWindowTitle("Ошибка ввода!")  
 msg.exec\_()  
 return  
 if not win.isFigure:  
 msg = QMessageBox()  
 msg.setText("Ошибка!")  
 msg.setInformativeText("Многоугольник не задан!")  
 msg.setWindowTitle("Ошибка ввода!")  
 msg.exec\_()  
 return  
 clockwise = isConvex(win.clipper)  
 if clockwise == 0:  
 msg = QMessageBox()  
 msg.setText("Ошибка!")  
 msg.setInformativeText("Отсекатель вогнутый!")  
 msg.setWindowTitle("Ошибка ввода!")  
 msg.exec\_()  
 return  
 p = sutherland\_hodgman(win.clipper, win.figure, clockwise)  
 if p:  
 p.append(p[0])  
 for i in range(len(p) - 1):  
 win.scene.addLine(p[i][0], p[i][1], p[i + 1][0], p[i + 1][1], QPen(win.clip\_color, 2))  
  
  
def is\_visible(point, peak1, peak2, clockwise):  
 v = vectP(point, peak2, peak1)  
 if clockwise \* v <= 0:  
 return True  
 else:  
 return False  
  
  
def is\_intersection(ed1, ed2, norm):  
 vis1 = is\_visible(ed1[0], ed2[0], ed2[1], norm)  
 vis2 = is\_visible(ed1[1], ed2[0], ed2[1], norm)  
 if (vis1 and not vis2) or (not vis1 and vis2):  
 p1 = ed1[0]  
 p2 = ed1[1]  
  
 q1 = ed2[0]  
 q2 = ed2[1]  
 delta = (p2[0] - p1[0]) \* (q1[1] - q2[1]) - (q1[0] - q2[0]) \* (p2[1] - p1[1])  
 delta\_t = (q1[0] - p1[0]) \* (q1[1] - q2[1]) - (q1[0] - q2[0]) \* (q1[1] - p1[1])  
  
 if abs(delta) <= 1e-6:  
 return p2  
  
 t = delta\_t / delta  
  
 I = []  
 I.append(ed1[0][0] + (ed1[1][0] - ed1[0][0]) \* t)  
 I.append(ed1[0][1] + (ed1[1][1] - ed1[0][1]) \* t)  
 return I  
 else:  
 return False  
  
  
def sutherland\_hodgman(clipper, polygon, clockwise):  
 clipper.append(clipper[0])  
  
 s = None  
 f = None  
 # цикл по вершинам отсекателя - 1  
 for i in range(len(clipper) - 1):  
 new = []  
 for j in range(len(polygon)): # цикл по вершинам многоугольника  
 if j == 0:  
 f = polygon[j]  
 else:  
 t = is\_intersection([s, polygon[j]], [clipper[i], clipper[i + 1]], clockwise) # пересечение s, p[j] и c[i], c[i+1]  
 if t: # если есть  
 new.append(t)  
  
 s = polygon[j]  
 if is\_visible(s, clipper[i], clipper[i + 1], clockwise): # видимость вершины многоугольника относительно ребра отсекателя  
 new.append(s)  
  
 if len(new) != 0:  
 t = is\_intersection([s, f], [clipper[i], clipper[i + 1]], clockwise) # точка пересечения s, f и c[i], c[i+1]  
 if t:  
 new.append(t)  
  
 polygon = copy.deepcopy(new) # новый многоугольник  
  
 if len(polygon) == 0:  
 return False  
 else:  
 return polygon