**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**По лабораторной работе № 3**

**по дисциплине «Компьютерная графика»**

Тема: **Построение фракталов**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0303 |  | Архипов В.А. |
| Преподаватель |  | Герасимова Т.В. |

Санкт-Петербург

2023

**Цель работы.**

Реализовать программу, отображающую фрактал, соответствующий индивидуальному варианту.

**Задание.**

На базе предыдущей лабораторной работы разработать программу, реализующую фрактал по индивидуальному заданию.

Фрактал для реализации представлен на рис. 1 (вариант 37).

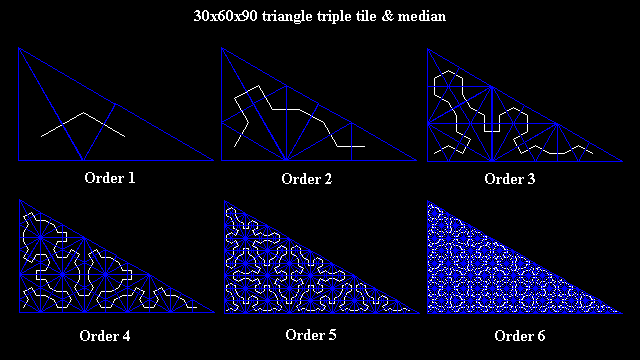


Рисунок 1 – конкретный фрактал для реализации

**Выполнение работы.**

Работа была выполнена в среде разработки PyCharm на языке программирования Python 3.10. Для реализации пользовательского интерфейса были использованы библиотека PyQt6 и программа Qt Designer, создающая пользовательский интерфейс по построенному в ней макету. Для работы с графикой была использована библиотека PyOpenGL. Подключение графической библиотеки к пользовательскому интерфейсу было осуществлено с помощью виджета QOpenGLWidget из библиотеки PyQt6.

Макет пользовательского интерфейса был разработан в программе Qt Designer (см. рис. 2).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – разработанный в Qt Designer макет пользовательского интерфейса

Класс главного окна MainWindow наследуется от базового для PyQt класса QMainWindow и содержит внутри себя виджет glWidget унаследованный от QOpenGLWidget, для отображения графики, а также меню для настройки параметров изображения (в конкретном случае параметр всего один – это число итераций).

При внимательном рассмотрении фрактала была выведана закономерность его построения.

Сперва берется прямоугольный треугольник с углами 30 и 60 градусов. Из угла 60 градусов проводится отрезок, который делит противолежащий катет в отношении 1:2. Из получившейся точки пересечения на гипотенузу опускается медиана. Таким образом мы получаем три треугольника. Далее в каждом треугольнике по очереди нужно отметить центр вписанной окружности (точку пересечения биссектрис треугольника). Прямой порядок обхода треугольников показан на рис. 3.

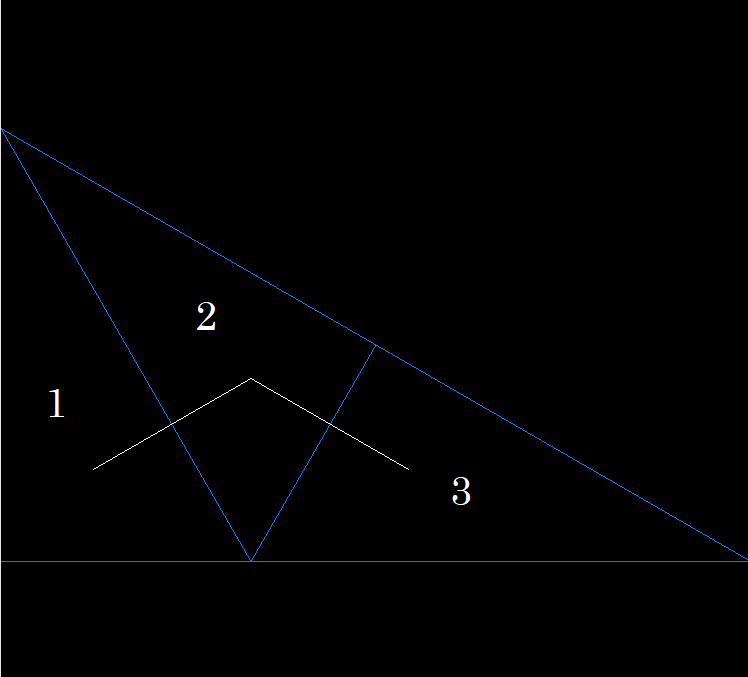


Рисунок 3 – прямой порядок обхода треугольников

Обратный порядок обхода треугольников при построении фрактала изображен на рис. 4.

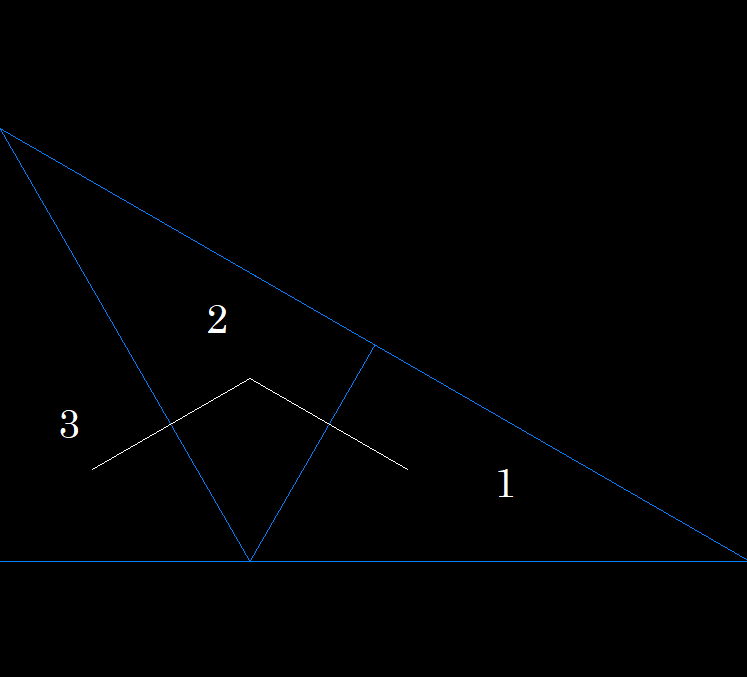


Рисунок 4 – обратный порядок обхода

Если число итераций больше одного, тогда для каждого из трех получившихся треугольников запускается алгоритм построения внутри одного треугольника трёх и так до тех пор, пока не будет достигнуто указанное число итераций. Когда оно будет достигнуто, внутри каждого треугольника будет отмечена точка пересечения его биссектрис, а после эти точки будут соединены в ломаную линию. Таким образом мы получим интересующий нас фрактал.

Теперь перейдем к рассмотрению реализации программы по построению фрактала. Для облегчения вычислений был реализован модуль geometry.py, который содержит следующие методы:

find\_len – вычисляет длину отрезка по точкам его начала и конца

median\_dots – вычисляет координаты середины заданного отрезка

bisectors\_intersection\_dot – вычисляет координаты точки пересечения биссектрис треугольника по его вершинам

divide\_cutoff – вычисляет координаты точки, которая делит заданный отрезок в определённом соотношении (по умолчанию это соотношение равно 1/2)

Рисование фрактала начинается с запуска функции draw\_fractal, которая принимает точки треугольника, внутри которого фрактал будет отрисован, а также число итераций (его она считывает из слайдера, которым управляет пользователь).

Для работы функции draw\_fractal необходимы три массива – массив текущих треугольников old\_triangles, массив, куда будут складываться треугольники, построенные внутри текущих, new\_triangles, и массив, куда будут складываться центры окружностей, вписанных в треугольники из массива old\_triangles.

В начале своей работы функция draw\_fractal помещает в old\_triangles переданный функции треугольник. Далее запускается цикл. Число итераций указывает пользователь, перемещая слайдер. Внутри цикла из массива old\_triangles по одному извлекаются треугольники. Для каждого рассматриваемого треугольника запускается функция count\_triangles, которая находит координаты трех треугольников (в виде списка), которые будут построены внутри переданного ей треугольника. Далее найденный список треугольников добавляется в список new\_triangles. Если индекс рассматриваемого в цикле треугольника нечетный, то в дальнейшем к найденным внутри него треугольникам стоит применить обратный порядок обхода, поэтому список треугольников, который мы нашли в таком треугольнике, перед суммированием с остальными разворачивается. Когда все треугольники на текущей итерации рассмотрены, массив new\_triangles будет заполнен треугольниками меньшего размера. Обозначим этот массив как old\_triangles, а new\_triangles затрем и продолжим основной цикл программы. Как только необходимое число итераций будет совершено, для каждого треугольника из old\_triangles будет найден центр вписанной в него окружности. Координаты этих центров будут записаны в список points.

Далее происходит отрисовка всех треугольников из old\_triangles и всех точек из points.

Ниже приведена часть кода, которая отвечает за итеративное построение фрактала.

def draw\_fractal(self, dots, iterator):  
 self.old\_triangles.append(dots)  
 for \_ in range(iterator):  
 for index,triangle in enumerate(self.old\_triangles):  
 res = self.count\_triangles(triangle)  
 if index % 2 != 0:  
 res.reverse()  
 self.new\_triangles += res  
 self.old\_triangles = self.new\_triangles  
 self.new\_triangles = []  
 for triangle in self.old\_triangles:  
 self.points.append(bisectors\_intersection\_dot(triangle))

Функция, которая находит координаты трёх треугольников внутри одного.

def count\_triangles(self, dots):  
 return [  
 [divide\_cutoff(dots[1], dots[2]), dots[1], dots[0]],  
 [divide\_cutoff(dots[1], dots[2]), median\_dots([dots[0], dots[2]]), dots[0]],  
 [divide\_cutoff(dots[1], dots[2]), median\_dots([dots[0], dots[2]]), dots[2]]  
 ]

Треугольники здесь расположены в прямом порядке обхода.

Часть программы, отвечающая за отрисовку фрактала.

*# Рисование треугольников*gl.glBegin(gl.GL\_LINE\_STRIP)  
gl.glColor3f(0.00, 0.5, 1)  
for triangle in self.old\_triangles:  
 for point in triangle:  
 gl.glVertex2fv(point)  
for point in self.start:  
 gl.glVertex2fv(point)  
gl.glEnd()  
  
*# Рисование точек*gl.glColor3f(1, 1, 1)  
gl.glBegin(gl.GL\_LINE\_STRIP)  
for point in self.points:  
 gl.glVertex2fv(point)  
gl.glEnd()

Разработанный программный код см. в приложении А.

**Тестирование.**

Тестирование проводилось методом сравнения построенного программой фрактала с референсом, предоставленным преподавателем.

Первые шесть итераций представлены на снимках экрана (см. рис. 5 – 10).

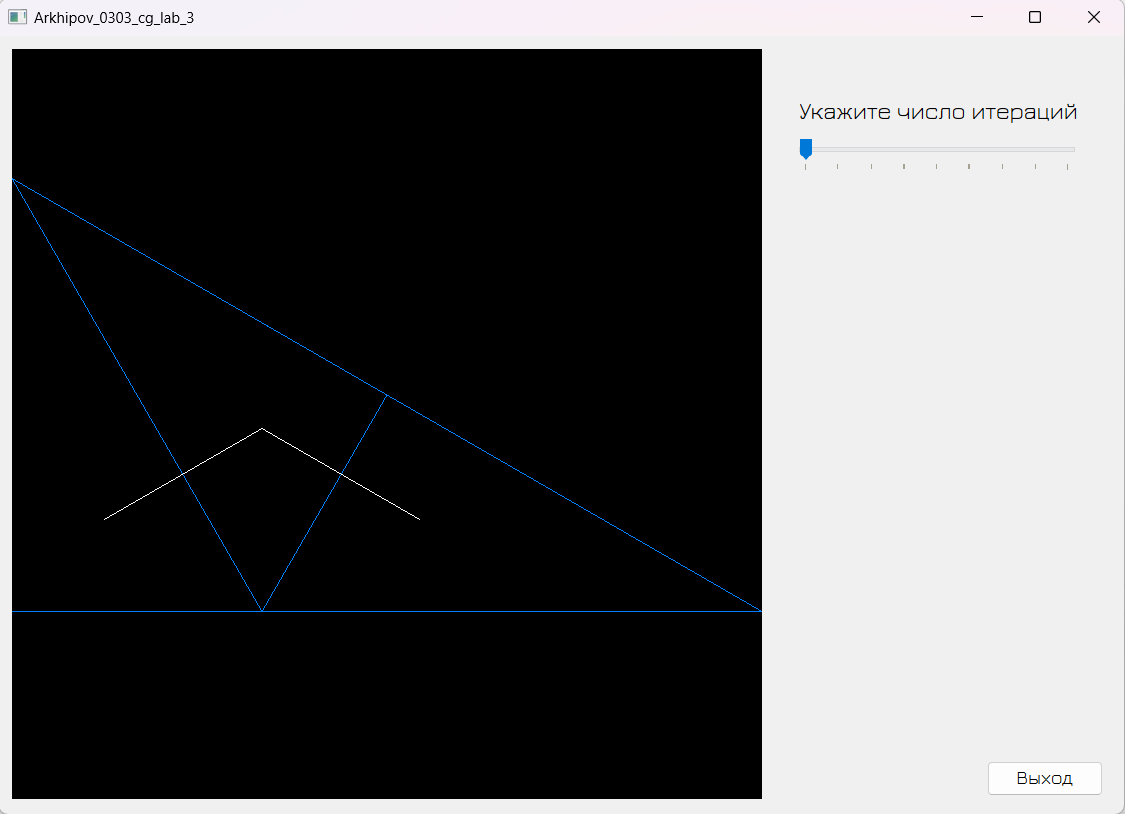


Рисунок 5 – первая итерация

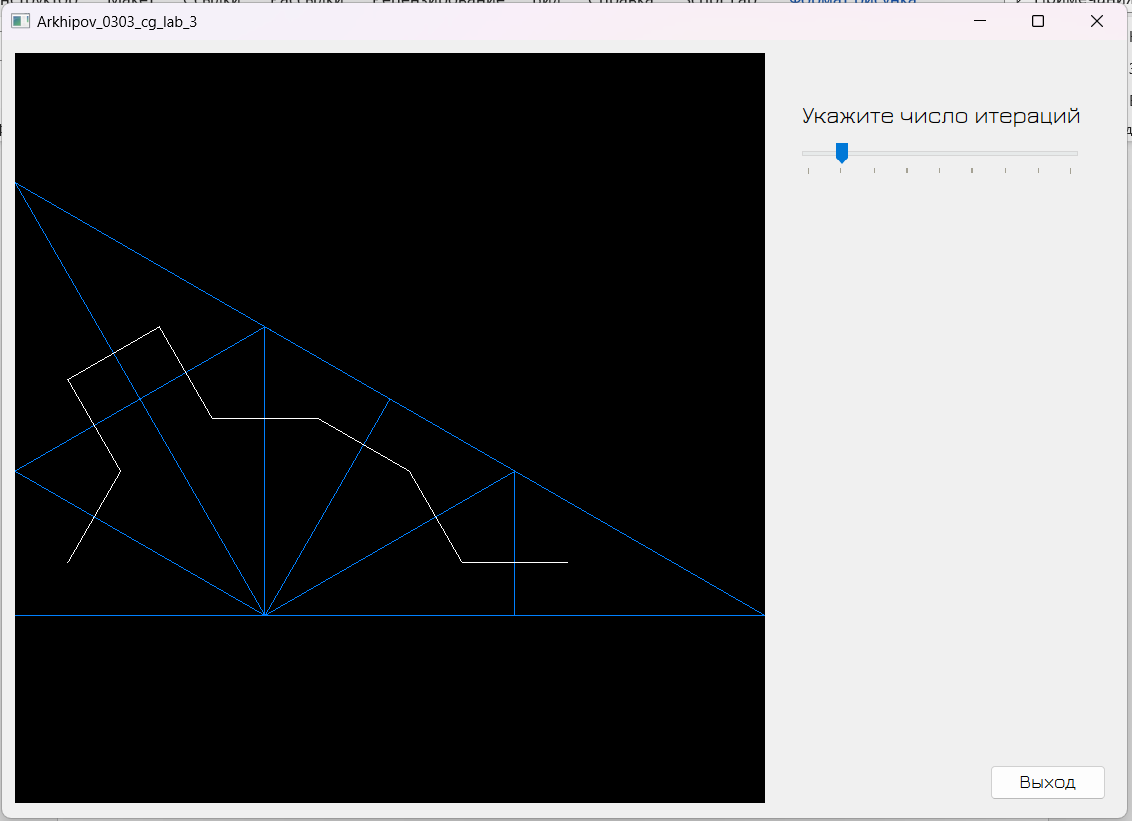


Рисунок 6 – вторая итерация

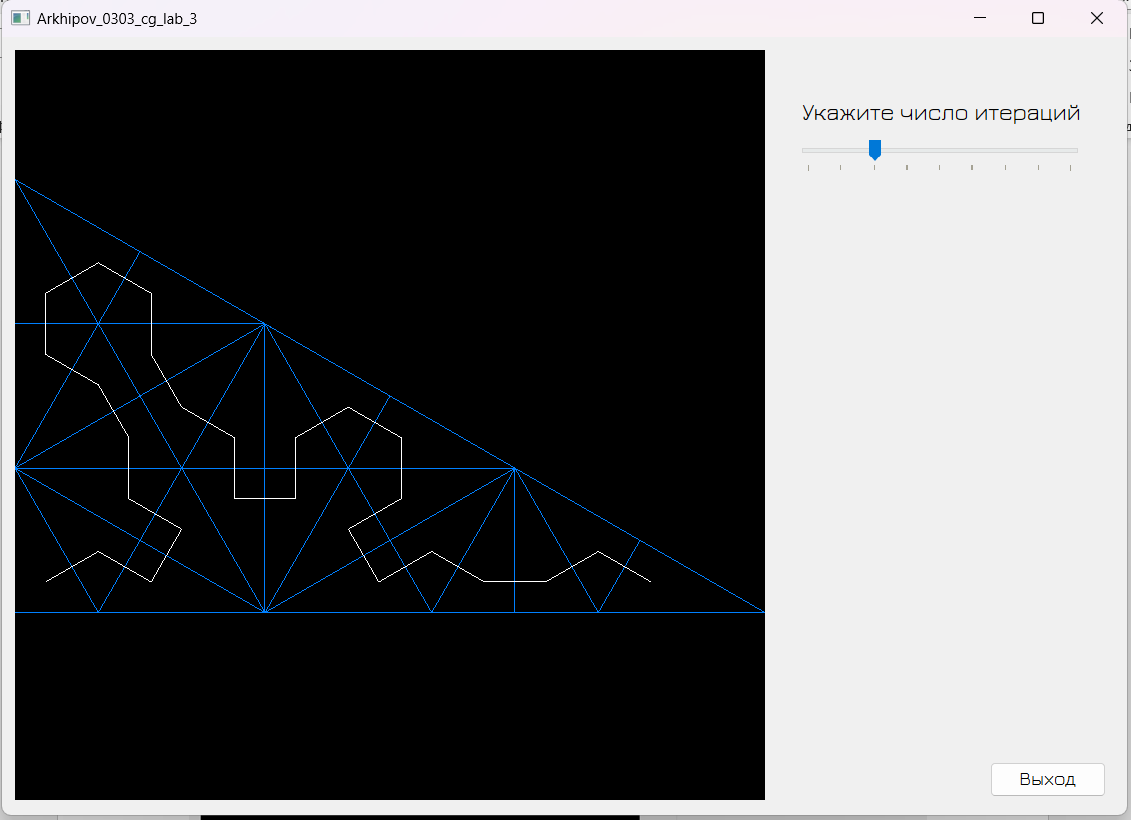


Рисунок 7 – третья итерация

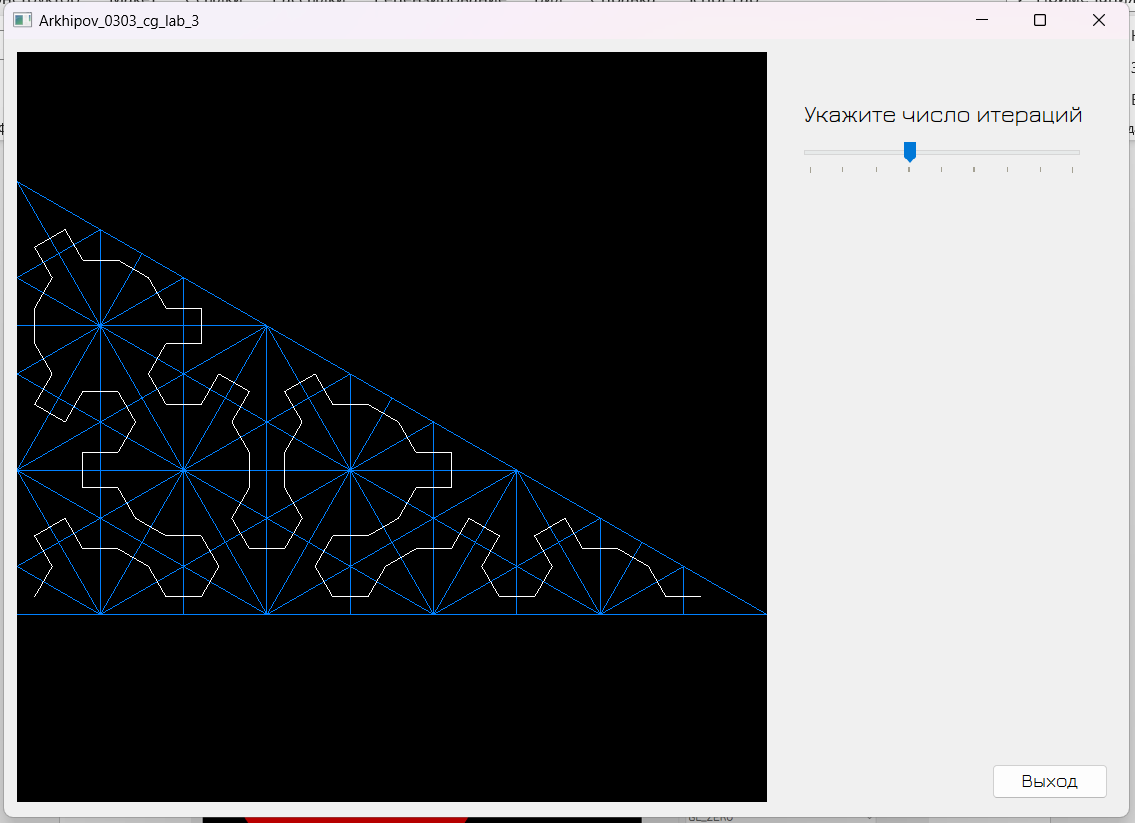


Рисунок 8 – четвертая итерация

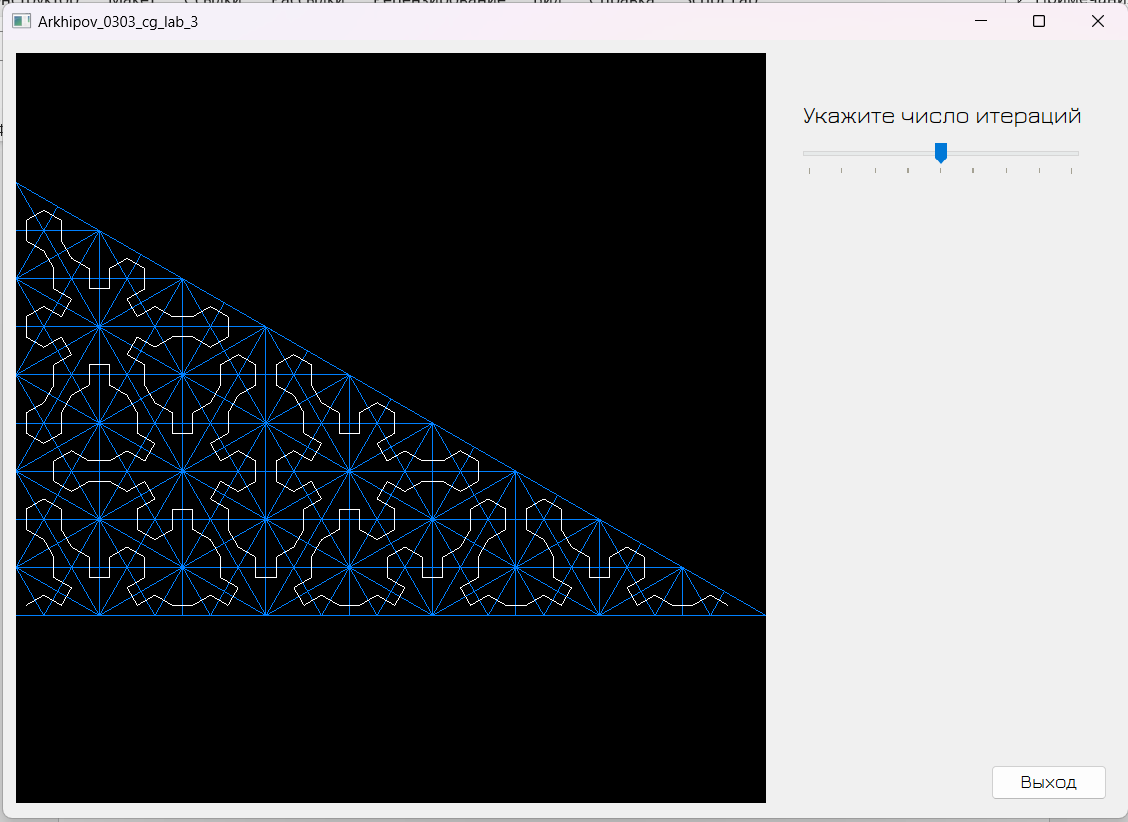


Рисунок 9 – пятая итерация

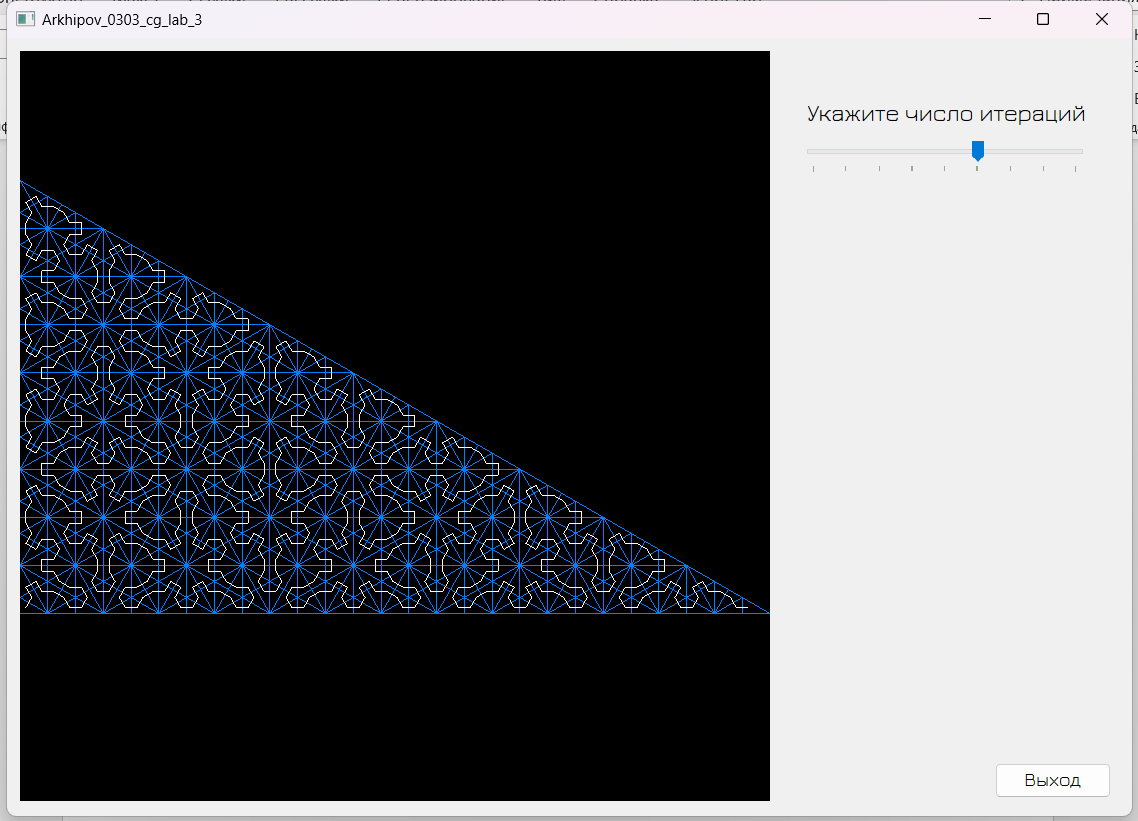


Рисунок 10 – шестая итерация

**Вывод.**

В результате выполнения лабораторной работы была разработана программа, реализующая графическое представление заданного фрактала.

**ПРИЛОЖЕНИЕ А**

**РАЗРАБОТАННЫЙ ПРОГРАММНЫЙ КОД**

Название файла: glWidget.py

from PyQt6.QtOpenGLWidgets import QOpenGLWidget  
from OpenGL import GL as gl  
from PyQt6 import QtCore  
from geometry import median\_dots, bisectors\_intersection\_dot, divide\_cutoff  
  
  
class glWidget(QOpenGLWidget):  
 def \_\_init\_\_(self, main\_window):  
 super().\_\_init\_\_(parent=main\_window.ui.centralwidget)  
 *# Ссылка на родительское окно* self.mw = main\_window  
 *# Массив точек, которые мы будем соединять* self.points = []  
 *# Старые треугольники* self.old\_triangles = []  
 *# Новые треугольники* self.new\_triangles = []  
 *# начальный треугольник* self.start = [[-1, 2 / 3 \*\* (1 / 2) - 1 + 0.5], [-1, -1 + 0.5], [1, -1 + 0.5]]  
  
 main\_window.ui.openGLWidget = self  
 main\_window.ui.openGLWidget.setGeometry(QtCore.QRect(10, 10, 600, 600))  
 main\_window.ui.openGLWidget.setObjectName("openGLWidget")  
  
 *# Настройка состояния. Вызывается один раз в самом начале* def initializeGL(self):  
 gl.glClearColor(0, 0, 0, 1)  
 gl.glClear(gl.GL\_COLOR\_BUFFER\_BIT | gl.GL\_DEPTH\_BUFFER\_BIT)  
  
 def paintGL(self):  
 self.points = []  
 self.old\_triangles = []  
 self.new\_triangles = []  
  
 self.draw\_fractal(self.start, self.mw.ui.horizontalSlider.value())  
  
 *# Рисование треугольников* gl.glBegin(gl.GL\_LINE\_STRIP)  
 gl.glColor3f(0.00, 0.5, 1)  
 for triangle in self.old\_triangles:  
 for point in triangle:  
 gl.glVertex2fv(point)  
 for point in self.start:  
 gl.glVertex2fv(point)  
 gl.glEnd()  
  
 *# Рисование точек* gl.glColor3f(1, 1, 1)  
 gl.glBegin(gl.GL\_LINE\_STRIP)  
 for point in self.points:  
 gl.glVertex2fv(point)  
 gl.glEnd()  
  
 def count\_triangles(self, dots):  
 return [  
 [divide\_cutoff(dots[1], dots[2]), dots[1], dots[0]],  
 [divide\_cutoff(dots[1], dots[2]), median\_dots([dots[0], dots[2]]), dots[0]],  
 [divide\_cutoff(dots[1], dots[2]), median\_dots([dots[0], dots[2]]), dots[2]]  
 ]  
  
 def draw\_fractal(self, dots, iterator):  
 self.old\_triangles.append(dots)  
 for \_ in range(iterator):  
 for index,triangle in enumerate(self.old\_triangles):  
 res = self.count\_triangles(triangle)  
 if index % 2 != 0:  
 res.reverse()  
 self.new\_triangles += res  
 self.old\_triangles = self.new\_triangles  
 self.new\_triangles = []  
 for triangle in self.old\_triangles:  
 self.points.append(bisectors\_intersection\_dot(triangle))

Название файла: MainWindow.py

import sys  
from PyQt6.QtWidgets import QMainWindow, QRubberBand  
from glWidget import glWidget  
from ui\_mainwindow import Ui\_MainWindow  
  
class MainWindow(QMainWindow):  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.ui = Ui\_MainWindow()  
 self.ui.setupUi(self)  
  
 *# Создание виджета для отображения графики* self.ui.openGLWidget = glWidget(self)  
  
 *#Обработка событий* self.ui.pushButton.clicked.connect(self.quit)  
 self.ui.horizontalSlider.valueChanged.connect(self.indicate)  
  
 *# Демонстрация окна* self.show()  
  
 def quit(self):  
 sys.exit()  
  
 def indicate(self):  
 self.ui.openGLWidget.update()

Название файла: geometry.py

import numpy as np  
import math  
  
*# Поиск длины отрезка по его координатам*def find\_len(point1, point2):  
 return math.sqrt((point1[0] - point2[0]) \*\* 2 + (point2[1] - point1[1]) \*\* 2)  
  
*# Поиск середины отрезка cut*def median\_dots(cut):  
 return (cut[0][0] + cut[1][0]) / 2, (cut[0][1] + cut[1][1]) / 2  
  
*# Поиск точки пересечения биссектрис треугольника с заданными вершинами (центр вписанной окружности)*def bisectors\_intersection\_dot(triangle\_dots):  
 A = triangle\_dots[0]  
 B = triangle\_dots[1]  
 C = triangle\_dots[2]  
 O = [0,0]  
  
 OA = np.array([A[0] - O[0], A[1] - O[1]])  
 OB = np.array([B[0] - O[0], B[1] - O[1]])  
 OC = np.array([C[0] - O[0], C[1] - O[1]])  
 a = find\_len(B, C)  
 b = find\_len(A, C)  
 c = find\_len(A, B)  
 OI = (a \* OA + b \* OB + c \* OC) / (a + b + c)  
 return OI[0], OI[1]  
  
*# Найти точку, которая делит заданный отрезок в определённом соотношении*def divide\_cutoff(A, B, ratio = 1/2):  
 x = (A[0] + ratio \* B[0]) / (1 + ratio)  
 y = (A[1] + ratio \* B[1]) / (1 + ratio)  
 return x,y

Название файла: main.py

import sys  
from PyQt6.QtWidgets import QApplication  
from MainWindow import MainWindow  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 main\_window = MainWindow()  
 main\_window.setFixedSize(900, 622)  
 sys.exit(app.exec())

Название файла: ui\_mainwindow.py

*# Form implementation generated from reading ui file 'ui\_mainwindow.ui'  
#  
# Created by: PyQt6 UI code generator 6.4.2  
#  
# WARNING: Any manual changes made to this file will be lost when pyuic6 is  
# run again. Do not edit this file unless you know what you are doing.*from PyQt6 import QtCore, QtGui, QtWidgets  
  
  
class Ui\_MainWindow(object):  
 def \_\_init\_\_(self):  
 self.openGLWidget = None  
  
 def setupUi(self, MainWindow):  
 MainWindow.setObjectName("MainWindow")  
 MainWindow.resize(900, 622)  
 self.centralwidget = QtWidgets.QWidget(parent=MainWindow)  
 self.centralwidget.setObjectName("centralwidget")  
 self.label = QtWidgets.QLabel(parent=self.centralwidget)  
 self.label.setGeometry(QtCore.QRect(640, 40, 271, 41))  
 font = QtGui.QFont()  
 font.setFamily("Jura")  
 font.setPointSize(14)  
 self.label.setFont(font)  
 self.label.setObjectName("label")  
 self.horizontalSlider = QtWidgets.QSlider(parent=self.centralwidget)  
 self.horizontalSlider.setGeometry(QtCore.QRect(640, 80, 221, 31))  
 self.horizontalSlider.setMinimum(1)  
 self.horizontalSlider.setMaximum(9)  
 self.horizontalSlider.setPageStep(1)  
 self.horizontalSlider.setOrientation(QtCore.Qt.Orientation.Horizontal)  
 self.horizontalSlider.setInvertedAppearance(False)  
 self.horizontalSlider.setInvertedControls(False)  
 self.horizontalSlider.setTickPosition(QtWidgets.QSlider.TickPosition.TicksBelow)  
 self.horizontalSlider.setTickInterval(1)  
 self.horizontalSlider.setObjectName("horizontalSlider")  
 self.pushButton = QtWidgets.QPushButton(parent=self.centralwidget)  
 self.pushButton.setGeometry(QtCore.QRect(790, 580, 93, 28))  
 font = QtGui.QFont()  
 font.setFamily("Jura")  
 font.setPointSize(12)  
 self.pushButton.setFont(font)  
 self.pushButton.setObjectName("pushButton")  
 MainWindow.setCentralWidget(self.centralwidget)  
  
 self.retranslateUi(MainWindow)  
 QtCore.QMetaObject.connectSlotsByName(MainWindow)  
  
 def retranslateUi(self, MainWindow):  
 \_translate = QtCore.QCoreApplication.translate  
 MainWindow.setWindowTitle(\_translate("MainWindow", "Arkhipov\_0303\_cg\_lab\_3"))  
 self.label.setText(\_translate("MainWindow", "Укажите число итераций"))  
 self.pushButton.setText(\_translate("MainWindow", "Выход"))