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

**Вариант 26**

|  |  |
| --- | --- |
| **Тема** Реализация алгоритма отсечения отрезка регулярным отсекателем.  **Студент** Челядинов И. Д.  **Группа** ИУ7-43Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Куров А. В. |  |

Москва.

2020 г.

**Цель работы**: изучение и программная реализация алгоритма отсечения отрезка.

**Задание:**

Алгоритм отсечения (простой, Сазерленда-Коэна, деления отрезка пополам) выбирается в соответствии с номером студента в списке группы.

Необходимо обеспечить ввод регулярного отсекателя - прямоугольника. 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.

Ввод осуществлять с помощью мыши и нажатия других клавиш.

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

Простой алгоритм реализуют студенты с номерами 1,4,7 и т.д.

Алгоритм Сазерленда-Коэна реализуют студенты с номерами 2,5,8 и т.д.

Алгоритм разбиения отрезка средней точкой реализуют студенты с номерами 3,6,9 и т.д.

Отсечение - это операция удаления изображения за пределами выделенной области, называемой окном. При отсечении на плоскости в качестве стандартного отсекателя рассматривается прямоугольник со сторонами, параллельными осям координат. Произвольный отсекатель - это обычно многоугольник, который может быть как выпуклым, так и невыпуклым, а также может иметь отверстия. Помимо отсекателя должны быть также известны геометрические характеристики изображенных объектов. В результате отсечения должны получиться геометрические характеристики объектов, остающихся в пределах окна отсечения в результате выполнения рассматриваемой операции.

Точки, лежащие целиком внутри окна, удовлетворяют условию (Xл ⩽ X ⩽ Xп) (Yн ⩽ Y ⩽ Yп), где (X,Y) - координаты точки. Считается, что точки, лежащие на границе окна, принадлежат внутренней области окна. Отрезок целиком лежит внутри окна, если обе его концевые точки лежат внутри окна. Однако обратное утверждение, к сожалению, верно не всегда. Отрезок, концевые точки которого лежат вне окна, может быть как полностью невидимым, так и частично видимым. Полностью невидимым называется отрезок, целиком лежащий вне отсекателя. Частично видимым называется отрезок, одна часть которого лежит в пределах отсекателя, а другая - вне его. Если обе концевые точки отрезка невидимы, то он будет заведомо невидимым, если они (вершины отрезка) одновременно лежат левее или правее или ниже или выше окна.

Так как у меня 26 вариант, я буду реализовать алгоритс Сазерленда-Коэна.

**АЛГОРИТМ ОТСЕЧЕНИЯ САЗЕРЛЕНДА-КОЭНА**

Для удобства работы данные, задающие отсекатель, заносятся в массив "окно" O(4), в котором O1=Xл, O2=Xп, O3=Yн, O4=Yв. Отсечение производится в определенном порядке: левой, правой, нижней, верхней границами отсекателя. Поэтому для отыскания точек пересечения в выражения следует на i-ом шаге подставлять i-ые элементы массива O. Такое задание исходных данных позволяет для горизонтального отрезка не проводить третий и четвертый этапы, а для вертикального отрезка - первый и второй этапы. В алгоритме используется признак (флаг) Fl, определяющий расположение отрезка: Fl= -1 - отрезок вертикальный, Fl=0 - общего положения, Fl=1 - горизонтальный.

Сам алгоритм Сазерленда-Коэна можно представить в следующем виде:

1. Ввод координат отсекателя Xл (O1), Xп (O2), Yн (O3), Yв (O4).

2. Ввод координат концов отрезка P1(X1,Y1), P2(X 2,Y2).

3. Установка начального значения флага Fl=0.

4. Проверка вертикальности отрезка: если P2.x-P1.x=0 (вертикальный), то Fl= -1, иначе вычислить тангенс угла наклона отрезка m=(P2.y -P1.y)/(P2.x-P1.x).

5. Проверка горизонтальности отрезка: если m=0, то Fl=1.

6. Начало цикла по i от 1 до 4 отсечения отрезка по всем четырем сторонам отсекателя.

7. Обращение к алгоритму (подпрограмме) определения видимости отрезка P1P2 относительно заданного окна. Подпрограмма возвращает признак pr, принимающий следующие значения:

pr=1 - отрезок видимый;

pr= -1 - отрезок полностью невидимый;

pr=0 - отрезок может быть частично видимым.

8. Анализ полученного признака видимости:

если pr= -1, то переход к п. 20;

если pr=1, то переход к п. 19.

9. Проверка видимости обеих вершин отрезка относительно текущей i-ой стороны окна: если T1i=T2i , то переход к п.18.

10. Проверка видимости первой вершины: если T1i=0 (вершина видима), то обмен местами вершин: R=P1; P1=P2; P2 =R.

11. Проверка вертикальности отрезка: если Fl= -1, то переход к п. 14.

12. Анализ номера шага отсечения: если i 3, то переход к п. 14.

13. Вычисление координат точки пересечения с i-ым ребром отсекателя (левым или правым): P1.y=m(Oi-P1.x)+P1.y; P1.x=Oi . Переход к п. 18.

14. Проверка горизонтальности отрезка: если Fl=1, то переход к п. 18.

15. Проверка вертикальности отрезка: если Fl=-1, то переход к п.17.

16. Вычисление абсциссы точки пересечения отрезка общего положения со стороной отсекателя (верхней или нижней): P1.x=(Oi-P1.y)/m+P1.x .

17. Присвоение ординате вершины отрезка ординаты стороны отсекателя: P1.y=Oi .

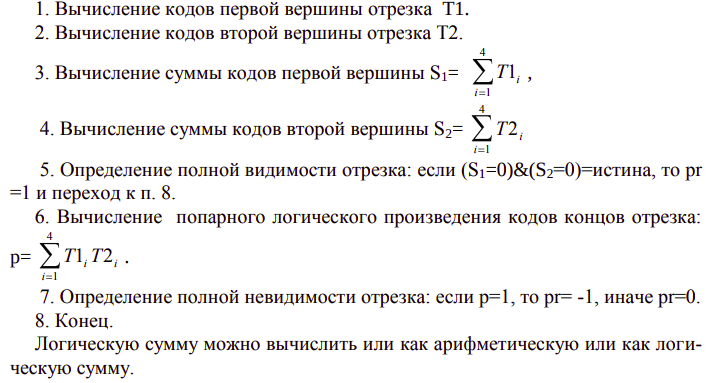
18. Конец цикла по i (вычисление нового значения параметра цикла i=i+1, анализ его значения и переход на повторное выполнение цикла или выход из цикла).

19. Вычерчивание отрезка P1P2.

20. Конец.

При работе с вертикальными отрезками нет необходимости определять номер шага отсечения. Невидимый вертикальный отрезок относительно бокового ребра отсекателя будет полностью невидимым, это обнаружится в самом начале. Если же он видим относительно бокового ребра, то на первых двух шагах автоматически произойдет переход к следующему шагу цикла.

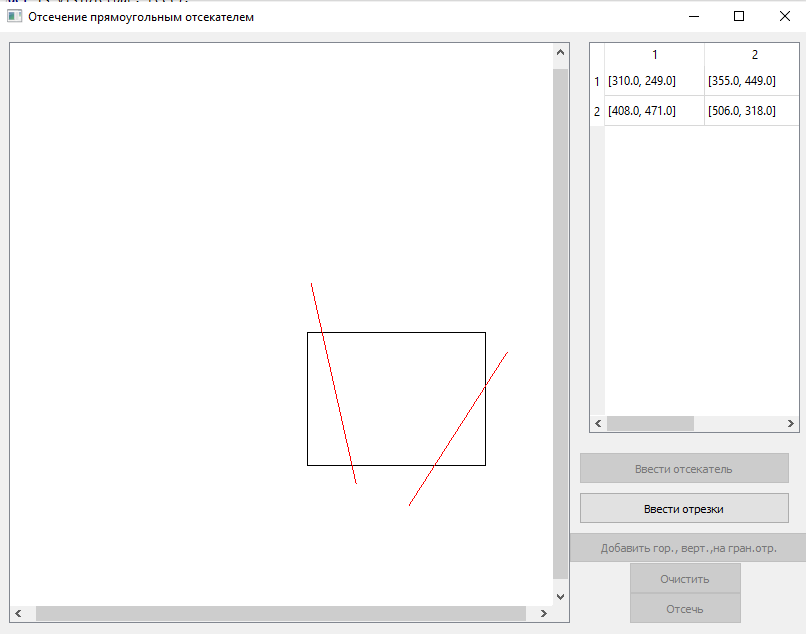
Алгоритм определения видимости отрезка относительно окна можно представить следующим образом.

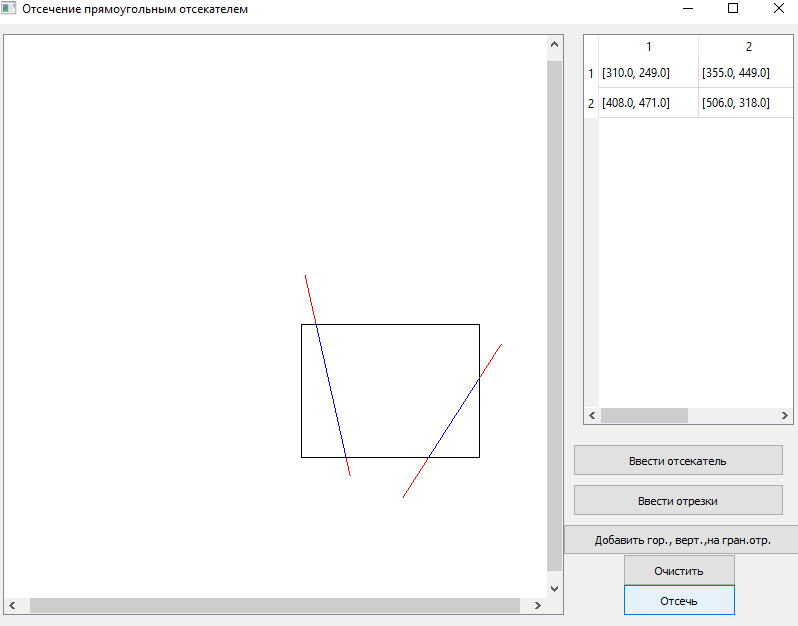


**Код программы:**

**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**)  
 **else**:  
 win.input\_bars = **True** win.rect.setDisabled(**True**)  
 win.erase.setDisabled(**True**)  
 win.paint.setDisabled(**True**)  
 win.ect.setDisabled(**True**)  
  
  
**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**)  
 **else**:  
 win.input\_rect = **True** win.bars.setDisabled(**True**)  
 win.erase.setDisabled(**True**)  
 win.paint.setDisabled(**True**)  
 win.ect.setDisabled(**True**)  
  
*# Добавить строку с координатами с таблицу***def** add\_row(win):  
 win.table.insertRow(win.table.rowCount())  
  
*# Добавить точку***def** add\_point(point):  
 **global** w  
 **if** w.input\_bars:  
 **if** w.point\_now **is None**:  
 w.point\_now = point  
 **else**:  
 w.lines.append([[w.point\_now.x(), w.point\_now.y()],  
 [point.x(), point.y()]])  
  
 add\_row(w)  
 i = w.table.rowCount() - 1  
 item\_b = QTableWidgetItem(**"[{0}, {1}]"**.format(w.point\_now.x(), w.point\_now.y()))  
 item\_e = QTableWidgetItem(**"[{0}, {1}]"**.format(point.x(), point.y()))  
 w.table.setItem(i, 0, item\_b)  
 w.table.setItem(i, 1, item\_e)  
 w.scene.addLine(w.point\_now.x(), w.point\_now.y(), point.x(), point.y(), w.pen)  
 w.point\_now = **None***# Сброс всех ранее введённых данных***def** clean\_all(win):  
 win.scene.clear()  
 win.table.clear()  
 win.lines = []  
 win.image.fill(Qt.white)  
 r = win.table.rowCount()  
 **for** i **in** range(r, -1, -1):  
 win.table.removeRow(i)  
  
*# Добавление вертикальных и горизонтальных прямых на прямоугольнике***def** add\_bars(win):  
 **global** now  
 **if** now **is None**:  
 QMessageBox.warning(win, **"Внимание!"**, **"Не введен отсекатель!"**)  
 **return** buf = win.scene.itemAt(now, QTransform())  
 **if** buf **is None**:  
 QMessageBox.warning(win, **"Внимание!"**, **"Не введен отсекатель!"**)  
 **else**:  
 buf = buf.rect()  
 win.clip = [buf.left(), buf.right(), buf.top(), buf.bottom()]  
  
 t = abs(win.clip[2] - win.clip[3]) \* 0.8  
 k = abs(win.clip[0] - win.clip[1]) \* 0.8  
 *# задаем граничные отрезки* win.pen.setColor(red)  
 w.lines.append([[win.clip[0], win.clip[2] + t], [win.clip[0], win.clip[3] - t]])  
 add\_row(w)  
 i = w.table.rowCount() - 1  
 item\_b = QTableWidgetItem(**"[{0}, {1}]"**.format(win.clip[0], win.clip[2] + t))  
 item\_e = QTableWidgetItem(**"[{0}, {1}]"**.format(win.clip[0], win.clip[3] - t))  
 w.table.setItem(i, 0, item\_b)  
 w.table.setItem(i, 1, item\_e)  
 win.scene.addLine(win.clip[0], win.clip[2] + t, win.clip[0], win.clip[3] - t, win.pen)  
  
 w.lines.append([[win.clip[1], win.clip[2] + t], [win.clip[1], win.clip[3] - t]])  
 add\_row(w)  
 i = w.table.rowCount() - 1  
 item\_b = QTableWidgetItem(**"[{0}, {1}]"**.format(win.clip[1], win.clip[2] + t))  
 item\_e = QTableWidgetItem(**"[{0}, {1}]"**.format(win.clip[1], win.clip[3] - t))  
 w.table.setItem(i, 0, item\_b)  
 w.table.setItem(i, 1, item\_e)  
 win.scene.addLine(win.clip[1], win.clip[3] - t, win.clip[1], win.clip[2] + t, win.pen)  
  
 w.lines.append([[win.clip[0] + k, win.clip[2]], [win.clip[1] - k, win.clip[2]]])  
 add\_row(w)  
 i = w.table.rowCount() - 1  
 item\_b = QTableWidgetItem(**"[{0}, {1}]"**.format(win.clip[0] + k, win.clip[2]))  
 item\_e = QTableWidgetItem(**"[{0}, {1}]"**.format(win.clip[1] - k, win.clip[2]))  
 w.table.setItem(i, 0, item\_b)  
 w.table.setItem(i, 1, item\_e)  
 win.scene.addLine(win.clip[0] + k, win.clip[2], win.clip[1] - k, win.clip[2], win.pen)  
  
 w.lines.append([[win.clip[0] + k, win.clip[3]], [win.clip[1] - k, win.clip[3]]])  
 add\_row(w)  
 i = w.table.rowCount() - 1  
 item\_b = QTableWidgetItem(**"[{0}, {1}]"**.format(win.clip[0] + k, win.clip[3]))  
 item\_e = QTableWidgetItem(**"[{0}, {1}]"**.format(win.clip[1] - k, win.clip[3]))  
 w.table.setItem(i, 0, item\_b)  
 w.table.setItem(i, 1, item\_e)  
 win.scene.addLine(win.clip[0] + k, win.clip[3], win.clip[1] - k, win.clip[3], win.pen)  
  
  
**def** get\_code(a, rect):  
 code = [0, 0, 0, 0]  
 **if** a[0] < rect[0]:  
 code[0] = 1  
 **if** a[0] > rect[1]:  
 code[1] = 1  
 **if** a[1] < rect[2]:  
 code[2] = 1  
 **if** a[1] > rect[3]:  
 code[3] = 1  
  
 **return** code  
  
*# отсекание***def** clipping(win):  
 buf = win.scene.itemAt(now, QTransform()).rect()  
 win.clip = [buf.left(), buf.right(), buf.top(), buf.bottom()]  
 **for** b **in** win.lines:  
 **pass** win.pen.setColor(blue)  
 cohen\_sutherland(b, win.clip, win)  
 win.pen.setColor(red)  
  
  
**def** log\_prod(code1, code2):  
 p = 0  
 **for** i **in** range(4):  
 p += code1[i] & code2[i]  
  
 **return** p  
  
*# Видимость***def** is\_visible(bar, rect):  
 *"""Видимость - 0 = невидимый  
 1 = видимый  
 2 = частично видимый"""  
 # вычисление кодов концевых точек отрезка* s1 = sum(get\_code(bar[0], rect))  
 s2 = sum(get\_code(bar[1], rect))  
  
 *# предположим, что отрезок частично видим* vis = 2  
  
 *# проверка полной видимости отрезка* **if not** s1 **and not** s2:  
 vis = 1  
 **else**:  
 *# проверка тривиальной невидимости отрезка* l = log\_prod(get\_code(bar[0], rect), get\_code(bar[1], rect))  
 **if** l != 0:  
 vis = 0  
  
 **return** vis  
  
*# Алгоритм Сазерленда-Коэна***def** cohen\_sutherland(bar, rect, win):  
 *# инициализация флага* flag = 1 *# общего положения* t = 1  
  
 *# проверка вертикальности и горизонтальности отрезка* **if** bar[1][0] - bar[0][0] == 0:  
 flag = -1 *# вертикальный отрезок* **else**:  
 *# вычисление наклона* t = (bar[1][1] - bar[0][1]) / (bar[1][0] - bar[0][0])  
 **if** t == 0:  
 flag = 0 *# горизонтальный  
  
 # для каждой стороны окна* **for** i **in** range(4):  
 vis = is\_visible(bar, rect)  
 **if** vis == 1:  
 win.scene.addLine(bar[0][0], bar[0][1], bar[1][0], bar[1][1], win.pen)  
 **return  
 elif not** vis:  
 **return** *# проверка пересечения отрезка и стороны окна* code1 = get\_code(bar[0], rect)  
 code2 = get\_code(bar[1], rect)  
  
 **if** code1[i] == code2[i]:  
 **continue** *# проверка нахождения Р1 вне окна; если Р1 внутри окна, то Р2 и Р1 поменять местами* **if not** code1[i]:  
 bar[0], bar[1] = bar[1], bar[0]  
  
 *# поиск пересечений отрезка со сторонами окна  
 # контроль вертикальности отрезка* **if** flag != -1:  
 **if** i < 2:  
 bar[0][1] = t \* (rect[i] - bar[0][0]) + bar[0][1]  
 bar[0][0] = rect[i]  
 **continue  
 else**:  
 bar[0][0] = (1 / t) \* (rect[i] - bar[0][1]) + bar[0][0]  
  
 bar[0][1] = rect[i]  
 win.scene.addLine(bar[0][0], bar[0][1], bar[1][0], bar[1][1], win.pen)

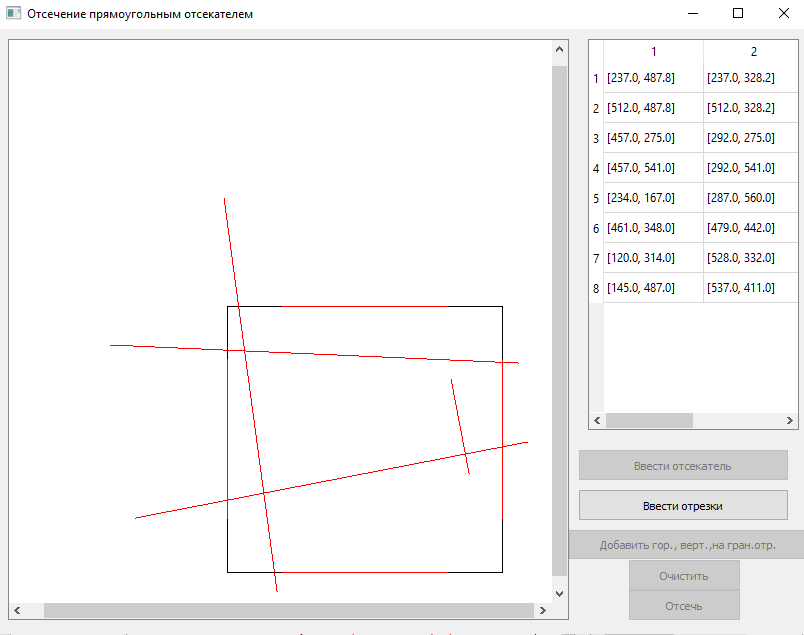
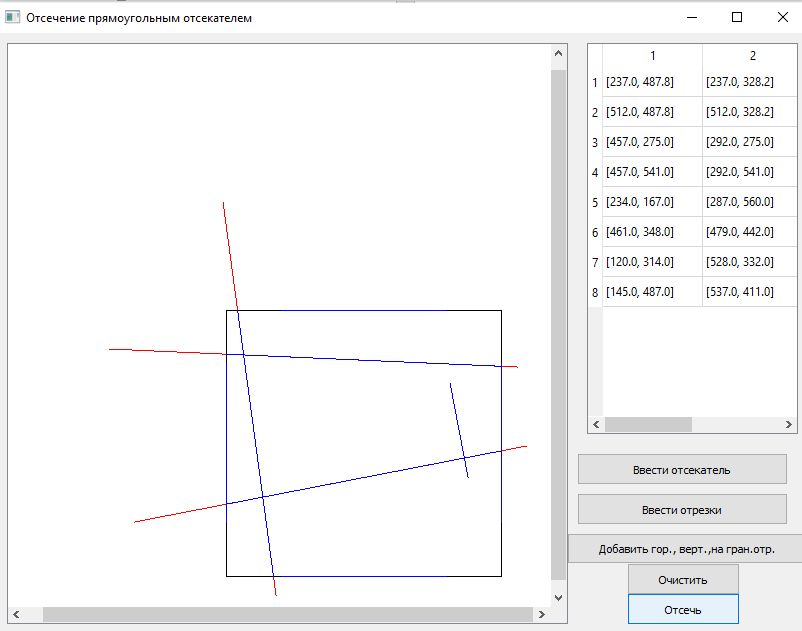
**Работа программы:**





Отсечённая часть выделена синим цветом.

Построим отрезки заново, добавим горизонтальные и вертикальные отрезки на отсекателе(обозначены оранжевым).

**Вывод:**

Благодаря этой лабораторной работе я изучил методы отчесения отрезков регулярным отсекателем, реализовав алгоритм Сазерленда-Коэна.