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

**по дисциплине «Компьютерная графика»**

|  |  |
| --- | --- |
| **ОТСЕЧЕНИЕ ПРОИЗВОЛЬНОГО МНОГОУГОЛЬНИКА ВЫПУКЛЫМ ОТСЕКАТЕЛЕМ**  **(АЛГОРИТМ САЗЕРЛЕНДА-ХОДЖМЕНА)**  **Студент:** Челядинов Илья  **Группа:** ИУ7-43Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель:** Куров А. В. |  |

Москва.

2020 г.

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

РЕЗУЛЬТАТ: должна быть разработана программа, позволяющая осуществлять ввод отсекателя, отсекаемого многоугольника и выполнять отсечение многоугольника по границам отсекателя.

Необходимо обеспечить ввод отсекателя – произвольного многоугольника. Высветить его первым цветом. Также необходимо обеспечить ввод отсекаемого многоугольника (высветить вторым цветом). Должна присутствовать проверка отсекателя на выпуклость. Должен быть предусмотрен ввод вершин многоугольника в произвольных точках ребер отсекателя (включая его вершины)

Ввод осуществлять с помощью мыши и нажатия других клавиш.

Выполнить отсечение многоугольника, показав результат третьим цветом. Исходный многоугольник не удалять.

**Теоретический материал**

Алгоритм Сазерленда-Ходжмена позволяет провести отсечение произвольного многоугольника по границам выпуклого отсекателя. Идея алгоритма достаточно проста.

Обычно исходный многоугольник задается количеством своих вершин N(в порядке их обхода ) и их координатами P1, P2, . . ., PN(две последовательные вершины являются концами одного ребра, например, P1P2, P2P.3, ... PnP1.). Отсекатель также задается количеством вершин M и их координатами C1, C 2, . .., Cm..

Исходный многоугольник отсекается сначала, например, левой стороной отсекателя (выберем направление по часовой стрелке), в результате чего получается промежуточный многоугольник. Этот многоугольник отсекает- ся далее верхней границей отсекателя, получается второй промежуточный многоугольник. Далее процесс продолжается аналогичным образом пока не будет выполнено отсечение последней границей отсекателя.

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.

Алгоритм Сазерленда-Ходжмена может быть представлен следующим образом.

1. Ввод исходных данных: Np - количества вершин отсекаемого многоугольника, P - массива координат вершин отсекаемого многоугольника, Nc - количества вершин отсекателя, C - массива координат вершин отсекателя. Элементами массивов являются записи, каждая из которых содержит два поля - координаты x и y вершины. Для удобства работы алгоритма первая вершина отсекателя заносится в массив C дважды: на первое место и еще раз в конец массива (это сделано потому, что последнее ребро отсекателя образуется последней и первой вершинами многоугольника).

2. Цикл по всем ребрам отсекателя (переменная цикла i изменяется от 1 до Nc.

2.1. Обнуление количества вершин результирующего многоугольника Nq.

2.2. Цикл по всем ребрам отсекаемого многоугольника (переменная цикла j изменяется от 1 до Nc ).

2.2.1. Анализ номера обрабатываемой вершины многоугольника: если j=1 (первая вершина), то ее координаты запоминаются в переменной F (F=P1). Переход к п. 2.2.5.

2.2.2. Определение факта пересечения ребра многоугольника SPj и ребра отсекателя CjCj+1.

2.2.3. Если пересечение ребер многоугольников установлено, то определение координат точки T пересечения этих ребер, иначе переход к п. 2.2.5.

2.2.4. Увеличение на единицу количества вершин результирующего многоугольника Nq= Nq+1. Занесение в массив координат результирующего многоугольника координат найденной точки Q(Nq)=T.

2.2.5. Изменение начальной точки ребра многоугольника: присвоение переменной S значения переменной Pj : S = Pj .

2.2.6. Проверка видимости вершины S относительно ребра CjCj+1. Если вершина видима, то занесение ее координат в массив Q: Nq= Nq+1; Q(Nq)=S.

2.2.7. Конец цикла по переменной j (цикл отсечения ребер многоугольника по текущей границе отсекателя).

2.3. Проверка ненулевого количества вершин в результирующем массиве: если Nq=0, то переход к п. (многоугольник невидим относительно текущей границы отсекателя, следовательно, он невидим относительно всего отсекателя).

2.4. Проверка факта пересечения ребра многоугольника SF с ребром отсекателя CjCj+1.

2.5.Если пересечение ребер многоугольников установлено, то определение координат точки T пересечения этих ребер, иначе переход к п. 2.

2.6.Увеличение на единицу количества вершин результирующего много угольника Nq= Nq+1. Занесение в массив координат результирующего многоугольника координат найденной точки Q(Nq)=T.

2.7. Присвоение полученных значений количества вершин и их координат результирующего многоугольника значениям количества вершин и их координат исходного многоугольника: Np =Nq , P=Q (полученный многоугольник отсекается далее следующей стороной отсекателя).

2.8.Конец цикла по переменной i (цикл отсечения по всем границам отсекателя).

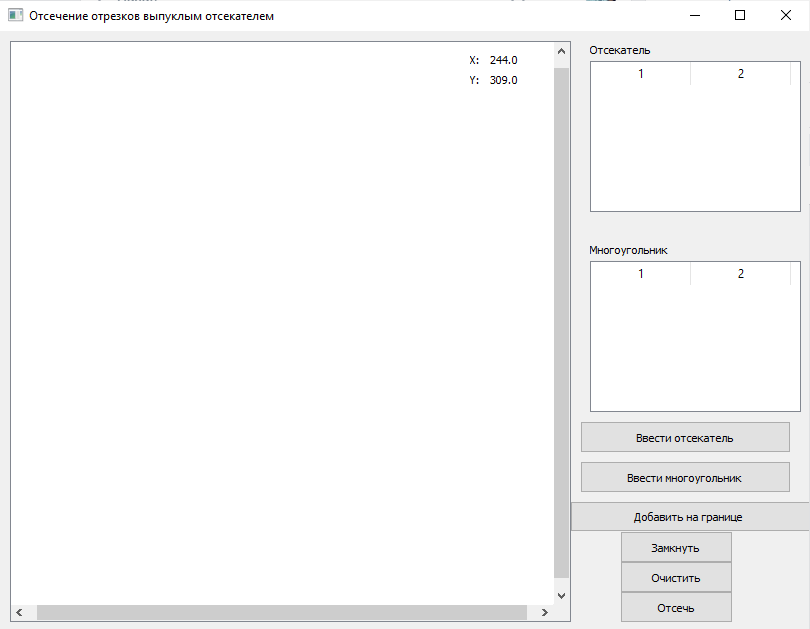
2.9.Визуализация полученного многоугольника P.

2.10. Конец алгоритма.

**Код программы:**

**Интерфейс и примеры работы.**

Интерфейс:



**Примеры работы:**

**def** sign(x):  
 **if not** x:  
 **return** 0  
 **else**:  
 **return** x / abs(x)  
  
  
**def** set\_pol(win):  
 **if** win.input\_pol:  
 win.input\_pol = **False** win.rect.setDisabled(**False**)  
 win.erase.setDisabled(**False**)  
 win.paint.setDisabled(**False**)  
 win.ect.setDisabled(**False**)  
 **else**:  
 win.input\_pol = **True** win.rect.setDisabled(**True**)  
 win.erase.setDisabled(**True**)  
 win.paint.setDisabled(**True**)  
 win.ect.setDisabled(**True**)  
  
  
**def** set\_rect(win):  
 **if** win.input\_clip:  
 win.input\_clip = **False** win.poly.setDisabled(**False**)  
 win.erase.setDisabled(**False**)  
 win.paint.setDisabled(**False**)  
 win.ect.setDisabled(**False**)  
 **else**:  
 win.input\_clip = **True** win.poly.setDisabled(**True**)  
 win.erase.setDisabled(**True**)  
 win.paint.setDisabled(**True**)  
 win.ect.setDisabled(**True**)  
  
  
**def** add\_point(point):  
 **global** w  
 **if** w.input\_clip:  
 w.pen.setColor(black)  
 **if** w.point\_now\_clip **is None**:  
 w.point\_now\_clip = point  
 w.point\_lock\_clip = point  
 add\_row(w.table\_rect)  
 i = w.table\_rect.rowCount() - 1  
 item\_x = QTableWidgetItem(**"{0}"**.format(point.x()))  
 item\_y = QTableWidgetItem(**"{0}"**.format(point.y()))  
 w.table\_rect.setItem(i, 0, item\_x)  
 w.table\_rect.setItem(i, 1, item\_y)  
 **else**:  
 w.clip.append(point)  
 w.point\_now\_clip = point  
 add\_row(w.table\_rect)  
 i = w.table\_rect.rowCount() - 1  
 item\_x = QTableWidgetItem(**"{0}"**.format(point.x()))  
 item\_y = QTableWidgetItem(**"{0}"**.format(point.y()))  
 w.table\_rect.setItem(i, 0, item\_x)  
 w.table\_rect.setItem(i, 1, item\_y)  
 item\_x = w.table\_rect.item(i-1, 0)  
 item\_y = w.table\_rect.item(i-1, 1)  
 w.scene.addLine(point.x(), point.y(), float(item\_x.text()), float(item\_y.text()), w.pen)  
  
 **if** w.input\_pol:  
 w.pen.setColor(blue)  
 **if** w.point\_now\_pol **is None**:  
 w.point\_now\_pol = point  
 w.point\_lock\_pol = point  
 add\_row(w.table\_pol)  
 i = w.table\_pol.rowCount() - 1  
 item\_x = QTableWidgetItem(**"{0}"**.format(point.x()))  
 item\_y = QTableWidgetItem(**"{0}"**.format(point.y()))  
 w.table\_pol.setItem(i, 0, item\_x)  
 w.table\_pol.setItem(i, 1, item\_y)  
 **else**:  
 w.pol.append(point)  
 w.point\_now\_pol = point  
 add\_row(w.table\_pol)  
 i = w.table\_pol.rowCount() - 1  
 item\_x = QTableWidgetItem(**"{0}"**.format(point.x()))  
 item\_y = QTableWidgetItem(**"{0}"**.format(point.y()))  
 w.table\_pol.setItem(i, 0, item\_x)  
 w.table\_pol.setItem(i, 1, item\_y)  
 item\_x = w.table\_pol.item(i-1, 0)  
 item\_y = w.table\_pol.item(i-1, 1)  
 w.scene.addLine(point.x(), point.y(), float(item\_x.text()), float(item\_y.text()), w.pen)  
  
  
**def** lock(win):  
 **if** w.input\_pol:  
 win.pol.append(win.point\_lock\_pol)  
 win.scene.addLine(win.point\_now\_pol.x(), win.point\_now\_pol.y(), win.point\_lock\_pol.x(), win.point\_lock\_pol.y(), w.pen)  
 win.point\_now\_pol = **None  
  
 if** w.input\_clip:  
 win.clip.append(win.point\_lock\_clip)  
 win.scene.addLine(win.point\_now\_clip.x(), win.point\_now\_clip.y(), win.point\_lock\_clip.x(), win.point\_lock\_clip.y(), w.pen)  
 win.point\_now\_clip = **None  
  
  
def** add\_row(win\_table):  
 win\_table.insertRow(win\_table.rowCount())  
  
  
**def** clean\_all(win):  
 win.scene.clear()  
 win.table\_rect.clear()  
 win.table\_pol.clear()  
 win.clip = []  
 win.pol = []  
 win.point\_now\_clip = **None** win.point\_now\_pol = **None** win.point\_lock\_clip = **None** win.point\_lock\_pol = **None** r = win.table\_rect.rowCount()  
 **for** i **in** range(r, -1, -1):  
 win.table\_rect.removeRow(i)  
  
 r = win.table\_pol.rowCount()  
 **for** i **in** range(r, -1, -1):  
 win.table\_pol.removeRow(i)  
  
  
**def** isConvex(edges):  
 flag = 1  
  
 *# начальные вершины* vo = edges[0] *# iая вершина* vi = edges[1] *# i+1 вершина* vn = edges[2] *# i+2 вершина и все остальные  
  
 # векторное произведение двух векторов* x1 = vi.x() - vo.x()  
 y1 = vi.y() - vo.y()  
  
 x2 = vn.x() - vi.x()  
 y2 = vn.y() - vi.y()  
  
 *# определяем знак ординаты* r = x1 \* y2 - x2 \* y1  
 prev = sign(r)  
  
 **for** i **in** range(2, len(edges) - 1):  
 **if not** flag:  
 **break** vo = edges[i - 1]  
 vi = edges[i]  
 vn = edges[i + 1]  
  
 *# векторное произведение двух векторов* x1 = vi.x() - vo.x()  
 y1 = vi.y() - vo.y()  
  
 x2 = vn.x() - vi.x()  
 y2 = vn.y() - vi.y()  
  
 r = x1 \* y2 - x2 \* y1  
 curr = sign(r)  
  
 *# если знак предыдущей координаты не совпадает, то возможно многоугольник невыпуклый* **if** curr != prev:  
 flag = 0  
 prev = curr  
  
 *# не забываем проверить последнюю с первой вершины* vo = edges[len(edges) - 1]  
 vi = edges[0]  
 vn = edges[1]  
  
 *# векторное произведение двух векторов* x1 = vi.x() - vo.x()  
 y1 = vi.y() - vo.y()  
  
 x2 = vn.x() - vi.x()  
 y2 = vn.y() - vi.y()  
  
 r = x1 \* y2 - x2 \* y1  
 curr = sign(r)  
 **if** curr != prev:  
 flag = 0  
  
 **return** flag \* curr  
  
  
**def** is\_intersection(ed1, ed2, norm):  
 vis1 = is\_visiable(ed1[0], ed2[0], ed2[1], norm)  
 vis2 = is\_visiable(ed1[1], ed2[0], ed2[1], norm)  
 **if** (vis1 **and not** vis2) **or** (**not** vis1 **and** vis2):  
 *# ищем пересечение* p1 = ed1[0]  
 p2 = ed1[1]  
  
 q1 = ed2[0]  
 q2 = ed2[1]  
  
 delta = (p2.x() - p1.x()) \* (q1.y() - q2.y()) - (q1.x() - q2.x()) \* (p2.y() - p1.y())  
 delta\_t = (q1.x() - p1.x()) \* (q1.y() - q2.y()) - (q1.x() - q2.x()) \* (q1.y() - p1.y())  
  
 **if** abs(delta) <= 1e-6:  
 **return** p2  
  
 t = delta\_t / delta  
  
 I = QPointF()  
 I.setX(ed1[0].x() + (ed1[1].x() - ed1[0].x()) \* t)  
 I.setY(ed1[0].y() + (ed1[1].y() - ed1[0].y()) \* t)  
 **return** I  
 **else**:  
 **return False  
  
  
def** is\_visiable(point, peak1, peak2, norm):  
 v = vector([point, peak1], [peak2, peak1])  
 **if** norm \* v < 0:  
 **return True  
 else**:  
 **return False  
  
  
def** vector(v1, v2):  
 x1 = v1[0].x() - v1[1].x()  
 y1 = v1[0].y() - v1[1].y()  
  
 x2 = v2[0].x() - v2[1].x()  
 y2 = v2[0].y() - v2[1].y()  
  
 **return** x1 \* y2 - x2 \* y1  
  
  
  
**def** clipping(win):  
 **if** len(win.clip) <= 1:  
 QMessageBox.warning(win, **"Ошибка!"**, **"Отсекатель не задан!"**)  
  
 **if** len(win.pol) <= 1:  
 QMessageBox.warning(win, **"Ошибка!"**, **"Многоугольник не задан!"**)  
  
 **if** len(win.pol) > 1 **and** len(win.clip) > 1:  
 norm = isConvex(win.clip)  
 **if not** norm:  
 QMessageBox.warning(win, **"Ошибка!"**, **"Отсекатель не выпуклый!Операция не может быть проведена!"**)  
 **else**:  
 p = sutherland\_hodgman(win.clip, win.pol, norm)  
 **if** p:  
 win.pen.setWidth(2)  
 win.pen.setColor(red)  
 win.scene.addPolygon(p, win.pen)  
 win.pen.setWidth(1)  
  
  
**def** sutherland\_hodgman(clip, pol, norm):  
 *# дублируем начальную вершину отсекателя в конец* clip.append(clip[0])  
  
 s = **None** f = **None** *# цикл по вершинам отсекателя* **for** i **in** range(len(clip) - 1):  
 new = [] *# новый массив вершин* **for** j **in** range(len(pol)): *# цикл по вершинам многоугольника* **if** j == 0:  
 f = pol[j]  
 **else**:  
 t = is\_intersection([s, pol[j]], [clip[i], clip[i + 1]], norm)  
 **if** t:  
 new.append(t)  
  
 s = pol[j]  
 **if** is\_visiable(s, clip[i], clip[i + 1], norm):  
 new.append(s)  
  
 **if** len(new) != 0:  
 t = is\_intersection([s, f], [clip[i], clip[i + 1]], norm)  
 **if** t:  
 new.append(t)  
  
 pol = copy.deepcopy(new)  
  
 **if** len(pol) == 0:  
 **return False  
 else**:  
 **return** QPolygonF(pol)

