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

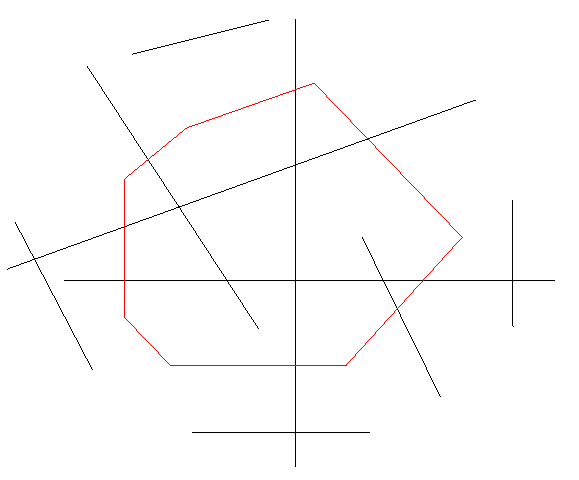
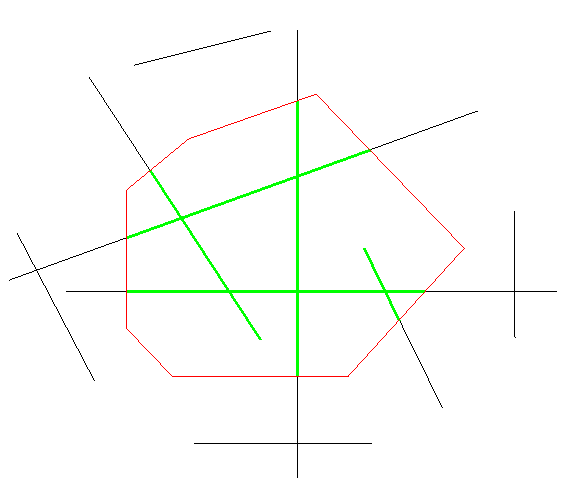
|  |  |
| --- | --- |
| **Тема: Реализация алгоритма отсечения отрезка произвольным выпуклым отсекателем. (Алгоритм Кируса-Бека)**  **Студент: Нгуен Ань Тхы**  **Группа: ИУ7-46Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель: Куров.А.В** |  |

Москва.

2020 г.

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

**I. Пример выполнения программы:**

**II. Алгоритм:**

1. Ввод P1, P2, K, C(K)
2. tH = 0, tB = 1
3. D = P2 – P1
4. Проверка отсекателя на выпукность
5. Цикл отсекателя (по i от 1 до K)
   1. Wi = P1 – fi
   2. Вычисление Dск, Wск
   3. Если Dск = 0, то проверить если Wск < 0, то отрезок невидим
   4. t = Wск / Dск
   5. Если Dск > 0, то проверить если t > 1, то отрезок невидимо, иначе tH = max(t, tH)
   6. Если Dск <= 0, то проверить если t < 0, то отрезок невидимо, иначе tB = min(t, tB)
   7. Конец цикла
   8. Если tH <= tB, то изобразить отрезок (tH, tB)
6. Конец.

**III. Код программы:**

from PyQt5.QtWidgets import \*

from PyQt5.QtCore import \*

from PyQt5.QtGui import \*

from PyQt5 import uic

import sys

from algorithms 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.coords = []

self.scene\_items = []

# 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.highlighted\_lines = []

self.ctrl\_pressed = False

self.lines = []

self.cur\_line = []

self.follow\_line = None

self.cutter = None

self.drawing\_cutter = False

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\_end\_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\_cutter,

self.inp\_y\_cutter,

]

# 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\_cutter.setValidator(int\_validator)

self.inp\_x\_cutter.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\_end\_cutter.clicked.connect(lambda: end\_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)

remove\_highlight(self)

if but == 2:

end\_cutter\_on\_screen(self)

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 = int(self.inp\_x\_cutter.text())

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

except ValueError:

mes("Неверные данные отсекателя")

return -1

if self.drawing\_cutter == False:

del\_cutter(self)

remove\_highlight(self)

if not self.cutter:

print("created cutter")

self.cutter = Cutter()

self.drawing\_cutter = True

self.cutter.coords.append((x, y))

if len(self.cutter.coords) > 1:

print("here")

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

c = self.cutter.coords

self.cutter.scene\_items.append(self.scene.addLine(c[-1][0], c[-1][1], c[-2][0], c[-2][1], self.pen))

def end\_cutter(self):

if self.drawing\_cutter:

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

if len(self.cutter.coords) > 2:

c = self.cutter.coords

self.cutter.scene\_items.append(self.scene.addLine(c[-1][0], c[-1][1], c[0][0], c[0][1], self.pen))

self.drawing\_cutter = False

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

def choose\_cutter(self):

del\_cutter(self)

self.cutter = Cutter()

self.drawing\_cutter = True

remove\_highlight(self)

def cut(self):

if self.cutter and len(self.cutter.coords) > 2:

convex, clockwise = check\_convex\_polygon(self.cutter.coords)

if convex == False:

mes("Отсекатель невыпуклый")

del\_cutter(self)

return -1

remove\_highlight(self)

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

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

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

visible, r1, r2 = cyrus\_beck(self.cutter.coords, p1, p2, clockwise)

if visible:

highlight(self, r1, r2)

redraw\_cutter(self)

def clear(self):

self.scene.clear()

self.lines.clear()

self.cur\_line.clear()

self.follow\_line = None

self.cutter = None

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\_()

# Рисует отрезки

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 redraw\_cutter(self):

if self.cutter:

n = len(self.cutter.coords)

for i in self.cutter.scene\_items:

self.scene.removeItem(i)

self.cutter.scene\_items.clear()

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

for i in range(n):

p1 = self.cutter.coords[i - 1]

p2 = self.cutter.coords[i]

self.cutter.scene\_items.append(self.scene.addLine(p1[0], p1[1], p2[0], p2[1], self.pen))

# OK

def del\_cutter(self):

if self.cutter:

for c in self.cutter.scene\_items:

self.scene.removeItem(c)

self.cutter = None

# OK Рисует отрезки

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)

redraw\_cutter(self)

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

if self.drawing\_cutter:

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

c = self.cutter.coords

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

c.append((x, y))

else:

prev = c[-1]

dx = x - prev[0]

dy = y - prev[1]

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

c.append((prev[0], y))

else:

c.append((x, prev[1]))

if len(c) > 1:

self.cutter.scene\_items.append(self.scene.addLine(c[-1][0], c[-1][1], c[-2][0], c[-2][1], self.pen))

def end\_cutter\_on\_screen(self):

if self.drawing\_cutter:

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

if len(self.cutter.coords) > 2:

c = self.cutter.coords

self.cutter.scene\_items.append(self.scene.addLine(c[-1][0], c[-1][1], c[0][0], c[0][1], self.pen))

self.drawing\_cutter = False

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

# OK

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 self.drawing\_cutter and len(self.cutter.coords) > 0:

prev = self.cutter.coords[-1]

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

if self.follow\_cutter:

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

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

else:

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

# OK

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

self.pen.setColor(color)

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

def highlight(self, p1, p2):

self.pen.setColor(self.cut\_line\_color)

dx = abs(p1[0] - p2[0])

dy = abs(p1[1] - p2[1])

for i in range(3):

if dx > dy:

line = self.scene.addLine(p1[0], p1[1] - 1 + i, p2[0], p2[1] - 1 + i, self.pen)

else:

line = self.scene.addLine(p1[0] - 1 + i, p1[1], p2[0] - 1 + i, p2[1], self.pen)

self.highlighted\_lines.append(line)

def remove\_highlight(self):

for i in self.highlighted\_lines:

self.scene.removeItem(i)

self.highlighted\_lines.clear()

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

app = QApplication([])

application = MyWindow()

application.show()

sys.exit(app.exec())

#algorithms

#Проверка на выпукность

def check\_convex\_polygon(c):

neg = 0

pos = 0

zer = 0

for i in range(len(c)):

c1 = c[i]

c2 = c[(i + 1) % len(c)]

c3 = c[(i + 2) % len(c)]

vec1 = (c2[0] - c1[0], c2[1] - c1[1])

vec2 = (c3[0] - c2[0], c3[1] - c2[1])

res = vec1[0] \* vec2[1] - vec1[1] \* vec2[0]

if res > 0:

pos += 1

elif res < 0:

neg += 1

else:

zer += 1

if len(c) == zer:

return False, None

elif len(c) == pos + zer:

return True, True

elif len(c) == neg + zer:

return True, False

else:

return False, None

# Вектор Нормали

def get\_normal\_vector(vec, clockwise):

if clockwise:

return (-vec[1], vec[0]) # по часовой стрелке

else:

return (vec[1], -vec[0]) # против часовой стрелки

# Вектор

def get\_vec(p1, p2):

return (p2[0] - p1[0], p2[1] - p1[1])

# Скалярное произведение

def mul\_scal(a, b):

return a[0] \* b[0] + a[1] \* b[1]

# P(t) = P1 + (P2 - P1)t

def P(t, p1, p2):

return (p1[0] + round((p2[0] - p1[0]) \* t), p1[1] + round((p2[1] - p1[1]) \* t))

# Алгоритм Кируса Бека

def cyrus\_beck(cutter, p1, p2, clockwise):

K = len(cutter)

visible = False

t\_bot, t\_top = 0, 1

p\_top, p\_top = p1, p2

D = get\_vec(p1, p2)

for i in range(K):

# Вектор ребра

edge\_vec = get\_vec(cutter[i], cutter[(i + 1) % K])

# Вектор Нормали

n\_vec = get\_normal\_vector(edge\_vec, clockwise)

W = get\_vec(cutter[i], p1)

Dsc = mul\_scal(D, n\_vec)

Wsc = mul\_scal(W, n\_vec)

if Dsc == 0:

if Wsc < 0:

return visible, p1, p2

else:

t = -Wsc / Dsc

if Dsc > 0:

if t > 1:

return visible, p1, p2

else:

t\_bot = max(t\_bot, t)

else:

if t < 0:

return visible, p1, p2

else:

t\_top = min(t\_top, t)

if t\_bot <= t\_top:

p\_bot = P(t\_bot, p1, p2)

p\_top = P(t\_top, p1, p2)

visible = True

return visible, p\_bot, p\_top