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

|  |  |
| --- | --- |
| **Тема: Реализация алгоритма отсечения отрезка**  **регулярным отсекателем.**  **Студент: Нгуен Ань Тхы**  **Группа: ИУ7-46Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель: Куров.А.В** |  |

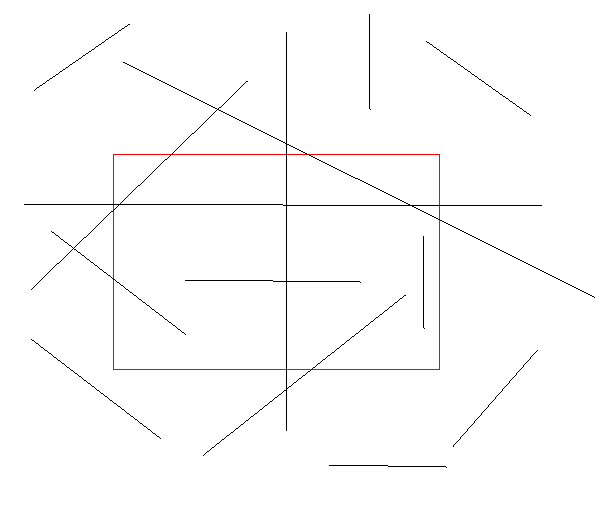
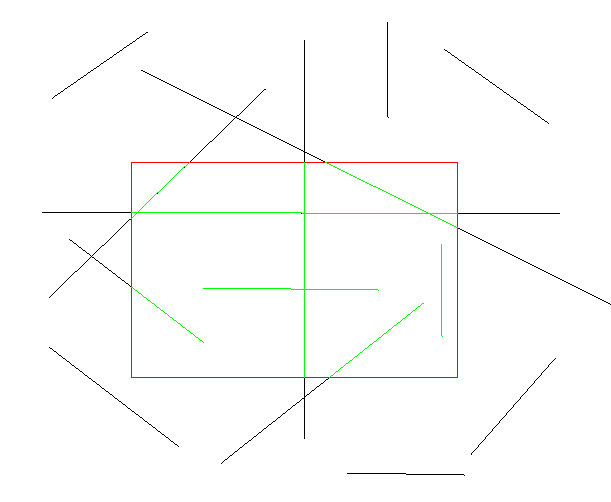
Москва.

2020 г.

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

Номер по списку 7: Простой алгоритм отсечения отрезка.

**I. Пример выполнения программы:**

****

**II. Алгоритм:**

1. Ввод координат отсекателя Xл, Xп, Yн, Yв.
2. Ввод координат концов отрезка P1(X1,Y1), P2(X 2,Y 2).
3. Вычисление кодов концов отрезка T1, T2.

Вычисление сумм кодов концов отрезка S1= *T*1*i* , S2=

1. Установка признака видимости отрезка pr =1

(pr =1 - отрезок видимый; pr = -1 - отрезок невидимый).

1. Задание начального значения тангенса угла наклона отрезка m = 1030 (большое число, вначале предполагается, что отрезок вертикальный).
2. Проверка полной видимости отрезка:

если (S1 = 0) & (S2 = 0) = истина, то отрезок видимый; выполнение в этом случае следующих действий: занесение в результат координат концов отрезка R1 = P1 ,R2 = P2 и переход к п. 31.

1. Вычисление логического произведения кодов концов отрезка

PL=

1. Проверка тривиальной невидимости отрезка, если PL <> 0, то отрезок невидим. В этом случае установка признака pr = -1 и переход к п. 31.
2. Проверка видимости первого конца отрезка:

если S1=0 (первый конец видим), то выполнение следующих действий: R1 = P1 (занесение этой вершины в результат), Q = P2 (занесение координат другой вершины в рабочую переменную Q), i = 2 (номер шага отсечения), переход к п.15.

1. Проверка видимости второго конца отрезка:

если S2 = 0 (второй конец видим), то выполнение следующих действий: R1 = P2 (занесение этой вершины в результат), Q = P1 (занесение координат дру- гой вершины в рабочую переменную Q), i=2 (номер шага отсечения), переход к п.15.

1. Установка начального значения шага отсечения i = 0.
2. Вычисление текущего номера шага отсечения i = i + 1.
3. Проверка завершения процедуры отсечения: если i > 2, то переход к п.31.
4. Занесение в рабочую переменную Q координат i-ой вершины Q = Pi.
5. Определение расположения отрезка: если X2 = X1 (отрезок вертикальный), то переход к п.23 (не может быть пересечения с левой и правой границами отсекателя).
6. Вычисление тангенса угла наклона отрезка m = (Y2 - Y1) / (X2 - X1).
7. Проверка возможности пересечения с левой границей отсекателя: если Qx > Xл (пересечения нет), то переход к п.20.
8. Вычисление ординаты точки пересечения отрезка с левой границей отсекателя: Yр = m(Xл - Qx) + Qy .
9. Проверка корректности найденного пересечения:

если (Yр >= Yн) & (Yp <= Yв) = истина (пересечение корректное), то выполнение следующих действий: Ri.x = Xл, Ri..y = Yр (занесение полученных координат в результат), переход к п.12.

1. Проверка возможности пересечения отрезка с правой границей отсекателя: если Qx < Xп (пересечения нет), то переход к п.23.
2. Вычисление ординаты точки пересечения с правой границей:

Yр= m(Xп - Qx) + Qy .

1. Проверка корректности найденного пересечения:

если (Yр >= Yн) & (Yр <= Yв) = истина (пересечение корректно), то выполнение следующих действий Ri.x =Xп, Ri..y = Yр ( занесение полученных координат в результат ), переход к п.12.

1. Проверка горизонтальности отрезка: если m = 0, то переход к п.12.
2. Проверка возможности пересечения с верхней границей отсекателя: если Qy < Yв (пересечения нет), то переход к п.27.
3. Вычисление абсциссы точки пересечения с верхней границей:

Xр= (Yв - Qy) / m + Qx.

1. Проверка корректности найденного пересечения:

если (Xр >= Xл) & (Xр <= Xп) = истина (пересечение корректно), то выполнение следующих действий: Ri.x = Xр; Ri.y = Yв (занесение полученных координат в результат); переход к п. 12.

1. Проверка возможности пересечения с нижней границей отсекателя: если Qx>Yн (пересечения нет), то переход к п. 30 (вершина невидима и ни одно пересечение не является корректным, следовательно, отрезок невидим).
2. Вычисление абсциссы точки пересечения с нижней границей:

Xр = (Yн - Qy) / m + Qx.

1. Проверка корректности найденного пересечения:

(Xр >= Xл) & (Xр <= Xп) = истина (пересечение корректно), то выполнение следующих действий: Ri.x = Xр; Ri.y = Yв (занесение полученных координат в результат); переход к п. 12.

1. Установка признака видимости pr = -1 (отрезок невидим полностью, так как ни одно пересечение не оказалось корректным).
2. Проверка признака видимости: если pr = 1, то вычерчивание отрезка R1R2.
3. Конец.

**III. Код программы:**

#Main

from PyQt5.QtWidgets import \*

from PyQt5.QtCore import \*

from PyQt5.QtGui import \*

from PyQt5 import uic

from PyQt5.QtWidgets import QMessageBox

import sys

from simple\_cut import \*

class Line:

def \_\_init\_\_(self):

self.x1 = 0

self.y1 = 0

self.x2 = 0

self.y2 = 0

self.scene\_item = None

class Cutter:

def \_\_init\_\_(self):

self.x\_left = 0

self.y\_up = 0

self.x\_right = 0

self.y\_down = 0

self.scene\_item = None

# Класс главного окна

class MyWindow(QMainWindow):

def \_\_init\_\_(self):

QWidget.\_\_init\_\_(self)

# Загрузка интерфейса

uic.loadUi("design.ui", self)

# Переменные

self.bg\_color = QColor(Qt.white)

self.line\_color = QColor(Qt.black)

self.cutter\_color = QColor(Qt.red)

self.cut\_line\_color = QColor(Qt.green)

self.ctrl\_pressed = False

self.lines = []

self.cur\_line = []

self.follow\_line = None

self.cutter = None

self.drawing\_cutter = False

self.cur\_cutter = []

self.follow\_cutter = None

# Добавляем полотно

self.scene = QGraphicsScene(0, 0, 1920, 1080)

self.mainview.setScene(self.scene)

self.pen = QPen()

self.mainview.ensureVisible(0, 0, 0, 0)

self.mainview.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff)

self.mainview.setVerticalScrollBarPolicy(Qt.ScrollBarAlwaysOff)

# Элементы ввода в интерфейсе

self.inputs = [

self.but\_add\_line,

self.but\_add\_cutter,

self.but\_choose\_cutter,

self.but\_cut,

self.but\_clear,

self.inp\_x1,

self.inp\_x2,

self.inp\_y1,

self.inp\_y2,

self.inp\_x\_left,

self.inp\_x\_right,

self.inp\_y\_up,

self.inp\_y\_down

]

# Настройка полей ввода

reg\_ex = QRegExp("[0-9]+")

int\_validator = QRegExpValidator(reg\_ex, self)

self.inp\_x1.setValidator(int\_validator)

self.inp\_x2.setValidator(int\_validator)

self.inp\_y1.setValidator(int\_validator)

self.inp\_y2.setValidator(int\_validator)

self.inp\_x\_left.setValidator(int\_validator)

self.inp\_x\_right.setValidator(int\_validator)

self.inp\_y\_up.setValidator(int\_validator)

self.inp\_y\_down.setValidator(int\_validator)

# Привязка кнопок

self.but\_add\_line.clicked.connect(lambda: get\_line(self))

self.but\_add\_cutter.clicked.connect(lambda: get\_cutter(self))

self.but\_choose\_cutter.clicked.connect(lambda: choose\_cutter(self))

self.but\_cut.clicked.connect(lambda: cut(self))

self.but\_clear.clicked.connect(lambda: clear(self))

# Остальные настройки

self.mainview.setMouseTracking(True)

self.mainview.viewport().installEventFilter(self)

# Отслеживание передвижения мыши

def eventFilter(self, source, event):

if event.type() == QEvent.MouseMove and source is self.mainview.viewport():

x = event.x()

y = event.y()

following\_line(self, x, y)

following\_cutter(self, x, y)

return QWidget.eventFilter(self, source, event)

# Нажатие клавиши

def keyPressEvent(self, event):

key = event.key()

if key == Qt.Key\_Control:

self.ctrl\_pressed = True

# Отжатие клавиши

def keyReleaseEvent(self, event):

key = event.key()

if key == Qt.Key\_Control:

self.ctrl\_pressed = False

# Нажатие кнопки мыши

def mousePressEvent(self, event):

but = event.button()

x = event.x()

y = event.y()

borders = self.mainview.geometry().getCoords()

if borders[0] <= x < borders[2] and borders[1] <= y < borders[3]:

x -= borders[0]

y -= borders[1]

else:

return

if but == 1:

line\_on\_screen(self, x, y)

cutter\_on\_screen(self, x, y)

def get\_line(self):

try:

x1 = int(self.inp\_x1.text())

y1 = int(self.inp\_y1.text())

x2 = int(self.inp\_x2.text())

y2 = int(self.inp\_y2.text())

except ValueError:

mes("Неверные данные отрезка")

return -1

add\_line(self, x1, y1, x2, y2, self.line\_color)

def get\_cutter(self):

try:

x\_left = int(self.inp\_x\_left.text())

y\_up = int(self.inp\_y\_up.text())

x\_right = int(self.inp\_x\_right.text())

y\_down = int(self.inp\_y\_down.text())

except ValueError:

mes("Неверные данные отрезка")

return -1

del\_cutter(self)

add\_cutter(self, x\_left, y\_up, x\_right, y\_down, self.cutter\_color)

def choose\_cutter(self):

del\_cutter(self)

self.drawing\_cutter = True

def cut(self):

if self.cutter:

for i in range(len(self.lines)):

xl = self.cutter.x\_left

xr = self.cutter.x\_right

yd = self.cutter.y\_down

yu = self.cutter.y\_up

p1 = [self.lines[i].x1, self.lines[i].y1]

p2 = [self.lines[i].x2, self.lines[i].y2]

yu, yd = yd, yu

visible, p1, p2 = easy\_cut(xl, xr, yd, yu, p1, p2)

if visible:

draw\_line(self, p1, p2, self.cut\_line\_color)

def clear(self):

self.scene.clear()

self.lines.clear()

self.cur\_line.clear()

self.follow\_line = None

self.cutter = None

self.cur\_cutter.clear()

self.follow\_cutter = None

self.drawing\_cutter = False

# Выводит окно с предупреждением

def mes(text):

msg = QMessageBox()

msg.setIcon(QMessageBox.Warning)

msg.setWindowTitle("Внимание")

msg.setText(text)

msg.setStandardButtons(QMessageBox.Ok)

retval = msg.exec\_()

def add\_line(self, x1, y1, x2, y2, color):

self.pen.setColor(color)

line = Line()

line.x1 = x1

line.y1 = y1

line.x2 = x2

line.y2 = y2

line.scene\_item = self.scene.addLine(x1, y1, x2, y2, self.pen)

self.lines.append(line)

def add\_cutter(self, x\_l, y\_u, x\_r, y\_d, color):

self.pen.setColor(color)

if x\_l > x\_r:

x\_l, x\_r = x\_r, x\_l

if y\_u > y\_d:

y\_u, y\_d = y\_d, y\_u

cutter = Cutter()

cutter.x\_left = x\_l

cutter.y\_up = y\_u

cutter.x\_right = x\_r

cutter.y\_down = y\_d

cutter.scene\_item = self.scene.addRect(x\_l, y\_u, x\_r - x\_l, y\_d - y\_u, self.pen)

self.cutter = cutter

def del\_cutter(self):

if self.cutter:

self.scene.removeItem(self.cutter.scene\_item)

self.cutter = None

def line\_on\_screen(self, x, y):

if not self.drawing\_cutter:

if self.ctrl\_pressed == 0 or len(self.cur\_line) == 0:

self.cur\_line.append((x, y))

else:

prev = self.cur\_line[0]

dx = x - prev[0]

dy = y - prev[1]

if abs(dy) >= abs(dx):

self.cur\_line.append((prev[0], y))

else:

self.cur\_line.append((x, prev[1]))

if len(self.cur\_line) == 2:

c1, c2 = self.cur\_line

add\_line(self, c1[0], c1[1], c2[0], c2[1], self.line\_color)

self.cur\_line.clear()

self.scene.removeItem(self.follow\_line)

def cutter\_on\_screen(self, x, y):

if self.drawing\_cutter:

if len(self.cur\_cutter) < 2:

self.cur\_cutter.append((x, y))

if len(self.cur\_cutter) == 2:

c1, c2 = self.cur\_cutter

add\_cutter(self, c1[0], c1[1], c2[0], c2[1], self.cutter\_color)

self.cur\_cutter.clear()

self.scene.removeItem(self.follow\_cutter)

self.drawing\_cutter = False

def following\_line(self, x, y):

if len(self.cur\_line) == 1:

prev = self.cur\_line[0]

self.pen.setColor(self.line\_color)

if self.follow\_line:

self.scene.removeItem(self.follow\_line)

if self.ctrl\_pressed:

dx = x - prev[0]

dy = y - prev[1]

if abs(dy) >= abs(dx):

cur = (prev[0], y)

else:

cur = (x, prev[1])

self.follow\_line = self.scene.addLine(prev[0], prev[1], cur[0], cur[1], self.pen)

else:

self.follow\_line = self.scene.addLine(prev[0], prev[1], x, y, self.pen)

def following\_cutter(self, x, y):

if len(self.cur\_cutter) == 1:

x\_l, y\_u = self.cur\_cutter[0]

x\_r, y\_d = x, y

self.pen.setColor(self.cutter\_color)

if self.follow\_cutter:

self.scene.removeItem(self.follow\_cutter)

if x\_l > x\_r:

x\_l, x\_r = x\_r, x\_l

if y\_u > y\_d:

y\_u, y\_d = y\_d, y\_u

self.follow\_cutter = self.scene.addRect(x\_l, y\_u, x\_r - x\_l, y\_d - y\_u, self.pen)

def draw\_line(self, dot1, dot2, color):

self.pen.setColor(color)

self.scene.addLine(dot1[0], dot1[1], dot2[0], dot2[1], self.pen)

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication([])

application = MyWindow()

application.show()

sys.exit(app.exec())

#Algorithms

import copy

INFINITY = 1000000000000000000000000000000

#Вычисление сумм кодов концов отрезка

def count\_S(T):

return sum(T)

#Вычисление логического произведения кодов концов отрезка

def count\_P(T1, T2):

P = 0

for i in range(len(T1)):

P += (T1[i] \* T2[i])

return P

#Вычисление кодов концов отрезка

def count\_T(p, xl, xr, yd, yu):

x, y = p[0], p[1]

# y = p[1]

T = [0, 0, 0, 0]

T[0] = 1 if x < xl else 0

T[1] = 1 if x > xr else 0

T[2] = 1 if y < yd else 0

T[3] = 1 if y > yu else 0

return T

def easy\_cut(xl, xr, yd, yu, p1, p2):

T1 = count\_T(p1, xl, xr, yd, yu)

T2 = count\_T(p2, xl, xr, yd, yu)

S1 = count\_S(T1)

S2 = count\_S(T2)

PR = 1 #признак видимости

m = INFINITY # тангенс

Q = p1

r1 = copy.deepcopy(p1)

r2 = copy.deepcopy(p2)

if (S1 == 0) and (S2 == 0):

return show\_pix(PR, r1, r2)

P = count\_P(T1, T2)

if P != 0:

PR = -1

return show\_pix(PR,r1, r2)

if S1 == 0:

r1 = copy.deepcopy(p1)

Q = copy.deepcopy(p2)

i = 2

return CUT(PR, i, Q, p1, p2, r1, r2, xl, xr, yd, yu, False)

if S2 == 0:

r1 = copy.deepcopy(p2)

Q = copy.deepcopy(p1)

i = 2

return CUT(PR, i, Q, p1, p2, r1, r2, xl, xr, yd, yu, False)

i = 0

return CUT(PR, i, Q, p1, p2, r1, r2, xl, xr, yd, yu)

def show\_pix(PR, p1, p2):

if PR == 1:

return True, p1, p2

else:

return False, p1, p2

def CUT(PR, i, Q, p1, p2, r1, r2, xl, xr, yd, yu, flag=True):

# Определение расположения отрезка

if flag:

i += 1

if i > 2:

return show\_pix(PR, r1, r2)

Q = p1 if i == 1 else p2 # Q = Pi

if p1[0] == p2[0]:

return CUT\_skip(PR, i, Q, p1, p2, r1, r2, xl, xr, yd, yu, INFINITY)

#Вычисление тангенса угла наклона отрезка

m = (p2[1] - p1[1]) / (p2[0] - p1[0])

if Q[0] < xl:

# Вычисление ординаты точки пересечения с левой границей отсекателя

yp = m \* (xl - Q[0]) + Q[1]

# проверка корректности найденного пересечения

if yp >= yd and yp <= yu:

if i == 1:

r1[0] = xl

r1[1] = yp

else:

r2[0] = xl

r2[1] = yp

return CUT(PR, i, Q, p1, p2, r1, r2, xl, xr, yd, yu)

if Q[0] > xr:

# Вычисление ординаты точки пересечения с правой границей

yp = m \* (xr - Q[0]) + Q[1]

# проверка корректности найденного пересечения

if yp >= yd and yp <= yu:

if i == 1:

r1[0] = xr

r1[1] = yp

else:

r2[0] = xr

r2[1] = yp

return CUT(PR, i, Q, p1, p2, r1, r2, xl, xr, yd, yu)

return CUT\_skip(PR, i, Q, p1, p2, r1, r2, xl, xr, yd, yu, m)

def CUT\_skip(PR, i, Q, p1, p2, r1, r2, xl, xr, yd, yu, m):

# проверка горизонтальности

if m == 0:

return CUT(PR, i, Q, p1, p2, r1, r2, xl, xr, yd, yu, m)

if Q[1] > yu:

# Вычисление абсциссы точки пересечения с верхней границей

xp = (yu - Q[1]) / m + Q[0]

# Проверка корректности найденного пересечения

if xp >= xl and xp <= xr:

if i == 1:

r1[0] = xp

r1[1] = yu

else:

r2[0] = xp

r2[1] = yu

return CUT(PR, i, Q, p1, p2, r1, r2, xl, xr, yd, yu)

if Q[1] < yd:

# Вычисление абсциссы точки пересечения с нижней границей

xp = (yd - Q[1]) / m + Q[0]

# Проверка корректности найденного пересечения

if xp >= xl and xp <= xr:

if i == 1:

r1[0] = xp

r1[1] = yd

else:

r2[0] = xp

r2[1] = yd

return CUT(PR, i, Q, p1, p2, r1, r2, xl, xr, yd, yu)

PR = -1

return show\_pix(PR, r1, r2)