**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 |  | Болкунов В.О. |
| Преподаватель |  | Герасимова Т.В. |

Санкт-Петербург

2023

**Задание**

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

**Общие сведения**

"Среди всех картинок, которые может создавать компьютер, лишь немногие могут поспорить с фрактальными изображениями, когда идет речь о подлинной красоте. У большинства из нас слово "фрактал" вызывает в памяти цветные завитушки, формирующие сложный, тонкий и составной узор."

Из книги Джефа Проузиса "Как работает компьютерная графика"

**Понятие «Фрактала»**

Понятия фрактал и фрактальная геометрия, появившиеся в конце 70-х, с середины 80-х прочно вошли в обиход математиков и программистов. Слово фрактал образовано от латинского **fractus** и в переводе означает состоящий из фрагментов. Оно было предложено Бенуа Мандельбротом в 1975 году для обозначения нерегулярных, но самоподобных структур, которыми он занимался. Рождение фрактальной геометрии принято связывать с выходом в 1977 году книги Мандельброта `The Fractal Geometry of Nature'. В его работах использованы научные результаты других ученых, работавших в период 1875-1925 годов в той же области (Пуанкаре, Фату, Жюлиа, Кантор, Хаусдорф). Но только в наше время удалось объединить их работы в единую систему.

Фрактал (лат. fractus — дробленый) — термин, означающий геометрическую фигуру, обладающую свойством самоподобия, то есть составленную из нескольких частей, каждая из которых подобна всей фигуре целиком.

Существует большое число математических объектов называемых фракталами (треугольник Серпинского, снежинка Коха, кривая Пеано, множество Мандельброта). Фракталы с большой точностью описывают многие физические явления и образования реального мира: горы, облака, турбулентные (вихревые) течения, корни, ветви и листья деревьев, кровеносные сосуды, что далеко не соответствует простым геометрическим фигурам.

**Классификация фракталов.**

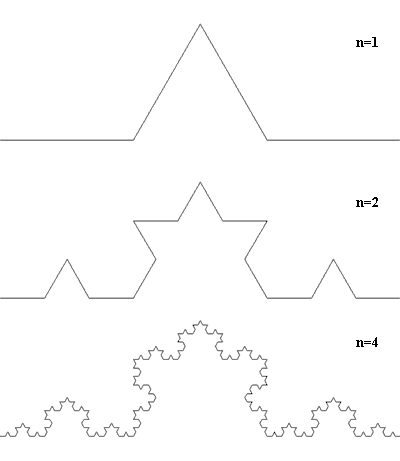
*1. Геометрические фракталы.*

Фракталы этого класса самые наглядные. В двухмерном случае их получают с помощью ломаной (или поверхности в трехмерном случае), называемой генератором. За один шаг алгоритма каждый из отрезков, составляющих ломаную, заменяется на ломаную-генератор в соответствующем масштабе. В результате бесконечного повторения этой процедуры получается геометрический фрактал.

Рассмотрим на примере один из таких фрактальных объектов – триадную кривую Коха.

*Построение триадной кривой Коха.*

Возьмем прямолинейный отрезок длины 1. Назовем его затравкой. Разобьем затравку на три равные части длиной в 1/3, отбросим среднюю часть и заменим ее ломаной из двух звеньев длиной 1/3.



Построение триадной кривой Коха

Мы получим ломаную, состоящую из 4 звеньев с общей длиной 4/3 , - так называем первое поколение. Для того чтобы перейти к следующему поколению кривой Коха, надо у каждого звена отбросить и заменить среднюю часть. Соответственно длина второго поколения будет 16/9, третьего – 64/27. если продолжить этот процесс до бесконечности, то в результате получится

*Особенности триадной кривой Коха:*

Во-первых, эта кривая не имеет длины – с числом поколений ее длина стремится к бесконечности.

Во-вторых, к этой кривой невозможно построить касательную – каждая ее точка является точкой перегиба, в которой производная не существует, - эта кривая не гладкая.

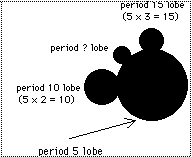
В-третьих, к триадной кривой Коха традиционные методы геометрического анализа оказались неприменимы.

*2. Алгебраические фракталы*

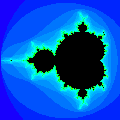
Это самая крупная группа фракталов. Получают их с помощью нелинейных процессов в n-мерных пространствах. Наиболее изучены двухмерные процессы. В качестве примера рассмотрим множество Мандельброта.

Математическое описание модели следующее: на комплексной плоскости в неком интервале для каждой точки ***с*** вычисляется рекурсивная функция ***Z=Z2+c.*** В модели Мандельброта изменяющимся фактором является начальная точка ***с,*** а параметр ***z,*** является зависимым.

Графическая реализация: начальная точка модели равна нулю. Графически она соответствует центру тела “груши”. Через N шагов заполнятся все тело груши и в том месте, где закончилась последняя итерация, начинает образовываться “голова” фрактала. “Голова” фрактала будет ровно в четыре раза меньше тела, так как математическая формула фрактала представляет из себя квадратный полином. Затем опять через N итераций у “тела” начинает образовываться “почка” (справа и слева от “тела”). И так далее. Чем больше задано числе итераций N, тем более детальным получится изображение фрактала, тем больше будет у него различных отростков. Схематическое изображение стадий роста фрактала Мандельброта представлено на рис.2:



**Схема образования фрактала Мандельброта**



**компьютерное изображение фрактала Мандельброта**

*3. Стохастические (случайные) фракталы*

Еще одним известным классом фракталов являются стохастические фракталы, которые получаются в том случае, если в итерационном процессе хаотически менять какие-либо его параметры. При этом получаются объекты очень похожие на природные - несимметричные деревья, изрезанные береговые линии и т.д. Двумерные стохастические фракталы используются при моделировании рельефа местности и поверхности моря. Примерами стохастических фракталов являются фрактальные кривые, возникающие в критических двумерных моделях статистической механики, траектория броуновского движения на плоскости и в пространстве, плазма.

## Способы построения фракталов

### L-система

L-система (от имени Lindenmayer) - это грамматика некоторого языка (достаточно простого), которая описывает инициатор и преобразование, выполняемое над ним, при помощи средств, аналогичных средствам языка Лого (аксиоматическое описание простейших геометрических фигур и допустимых преобразований на плоскости и в пространстве).

L-системы часто называются ещё и системами черепашьей графики. Черепашья графика - это такой способ рисования линий на экране компьютера. Он состоит в том, что программист как бы управляет движением как бы черепашки. Черепашка, ползая по экрану, оставляет за собой след. При этом цель программиста – управлять черепашкой так, чтобы черепашка нарисовала нужную линию. Команды управления черепашкой просты: сделать шаг вперёд (обозначается F), повернуть направо (обозначается +), повернуть налево (обозначается -), сделать шаг вперёд без перерисовки (прыжок, обозначается B). Вот из этих команд и составляется сценарий построения линии – строка команд. Величина одного шага и угол одного поворота при движении черепашки всегда остаются постоянными и задаются предварительно.

Например, F++F++F это равносторонний треугольник, если угол поворота равен pi/3, а F+F+F+F – это квадрат, если угол поворота равен pi/2 .

Построение L-системы происходит в три этапа.

Сначала создаётся сценарий поведения черепашки.

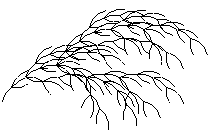
Потом подсчитывается размер линии, которая получится, если запустить этот сценарий на исполнение. Линия как бы рисуется в уме, а потом смотрится её размер. На основе этого размера подправляется масштаб, чтобы вся линия уместилась на экране.

На экран запускается черепашка, которая рисует линию.

Фракталы – самоподобные фигуры, значит, и сценарии у них должны быть самоподобные. Вот как это делается.

* Берётся начальная фигура (называется - аксиома), например, F++F++F с углом pi/3.
* Задаётся правило замены F (называется правило newF) , например, new F = F-F++F-F;
* в имеющейся фигуре все F заменяются на newF.

У деревьев есть ветки. Тут имеющимся набором команд не обойдёшься. Приходится вводить ещё пару команд: начало ветви (обозначается [ ), конец ветви (обозначается ] ). Что бедная черепашка должна делать по этим командам? В начале ветви она должна запомнить своё состояние (положение и направление взгляда), а вот когда ей встретится соответствующий конец ветви, она должна вернуться в то положение, которое запомнила.

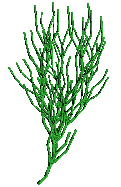
Вот, например, данные для построения куста.

axiom = F

newF = -F+F+[+F-F-]-[-F+F+F]

turn = pi / 8

Если вы хотите строить объёмные деревья и другие фракталы, то вам придётся ввести ещё пару команд – поворотов из плоскости (предлагаю символы >, <). Короче, + - это повороты в плоскости XY, а >, < это повороты в плоскости XZ.

 Вот, например, данные для построения 3d-кустика.

axiom = F

newF = -F+F[>F+F<F-F][<F-F>F+F]

[+F<F-F>F][-F>F+F<F]

turn = pi / 8

Для двух шагов получите вот такое деревце --->Для трёх шагов получим такую картинку ->

|  |  |
| --- | --- |
|  |  |

### 4.3.2. Система итерирующих функций IFC

Система итерирующих функций IFC Применение таких преобразований, которые дают ту фигуру которую необходимо. Система итерирующих функций - это совокупность сжимающих аффинных преобразований. Как известно, аффинные преобразования включают в себя масштабирование, поворот и параллельный перенос. Аффинное преобразование считается сжимающим, если коэффициент масштабирования меньше единицы.

Рассмотрим подробнее построение кривой Кох с использованием аффинных преобразований. Каждый новый элемент кривой содержит четыре звена, полученных из образующего элемента использованием масштабирования, поворота и переноса.

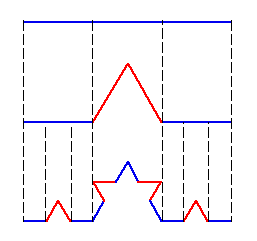
1. Для получения первого звена достаточно сжать исходный отрезок в три раза. Следует отметить, что тоже масштабирование применяется для всех звеньев.

2. Следующее звено строится с использованием всех возможных преобразований, а именно: сжатие в три раза, поворот на - 60о и параллельный перенос на 1/3 по оси X.

3. Третье звено строится аналогично второму: сжатие в три раза, пово-рот на 60о, параллельный перенос на 2/3 по оси X.

4. Последнее звено: сжатие в три раза, параллельный перенос на 2/3 по оси X.

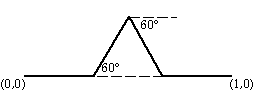
В дальнейшем правила построения кривой Кох будем называть IFS дл кривой Кох.

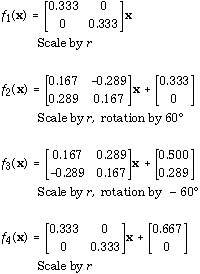
 На первой итерации кривая состоит из 4 фрагментов с коэффициентом сжатия r =1/3, два сегмента повернуты на 60 град. по час. и против час. ст.

f1(x) -> масшт. на r

f2(x) -> масшт. на r, поворот на 600

f3(x) -> масшт. на r, поворот на - 600

f4(x) -> масшт. на r ****

****

В результате выполнения лабораторной работы должна быть разработана программа, реализующая представление заданного фрактала

# Конкретный фрактал для реализации

### Задание 53



**Выполнение работы.**

Заданный фрактал поддаётся удобному алгебраическому описанию: для его построения достаточно найти точки шестиугольников на разных уровнях (до определённого максимального уровня N – заданного количества итераций)

Тогда координату j-ой точки шестиугольника на уровне i можно найти по формуле:

Для соединения точек линиямив данной работе были проведены 2 линии от каждой точке к двум соседним на этом же уровне, на расстоянии в 1 шаг: то есть точка соединяется с точками и ; и также с двумя соседними но из предыдущего уровня: и , однако благодаря возможности обратной индексации в языке python, эти формулы можно упростить, что и было написано в реализации алгоритма.

Для выполнения работы были использованы язык Python 3.10 и библиотеки PyQt6, numpy и PyOpenGL.

Подключение графической библиотеки осуществляется с помощью виджета QOpenGLWidget

В нашем случае был создан виджет GLWidget наследуемый от данного класса, в котором с помощью переопределённых методов **initializeGL и paintGL** осуществляется соответственно подготовка кадра и отрисовка изображения. Сама отрисовка для гибкости использования осуществляется задаваемой функцией в поле **function**. В переопределённом методе resizeGL отслеживается изменение размера окна, устанавливается Viewport и посылается сигнал об изменении размера области отрисовки.

Задавав функцию рисования и вызвав метод **update** у GLWidget можно добиться рисования любых объектов в соответствии с заданной функцией.

В модуле drawing описаны функции для генерации и рисования необходимых объектов, перечислим содержимое данного модуля:

* Нормализованные векторы точек шестиугольника используемые для генерации шестиугольников на заданных уровнях

hexagon = np.array([np.array([np.cos(t), np.sin(t)]) for t in np.linspace(0, 2 \* np.pi, 7)[:-1] + np.pi / 6])

* Генерация n шестиугольников с радиусом r (расстоянием между соседними уровнями) по описанным выше формулам нахождения точек

def generate(n, r):

* Цветовая палитра фрактала

colors = [[0.06, 0.28, 0.66], [0.78, 0.0, 0.49], [0.66, 0.94, 0.0], [1.0, 0.65, 0.0]]

* Нормализованные векторы точек окружности для рисования окружностей в вершинах шестиугольников

circleVecs = [np.array([np.cos(t), np.sin(t)]) for t in np.linspace(0, 2 \* np.pi, 50)]

* Рисование окружности радиуса r в точке p и заданным цветом

def drawCircle(r, p, color):

* Рисование линий между точками сгенерированных шестиугольников по описанному алгоритму соединения точек

def drawLines(dots):

Управление приложением осуществляется с помощью виджета **ControlPanel**, который содержит в себе два ползунка для регулирования количества уровней фрактала и для управления масштабом отображения фрактала (расстоянием между уровнями). Также ширина линий напрямую зависит от выбранного радиуса, что позволяет удобно отобразить рисунок с эффектом перспективы.

Элементы управления и графический виджет объединены компонентом **MainWindow** (наследуемом от QMainWindow), в нём происходит связывание событий интерфейса управления с обновлениями изображения.

**Тестирование**

Возможные изображения интерфейса программы представлены на рисунках 1, 2.

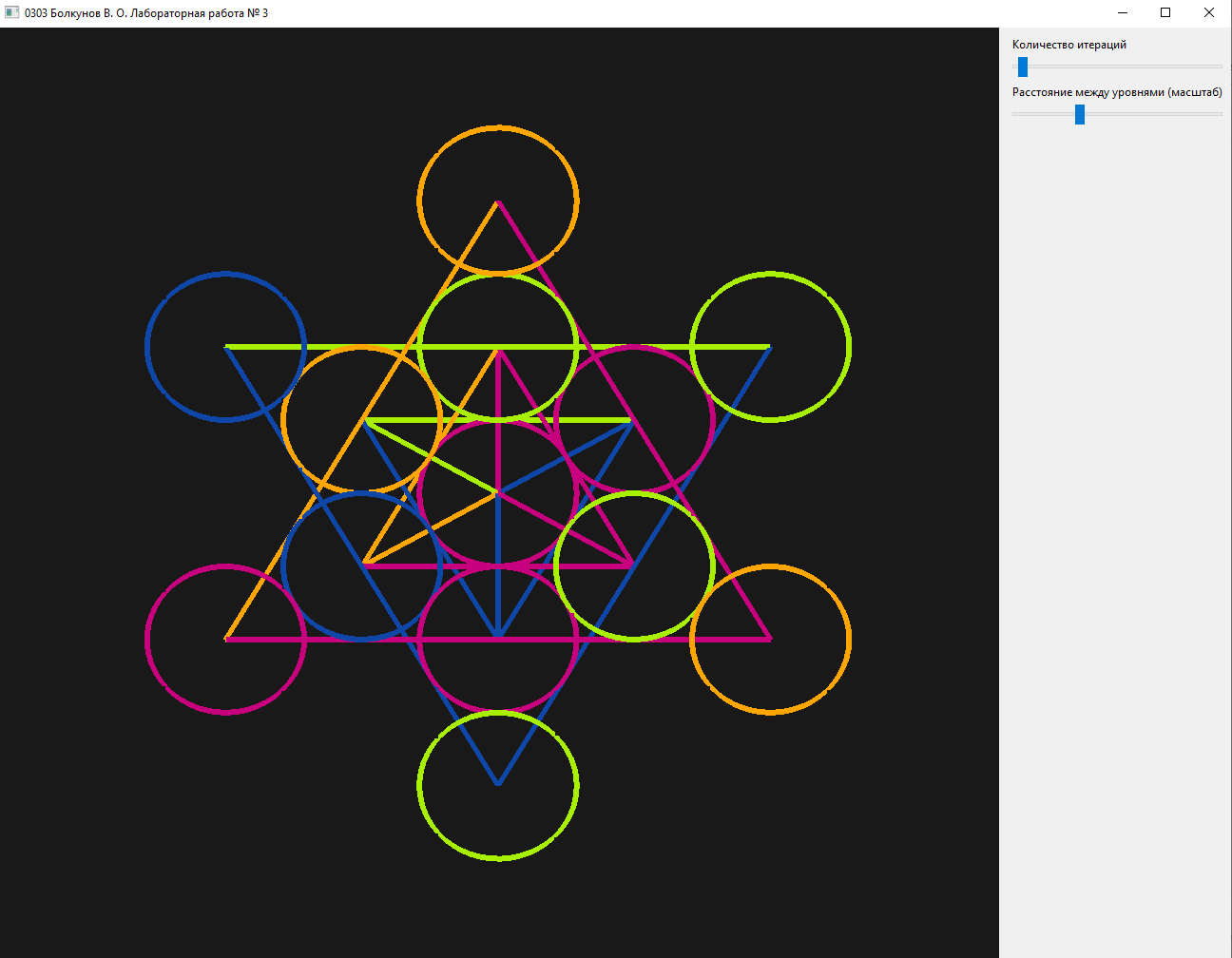


Рисунок 1: несколько уровней фрактала в приближении

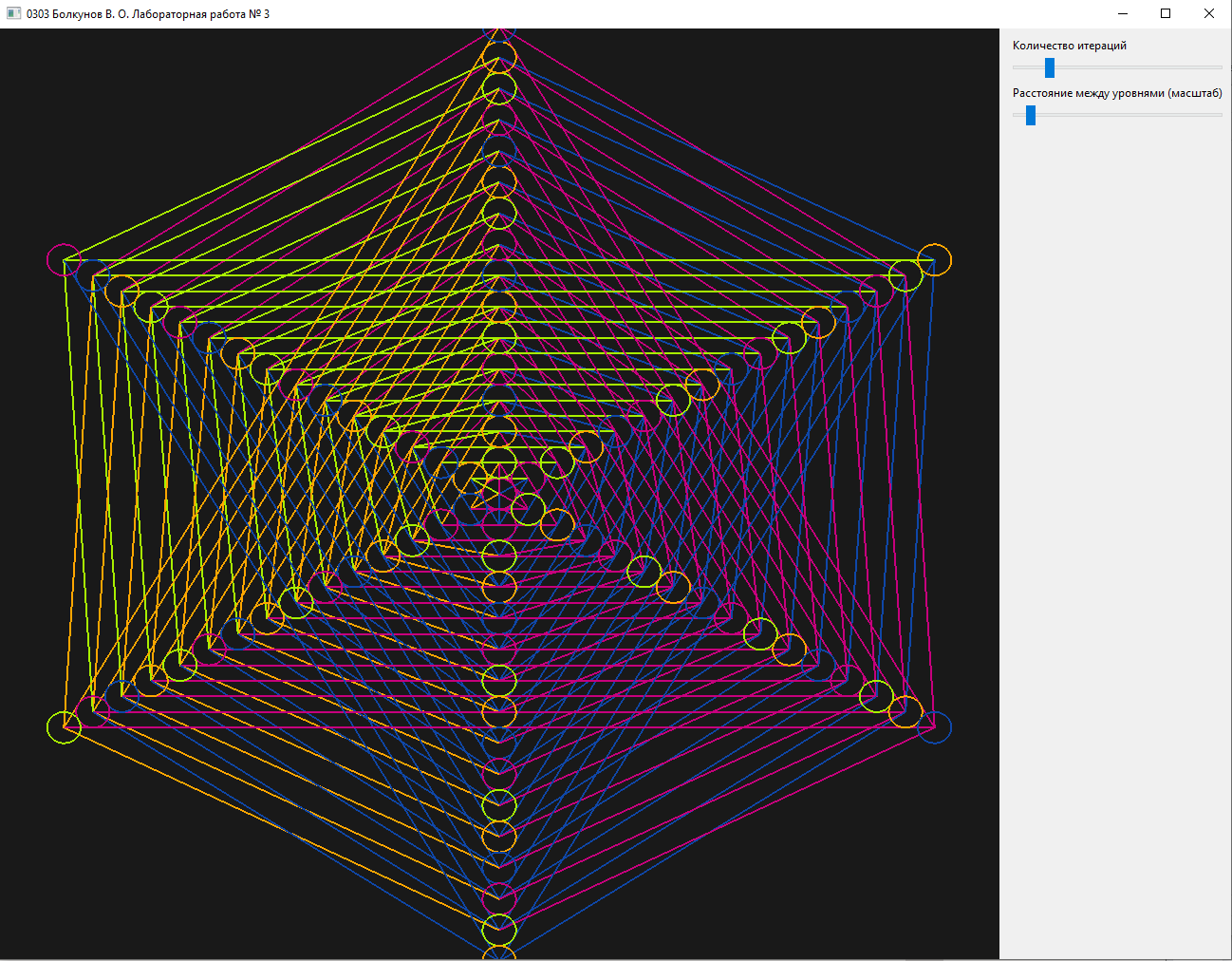


Рисунок 2: множество уровней фрактала в отдалении

**Выводы:**

В ходе лабораторной работы были изучены алгоритмы построения фрактальных изображений, был описан алгоритм построения конкретного фрактала и реализован алгоритм его генерации и рисования с помощью средств библиотек OpenGL и numpy. Отображение контекста рисования OpenGL и настройка фрактала с помощью элементов интерфейса были реализованы с помощью библиотеки пользовательского графического интерфейса PyQt6

**ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД**

***Файл main.py***

import sys  
  
from PyQt6.QtWidgets import QApplication  
  
from MainWindow import MainWindow  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 w = MainWindow()  
 w.show()  
 sys.exit(app.exec())

***Файл GLWidget.py***

from OpenGL import GL as gl  
from PyQt6 import QtCore  
from PyQt6.QtOpenGLWidgets import QOpenGLWidget  
  
  
*# Виджет OpenGL*class GLWidget(QOpenGLWidget):  
 viewPortResized = QtCore.pyqtSignal((int, int))  
  
 def \_\_init\_\_(self, parent=None):  
 super().\_\_init\_\_(parent)  
 *# Функция вызываемая в цикле отрисовки (при обновлениях)* self.function = None  
  
 def resizeGL(self, w: int, h: int) -> None:  
 gl.glViewport(0, 0, w, h)  
 self.viewPortResized.emit(w, h)  
  
 *# Функция вызываемая перед любым обновлением* def initializeGL(self):  
 *# Заливка кадра* gl.glClearColor(0.1, 0.1, 0.1, 1)  
 *# Очистка буферов (цвета и глубины)* gl.glClear(gl.GL\_COLOR\_BUFFER\_BIT | gl.GL\_DEPTH\_BUFFER\_BIT)  
  
 *# Функция вызываемая при обновлении (посредством update или при изменении размеров)* def paintGL(self):  
 *# Вызов рендер-функции* if self.function is not None:  
 self.function()

***Файл drawing.py***

from OpenGL import GL as gl  
import numpy as np  
  
*# Нормализованные векторы точек шестиугольника*hexagon = np.array([  
 np.array([np.cos(t), np.sin(t)])  
 for t in np.linspace(0, 2 \* np.pi, 7)[:-1] + np.pi / 6  
])  
  
  
*# Генерация n уровней вложенных шестиугольников на расстоянии r*def generate(n, r):  
 return [i \* r \* hexagon for i in range(n)]  
  
  
*# Палитра цветов*colors = [  
 [0.06, 0.28, 0.66],  
 [0.78, 0.0, 0.49],  
 [0.66, 0.94, 0.0],  
 [1.0, 0.65, 0.0]  
]  
  
*# Векторы точек окружности*circleVecs = [np.array([np.cos(t), np.sin(t)]) for t in np.linspace(0, 2 \* np.pi, 50)]  
  
  
*# Рисование окружности радиуса r в точке p*def drawCircle(r, p, color):  
 gl.glBegin(gl.GL\_LINE\_STRIP)  
 gl.glColor3dv(color)  
 for a in circleVecs:  
 gl.glVertex2dv(p + a \* r)  
 gl.glEnd()  
  
  
*# Рисованиее линий списка уровней шестиугольников*def drawLines(dots):  
 gl.glBegin(gl.GL\_LINES)  
 *# Уровни* for i in range(1, len(dots)):  
 *# Точки в уровне* for j in range(6):  
 gl.glColor3dv(colors[j % 4])  
 *# Соединение точки с двумя другими на своём уровне* for k in range(1, 3):  
 gl.glVertex2dv(dots[i][j])  
 gl.glVertex2dv(dots[i][j - 2 \* k])  
 *# Соединение точки с двумя точками предыдущего уровня* gl.glVertex2dv(dots[i][j])  
 gl.glVertex2dv(dots[i - 1][j - 1])  
 gl.glVertex2dv(dots[i][j])  
 gl.glVertex2dv(dots[i - 1][(j + 1) % 6])  
 gl.glEnd()

***Файл ControlPanel.py***

from PyQt6.QtCore import Qt  
from PyQt6.QtWidgets import QWidget, QVBoxLayout, QSlider, QLabel  
  
  
*# Виджет панели управления*class ControlPanel(QWidget):  
 def \_\_init\_\_(self, parent=None):  
 super().\_\_init\_\_(parent)  
 lt = QVBoxLayout(self)  
 self.setLayout(lt)  
 levelsLabel = QLabel('Количество итераций', self)  
 self.iterations = QSlider(Qt.Orientation.Horizontal, self)  
 radiusLabel = QLabel('Расстояние между уровнями (масштаб)', self)  
 self.radius = QSlider(Qt.Orientation.Horizontal, self)  
 *#* for i in [levelsLabel, self.iterations, radiusLabel, self.radius]:  
 lt.addWidget(i)  
 lt.addStretch()

***Файл MainWindow.py***

from PyQt6 import QtCore  
from PyQt6.QtWidgets import QMainWindow, QSplitter  
from ControlPanel import ControlPanel  
from GLWidget import GLWidget  
from drawing import \*  
  
  
*# Главное окно*class MainWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("0303 Болкунов В. О. Лабораторная работа № 3")  
 self.control = ControlPanel(self)  
 self.glwidget = GLWidget(self)  
 sp = QSplitter(self)  
 sp.addWidget(self.glwidget)  
 sp.addWidget(self.control)  
 sp.setStretchFactor(0, 1)  
  
 self.setCentralWidget(sp)  
 self.resize(900, 600)  
  
 *# Задаём рендер-функцию* self.glwidget.function = self.renderFunction  
  
 self.radiusFraction = 1000  
  
 self.control.radius.setMaximum(self.radiusFraction)  
 self.control.iterations.setMaximum(100)  
 self.control.iterations.setTickInterval(1)  
 self.control.radius.valueChanged.connect(self.redraw)  
 self.control.iterations.valueChanged.connect(self.redraw)  
  
 def renderFunction(self):  
 radius = self.control.radius.value() / self.radiusFraction  
 *# Размытие цвета между вершинами* gl.glShadeModel(gl.GL\_SMOOTH)  
 *# Ширина линий* gl.glLineWidth(1 + 15 \* radius)  
 *# Генерация шестиугольников* dots = generate(self.control.iterations.value(), radius)  
 *# Рисование их линий* drawLines(dots)  
 *# Рисование окружностей в вершинах шестиугольников* for i in range(len(dots)):  
 for p in range(len(dots[i])):  
 drawCircle(radius / 2, dots[i][p], colors[(i + p) % 4])  
  
 *# Вызов обновления изображения* @QtCore.pyqtSlot()  
 def redraw(self):  
 self.glwidget.update()