МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра САПР

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Компьютерная графика»

Тема: Исследование алгоритмов отсечения отрезков и многоугольников

окнами различного вида

Студенты гр. 5371 Бергер Э. Э.

Локкина О. С.

Уруков С. Д.

Преподаватель Матвеева И. В.

Санкт-Петербург

2017

**ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ**

*Вариант 4.3:*

Обеспечить реализацию алгоритма отсечения массива произвольных отрезков заданным прямоугольным окном с использование алгоритма Коэна - Сазерленда. Вначале следует вывести на экран сгенерированные отрезки полностью, а затем другим цветом или яркостью те, которые полностью или частично попадают в область окна.

**ХОД РАБОТЫ**

Использовалась библиотека OpenGL с языком программирования Python 3.6.x.

Задано прямоугольное окно со следующими границами:

xmin = -5

xmax = 5

ymin = -5

ymax = 5

Программа строит 7 отрезков, вершины которых задаются случайным образом в промежутке от -8 до 8. Каждой вершине соответствует код, который состоит из 4 бит и определяет положение вершины относительно окна. Затем, с помощью алгоритма Коэна-Сазерленда, определяется, какие отрезки лежат в окне, а какие – нет.

Блок-схема алгоритма Коэна-Сазерленда:

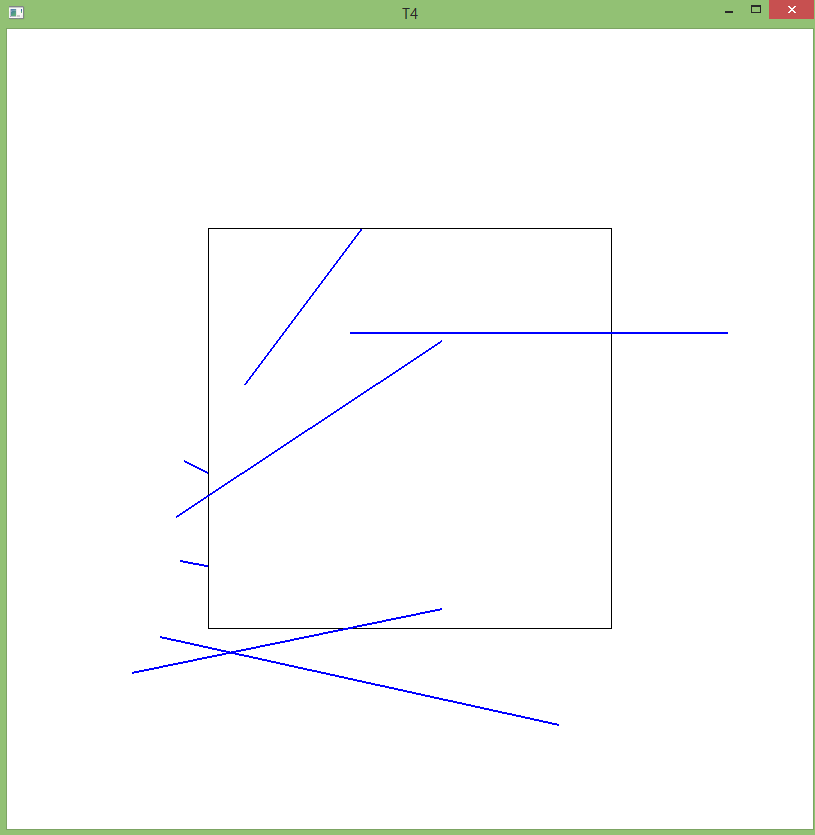


**ПРИМЕР ЗАПУСКА ПРОГРАММЫ**

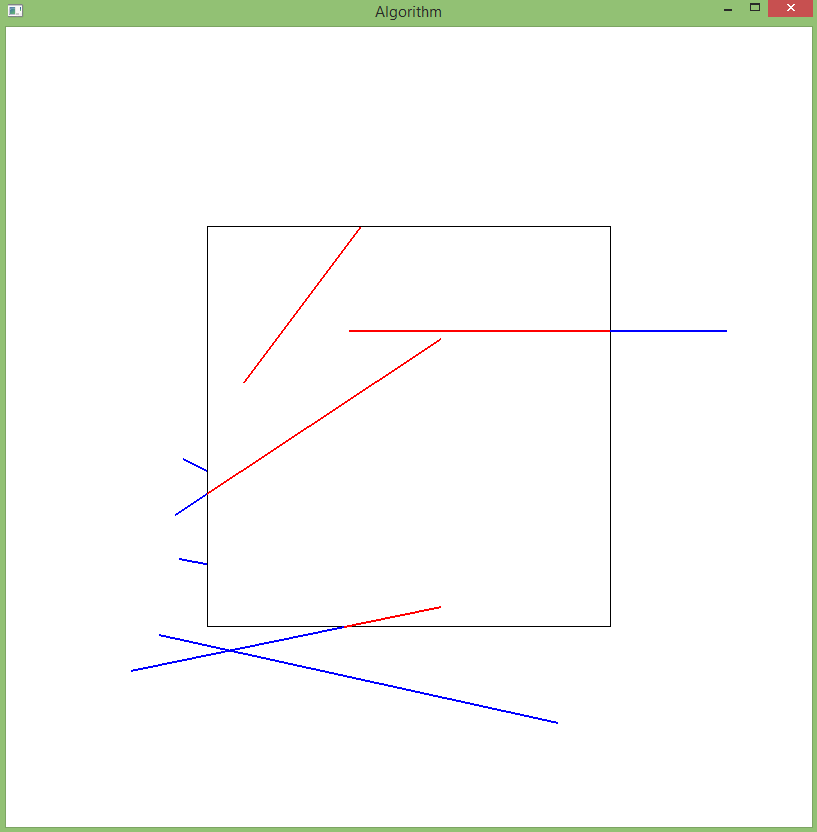
При запуске программы выводится массив со сгенерированными случайным образом вершинами:



Также открываются два окна. В первом окне изображен прямоугольник и получившиеся отрезки:



Во втором окне отрезки или части отрезков, попавшие в прямоугольную область, окрашиваются красным цветом, а не попавшие – остаются синими:



**ВЫВОД**

В ходе выполнения лабораторной работы были получены навыки отсечения отрезков прямоугольным окном с помощью алгоритма Коэна-Сазерленда.

**КОД ПРОГРАММЫ**

from OpenGL.GL import \*

from OpenGL.GLUT import \*

from random import choice

window = 0

x, y = 0, 0

xmin = -5 # Границы прямоугольника

xmax = 5

ymin = -5

ymax = 5

LEFT = 1 # двоичное 0001

RIGHT = 2 # двоичное 0010

BOT = 4 # двоичное 0100

TOP = 8 # двоичное 1000

points = []

def createPoints(count, start, stop, step=0.1):

global points

listOfNumbers = []

result = start

while result < stop:

listOfNumbers.append(round(result, 1))

result += step

for i in range(count):

points.append([choice(listOfNumbers), choice(listOfNumbers)])

def drawSegm(vertices):

glLineWidth(2)

glBegin(GL\_LINES)

glColor3f(0, 0, 1)

for element in vertices:

glVertex2dv(element)

glEnd()

def initBox():

global ymin, ymax, xmax, xmin

glLineWidth(1)

glBegin(GL\_LINES)

glColor3f(0, 0, 0)

glVertex2f(xmin, ymin)

glVertex2f(xmin, ymax)

glVertex2f(xmin, ymax)

glVertex2f(xmax, ymax)

glVertex2f(xmax, ymax)

glVertex2f(xmax, ymin)

glVertex2f(xmax, ymin)

glVertex2f(xmin, ymin)

glEnd()

def checkZero(a, b):

if a - b:

return a - b

else:

return a

def bitCode(point):

x = point[0]

y = point[1]

code = 0

if x < xmin:

code |= LEFT

elif x > xmax:

code |= RIGHT

if y < ymin:

code |= BOT

elif y > ymax:

code |= TOP

return code

def CohenSutherlandAlgorithm(pointOne, pointTwo):

global ymin, ymax, xmax, xmin, x, y

kb = bitCode(pointOne) # k begin

ke = bitCode(pointTwo) # k end

if (kb | ke) == 0: # Лежит полностью

glColor3f(1, 0, 0) # Красный

glVertex2fv(pointOne)

glVertex2fv(pointTwo)

elif kb & ke: # За пределами

glColor3f(0, 0, 1) # Синий

glVertex2fv(pointOne)

glVertex2fv(pointTwo)

else:

while kb | ke:

if kb == 0:

kb, ke = ke, kb

pointOne[0], pointTwo[0] = pointTwo[0], pointOne[0]

pointOne[1], pointTwo[1] = pointTwo[1], pointOne[1]

if kb & LEFT:

x = xmin

y = pointOne[1] + (pointTwo[1] - pointOne[1]) \* (x - pointOne[0]) / checkZero(pointTwo[0], pointOne[0])

elif kb & RIGHT:

x = xmax

y = pointOne[1] + (pointTwo[1] - pointOne[1]) \* (x - pointOne[0]) / checkZero(pointTwo[0], pointOne[0])

elif kb & TOP:

y = ymax

x = pointOne[0] + (pointTwo[0] - pointOne[0]) \* (y - pointOne[1]) / checkZero(pointTwo[1], pointOne[1])

else:

y = ymin

x = pointOne[0] + (pointTwo[0] - pointOne[0]) \* (y - pointOne[1]) / checkZero(pointTwo[1], pointOne[1])

glColor3f(0, 0, 1) # Синий

glVertex2fv(pointOne)

glVertex2f(x, y)

pointOne = [x, y]

kb = bitCode(pointOne)

glColor3f(1, 0, 0) # Красный

glVertex2fv(pointOne)

glVertex2fv(pointTwo)

def draw2DScene():

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

glLoadIdentity()

glClearColor(1.0, 1.0, 1.0, 1.0)

glScalef(0.1, 0.1, 10)

initBox()

drawSegm(points)

glutSwapBuffers()

def draw2DSceneAlgorithm():

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

glLoadIdentity()

glClearColor(1.0, 1.0, 1.0, 1.0)

glScalef(0.1, 0.1, 10)

initBox()

initAlgorithm()

glutSwapBuffers()

def initAlgorithm():

global points

glLineWidth(2)

glBegin(GL\_LINES)

for i in range(0, len(points)-1, 2):

CohenSutherlandAlgorithm(points[i], points[i+1])

glEnd()

def initWindow():

global window

glutInit()

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE | GLUT\_ALPHA | GLUT\_DEPTH)

glutInitWindowSize(800, 800)

glutInitWindowPosition(500, 100)

window = glutCreateWindow(b"T4")

glutDisplayFunc(draw2DScene)

def initSubWindow():

glutInit()

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE | GLUT\_ALPHA | GLUT\_DEPTH)

glutInitWindowSize(800, 800)

glutInitWindowPosition(500, 100)

SubWindow = glutCreateWindow(b"Algorithm")

glutSetWindow(SubWindow)

glutDisplayFunc(draw2DSceneAlgorithm)

def main():

global points

createPoints(14, -8, 8)

initWindow()

print(points)

initSubWindow()

glutMainLoop()

main()