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

**по дисциплине «Компьютерная графика»**

|  |  |
| --- | --- |
| **Тема: Реализация алгоритма отсечения отрезка произвольным выпуклым отсекателем (Алгоритм Кируса-Бека).**  **Студент:** Челядинов Илья  **Группа:** ИУ7-43Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель:** Куров А. В. |  |

Москва.

2020 г.

Цель работы: изучение и программная реализация алгоритма отсечения отрезка произвольным выпуклым отсекателем.

Необходимо обеспечить ввод отсекателя – произвольного многоугольника. Высветить его первым цветом. Также необходимо обеспечить ввод нескольких (до десяти) различных отрезков (высветить их вторым цветом). Отрезки могут иметь произвольное расположение: горизонтальные, вертикальные, имеющие произвольный наклон.

Предусмотреть ввод отрезков, параллельных границе отсекателя.

Ввод осуществлять с помощью мыши и нажатия других клавиш.

Выполнить отсечение отрезков, показав результат третьим цветом. Исходные отрезки не удалять.

**Теоретический материал**

Отрезки описываются формулой:

P(t) = P1 + (P2- P1) \* t; где t - параметр (0 <= t <= 1).

Векторное уравнение, которое сводится к двум параметрическим уравнениям:

Px(t) = P1.x + (P2.x- P1.x) \* t

Py(t) = P1.y+ (P2.y- P1.y) \* t

параметр t так же принадлежит [0, 1]

При нахождении пересечений с гранями многоугольника находится значение параметра t, при котором происходит пересечение, и если значение параметра лежит вне допустимых границ, то пересечение происходит с продолжением отрезка (получается, что его нет). Такие пересечения отвергаются.

В данном алгоритме используется понятие "вектора внутренней нормали" nв. Вектор внутренней нормали - вектор, перпендикулярный грани многоугольника и направлен внутрь этого многоугольника. Этот факт проверяется аналитическим выражением след. вида: nв \* (B - A) >= 0, где А - точка грани, из которой исходит данная нормаль, а В любая другая точка нормали, однако в качестве точки В следует брать точку, не принадлежащую рассматриваемой грани, иначе что для внутренней, что для внешней нормали скалярное произведение будет равно 0.

Так же в алгоритме представляет интерес следующий вектор:

[P(t) - fi], где fi - произвольная точка рассматриваемой грани (не совпадающая с точкой пересечения рассматриваемых грани и отрезка)

Проанализируем скалярное произведение этого вектора и вектора нормали к рассматриваемой грани:

nв[P(t) - fi] > 0 - вектор направлен внутрь области многоугольника (так как из скалярного произведения следует, что модуль угла между этим вектором и вектором внутренней нормали < 90).

nв[P(t) - fi] = 0 - вектор перпендикулярен нормали (то есть параллелен грани)

nв[P(t) - fi] < 0 - направлен вне области многоугольника (противоположность первой ситуации)

Очевидно, что при различных параметрах t, рассматриваемая точка P(t) может находиться как внутри, так и вне области многоугольника относительно рассматриваемой грани, однако в данном случае нас больше интересует (и понадобится) тот факт, что мы можем определить "входит" или "выходит" отрезок из многоугольника при пересечении определенной грани. Это выясняется так: если отрезок пересекает грань и его начало было внутри многоугольника относительно этой грани (анализ скалярного произведение рассмотренного выше при P(t), t = 0), то получается что при пересекании он выйдет за грань (получается, он выходит за пределы многоугольника).

Теперь рассмотрим еще лучше ситуацию: nв[P(t) - fi] = 0. Мы уже сказали, что она соответствует ситуации, когда вектор, состоящий из точки отрезка (P(t)) и точки fi грани параллелен этой грани. Очевидно, что если стоит задача построить параллельную прямую к некоторой прямой L через точку А, находящейся на этой прямой L, то решение этой задачи - прямая, совпадающая с прямой L. Из этого следует, что nв[P(t) - fi] = 0 выполняется тогда, когда вектор лежит на одной прямой с гранью [P(t) - fi], а P(t) для некоторого t - точка пересечения грани и отрезка.

Подставим параметрическую форму уравнения в данное выражение:

nв[P1 + (P2- P1)t - fi] = 0

Преобразуем: nв[P1 - fi] + nв[P2- P1]t = 0 (\*)

В данном уравнении вектор [P2- P1] - вектор, определяющий направление (ориентацию) отрезка, а вектор [P1 - fi] рассматривался выше в общем виде, но в данном случае - вектор, соединящий некоторую точку грани с началом отрезка (по его скалярному произведению с внутренней нормалью можно судить о положении отрезка относительно внутренней нормали)

Введем обозначения (почти такие же обозначения указаны в коде):

Wi = nв[P1 - fi]

Dск = nв[P2- P1]

Выразим t из уравнения (\*), используя обозначения выше:

t = -Wi / Dск

Данное выражение нельзя рассматривать при Dск = 0. Рассмотрим случаи, когда Dск = 0:

1. вектор ориентации отрезка вырожден (нулевой). Такой случай нас не очень интересует.

2. Dcк перпендикулярен nв. Данный случай инетресен: получается, что отрезок парллелен грани. Здесь может быть 2 случая: отрезок лежит вне многоугольника относительно грани - тогда можно однозначно сделать вывод о том, что он не видим и закончить операцию отсечения данного отрезка. Отрезок лежит внутри многоугольника относительно грани - тогда следует перейти на следующую итерацию и продолжить операцию отсечения. Определить "вне" или "внутри" довольно-таки легко с помощью вектора Wi. Этот вектор начинается в некоторой точке рассматриваемой грани многоугольника и заканчивается в некоторой точке отрезка, таким образом можно сказать, что он направлен "от грани к отрезку". Внутренняя нормаль начинается в некоторой точке грани и может быть направлена к отрезку и в противоположную сторону. Проверяется это, например, вот таким скалярным произведением: Wi = nв[P1 - fi] (это тот самый коэффициент, который мы уже выразили и посчитали выше). Если скалярное произведение < 0, то угол между вектором нормали и вектором, направленным к отрезку > 90, и вектор лежит вне фигуры, иначе - внутри.

Теперь когда мы научились находить значения t всех пересечений с гранями, а так же научились определять "входит" отрезок в многоугольник или нет, осталось решить только одну проблему: какие конкретно значения t выбрать в качестве начального и конечного. Очевидно, что если отрезок виден, то он виден относительно всех граней. Из этого можно сделать вывод, что если отрезок входит в многоугольник относительно какой-то грани, то относительно других граней он должен был уже войти (то есть надо выбирать в качестве начала последний вход), и при этом не должен был выйти (то есть получается последний вход должен быть раньше всех выходов). С выходами ситуация та же: выйдя за первую грань, отрезок перестанет быть виден относительно нее и, следовательно, будет на оставшемся промежутке (из этого вывод - в качестве конца следует брать первых выход). Далее, убедившись, что параметр tвх , соответствующий последнему входу, меньше, чем параметр tвых, соответствующий первому выходу, чертим видимую часть отрезка (если условие не выполняется - не чертим).

**Код программы:**

**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)

**def** add\_point(point):  
 **global** w  
 **if** w.input\_rect:  
 w.pen.setColor(black)  
 **if** w.point\_now\_rect **is None**:  
 w.point\_now\_rect = point  
 w.point\_lock = 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.edges.append(point)  
 w.point\_now\_rect = 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\_bars:  
 w.pen.setColor(red)  
 **if** w.point\_now\_bars **is None**:  
 w.point\_now\_bars = point  
 **else**:  
 w.lines.append([[w.point\_now\_bars.x(), w.point\_now\_bars.y()],  
 [point.x(), point.y()]])  
  
 add\_row(w.table\_bars)  
 i = w.table\_bars.rowCount() - 1  
 item\_b = QTableWidgetItem(**"[{0}, {1}]"**.format(w.point\_now\_bars.x(), w.point\_now\_bars.y()))  
 item\_e = QTableWidgetItem(**"[{0}, {1}]"**.format(point.x(), point.y()))  
 w.table\_bars.setItem(i, 0, item\_b)  
 w.table\_bars.setItem(i, 1, item\_e)  
 w.scene.addLine(w.point\_now\_bars.x(), w.point\_now\_bars.y(), point.x(), point.y(), w.pen)  
 w.point\_now\_bars = **None**

**def** scalar(v1, v2):  
 **return** v1.x() \* v2.x() + v1.y() \* v2.y()

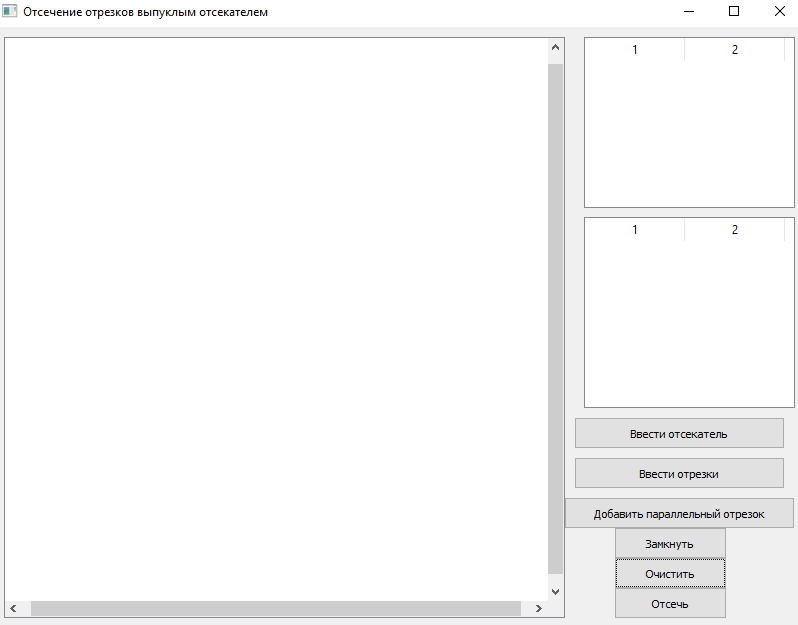
**def** clipping(win):  
 norm = isConvex(win.edges)  
 **if not** norm:  
 QMessageBox.warning(win, **"Ошибка!"**, **"Отсекатель не выпуклый!Операция не может быть проведена!"**)  
  
 **for** b **in** win.lines:  
 win.pen.setColor(blue)  
 cyrus\_beck(b, win.edges, norm, win.scene, win.pen)  
 win.pen.setColor(red)  
  
  
**def** sign(x):  
 **if not** x:  
 **return** 0  
 **else**:  
 **return** x / abs(x)  
  
  
**def** lock(win):  
 win.edges.append(win.point\_lock)  
 win.scene.addLine(win.point\_now\_rect.x(), win.point\_now\_rect.y(), win.point\_lock.x(), win.point\_lock.y(), w.pen)  
 win.point\_now\_rect = **None  
  
  
def** add\_bars(win):  
 **if** len(win.edges) == 0:  
 QMessageBox.warning(win, **"Внимание!"**, **"Не введен отсекатель!"**)  
 **return** win.pen.setColor(red)  
 w.lines.append([[win.edges[0].x() - 15, win.edges[0].y() - 15],  
 [win.edges[1].x() - 15, win.edges[1].y() - 15]])  
 add\_row(w.table\_bars)  
 i = w.table\_bars.rowCount() - 1  
 item\_b = QTableWidgetItem(**"[{0}, {1}]"**.format(win.edges[0].x() - 15 , win.edges[0].y() - 15))  
 item\_e = QTableWidgetItem(**"[{0}, {1}]"**.format(win.edges[1].x() - 15, win.edges[1].y() - 15))  
 w.table\_bars.setItem(i, 0, item\_b)  
 w.table\_bars.setItem(i, 1, item\_e)  
 w.scene.addLine(win.edges[0].x() - 15, win.edges[0].y() - 15, win.edges[1].x() - 15, win.edges[1].y() - 15, w.pen)  
  
 win.pen.setColor(red)  
 w.lines.append([[win.edges[0].x() + 15, win.edges[0].y() + 15],  
 [win.edges[1].x() + 15, win.edges[1].y() + 15]])  
 add\_row(w.table\_bars)  
 i = w.table\_bars.rowCount() - 1  
 item\_b = QTableWidgetItem(**"[{0}, {1}]"**.format(win.edges[0].x() + 15, win.edges[0].y() + 15))  
 item\_e = QTableWidgetItem(**"[{0}, {1}]"**.format(win.edges[1].x() + 15, win.edges[1].y() + 15))  
 w.table\_bars.setItem(i, 0, item\_b)  
 w.table\_bars.setItem(i, 1, item\_e)  
 w.scene.addLine(win.edges[0].x() + 15, win.edges[0].y() + 15, win.edges[1].x() + 15, win.edges[1].y() + 15, w.pen)  
  
  
**def** clean\_all(win):  
 win.scene.clear()  
 win.table\_rect.clear()  
 win.table\_bars.clear()  
 win.lines = []  
 win.edges = []  
 win.point\_now\_rect = **None** win.point\_now\_bars = **None** win.point\_lock = **None** win.image.fill(Qt.white)  
 r = win.table\_rect.rowCount()  
 **for** i **in** range(r, -1, -1):  
 win.table\_rect.removeRow(i)  
  
 r = win.table\_bars.rowCount()  
 **for** i **in** range(r, -1, -1):  
 win.table\_bars.removeRow(i)  
  
  
**def** add\_row(win\_table):  
 win\_table.insertRow(win\_table.rowCount())  
  
  
**def** set\_bars(win):  
 **if** win.input\_bars:  
 win.input\_bars = **False** win.rect.setDisabled(**False**)  
 win.erase.setDisabled(**False**)  
 win.paint.setDisabled(**False**)  
 win.ect.setDisabled(**False**)  
 win.lock.setDisabled(**False**)  
 **else**:  
 win.input\_bars = **True** win.rect.setDisabled(**True**)  
 win.erase.setDisabled(**True**)  
 win.paint.setDisabled(**True**)  
 win.ect.setDisabled(**True**)  
 win.lock.setDisabled(**True**)  
  
**def** cyrus\_beck(r, edges, n, scene, p):  
 *# инициализируем пределы значений параметра, предполагая, что весь отрезок полностью видимый  
 # максимизируем t нижнее и t верхнее, исходя из того что 0 <= t <= 1* tb = 0  
 te = 1  
  
 *# вычисляем директрису(определяет направление/ориентацию отрезка) D= p1 - p2* D = QPointF()  
 D.setX(r[1][0] - r[0][0])  
 D.setY(r[1][1] - r[0][1])  
  
 *# главный цикл по сторонам отсекателя* **for** i **in** range(len(edges)):  
 *# вычисляем wi, D \* ni, wi \* n  
 # весовой множитель удаленности гранничной точки от р1(берем граничную точку равной вершине)* W = QPointF()  
 W.setX(r[0][0] - edges[i].x())  
 W.setY(r[0][1] - edges[i].y())  
  
 *# определяем нормаль* N = QPointF()  
 **if** i == len(edges) - 1:  
 N.setX(-n \* (edges[0].y() - edges[i].y()))  
 N.setY(n \* (edges[0].x() - edges[i].x()))  
 **else**:  
 N.setX(-n \* (edges[i + 1].y() - edges[i].y()))  
 N.setY(n \* (edges[i + 1].x() - edges[i].x()))  
 *# определяем скалярные произведения* Dscalar = scalar(D, N)  
 Wscalar = scalar(W, N)  
  
 **if** Dscalar == 0:  
 *# если отрезок параллелен ребру отсекателю* **if** Wscalar < 0:  
 *# виден ли?* **return  
 else**:  
 *# отрезок невырожден, определяем t* t = - Wscalar / Dscalar  
 *# поиск верхнего и нижнего предела t* **if** Dscalar > 0:  
 *# поиск нижнего предела  
 # верно ли, что t <= 1* **if** t > 1:  
 **return  
 else**:  
 tb = max(tb, t)  
 **elif** Dscalar < 0:  
 *# поиск верхнего предела  
 # верно ли, что t >= 0* **if** t < 0:  
 **return  
 else**:  
 te = min(te, t)  
  
 *# проверка фактической видимости отрезка* **if** tb <= te:  
 scene.addLine(r[0][0] + (r[1][0] - r[0][0]) \* te,  
 r[0][1] + (r[1][1] - r[0][1]) \* te,  
 r[0][0] + (r[1][0] - r[0][0]) \* tb,  
 r[0][1] + (r[1][1] - r[0][1]) \* tb, p)

**def** set\_rect(win):  
 **if** win.input\_rect:  
 win.input\_rect = **False** win.bars.setDisabled(**False**)  
 win.erase.setDisabled(**False**)  
 win.paint.setDisabled(**False**)  
 win.ect.setDisabled(**False**)  
 win.lock.setDisabled(**False**)  
 **else**:  
 win.input\_rect = **True** win.bars.setDisabled(**True**)  
 win.erase.setDisabled(**True**)  
 win.paint.setDisabled(**True**)  
 win.ect.setDisabled(**True**)

**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** cyrus\_beck(r, edges, n, scene, p):  
 *# инициализируем пределы значений параметра, предполагая, что весь отрезок полностью видимый  
 # максимизируем t нижнее и t верхнее, исходя из того что 0 <= t <= 1* tb = 0  
 te = 1  
  
 *# вычисляем директрису(определяет направление/ориентацию отрезка) D= p1 - p2* D = QPointF()  
 D.setX(r[1][0] - r[0][0])  
 D.setY(r[1][1] - r[0][1])  
  
 *# главный цикл по сторонам отсекателя* **for** i **in** range(len(edges)):  
 *# вычисляем wi, D \* ni, wi \* n  
 # весовой множитель удаленности гранничной точки от р1(берем граничную точку равной вершине)* W = QPointF()  
 W.setX(r[0][0] - edges[i].x())  
 W.setY(r[0][1] - edges[i].y())  
  
 *# определяем нормаль* N = QPointF()  
 **if** i == len(edges) - 1:  
 N.setX(-n \* (edges[0].y() - edges[i].y()))  
 N.setY(n \* (edges[0].x() - edges[i].x()))  
 **else**:  
 N.setX(-n \* (edges[i + 1].y() - edges[i].y()))  
 N.setY(n \* (edges[i + 1].x() - edges[i].x()))  
 *# определяем скалярные произведения* Dscalar = scalar(D, N)  
 Wscalar = scalar(W, N)  
  
 **if** Dscalar == 0:  
 *# если отрезок параллелен ребру отсекателю* **if** Wscalar < 0:  
 *# виден ли?* **return  
 else**:  
 *# отрезок невырожден, определяем t* t = - Wscalar / Dscalar  
 *# поиск верхнего и нижнего предела t* **if** Dscalar > 0:  
 *# поиск нижнего предела  
 # верно ли, что t <= 1* **if** t > 1:  
 **return  
 else**:  
 tb = max(tb, t)  
 **elif** Dscalar < 0:  
 *# поиск верхнего предела  
 # верно ли, что t >= 0* **if** t < 0:  
 **return  
 else**:  
 te = min(te, t)  
  
 *# проверка фактической видимости отрезка* **if** tb <= te:  
 scene.addLine(r[0][0] + (r[1][0] - r[0][0]) \* te,  
 r[0][1] + (r[1][1] - r[0][1]) \* te,  
 r[0][0] + (r[1][0] - r[0][0]) \* tb,  
 r[0][1] + (r[1][1] - r[0][1]) \* tb, p)

**Интерфейс и примеры работы.**

Интерфейс:



**Примеры работы:**

Прямоугольный отсекатель:

