Санкт-Петербургский Государственный
Электротехнический Университет

Кафедра МОЭВМ

Задание для лабораторной работы № 3
" Построение фракталов "

Выполнил: Калмак Д.А.
Факультет: ФКТИ
Группа: 0303
Преподаватель: Герасимова Т.В.

Санкт-Петербург
2023 г.

Цель работы.

- ознакомление с фракталами.

- проанализировать полученное задание, выделить информационные объекты и действия;

- разработать программу, реализующую фрактал.

Основные теоретические положения.

"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 "фрактал"

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.

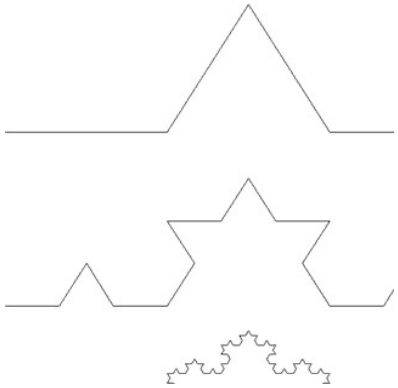
1. Геометрические фракталы.

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 ***c*** вычисляется рекурсивная функция $Z=Z^2+c$. В модели Мандельброта изменяющимся фактором является начальная точка ***c,*** 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:

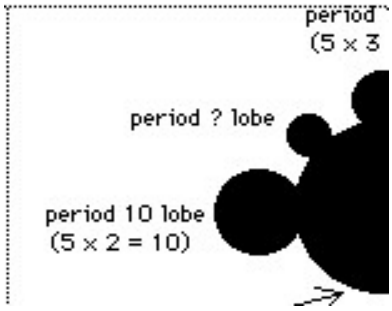
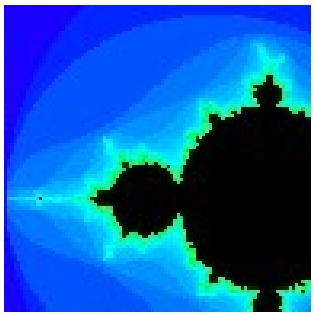


Схема образования фрактала Мандельброта



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.

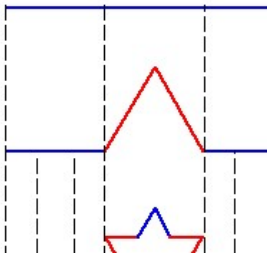
1. Для получения первого звена достаточно сжать исходный отрезок в три раза. Следует отметить, что тоже масштабирование применяется для всех звеньев.

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

3. Третье звено строится аналогично второму: сжатие в три раза, пово-рот на 60°, 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.



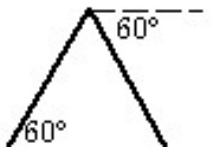
На первой итерации кривая состоит из 4 фрагментов с коэффициентом сжатия r =1/3, два сегмента повернуты на 60 град. по час. и против час. ст.

$f_1(x)$ -> масшт. на r

$f_2(x)$ -> масшт. на r, поворот на 60^0

$f_3(x)$ -> масшт. на r, поворот на - 60^0

$f_4(x)$ -> масшт. на r



$$f_1(\mathbf{x}) = \begin{bmatrix} 0.333 & 0 \\ 0 & 0.333 \end{bmatrix}\mathbf{x}$$

Scale by r

$$f_2(\mathbf{x}) = \begin{bmatrix} 0.167 & -0.289 \\ 0.289 & 0.167 \end{bmatrix}\mathbf{x} + \begin{bmatrix}$$

Scale by r, rotation by

$$f_3(\mathbf{x}) = \begin{bmatrix} 0.167 & 0.289 \\ -0.289 & 0.167 \end{bmatrix}\mathbf{x} + \begin{bmatrix}$$

Scale by r, rotation by

ЗАДАНИЕ.

На базе предыдущей лабораторной работы разработать программу, реализующую фрактал, представленный на рис. 1.

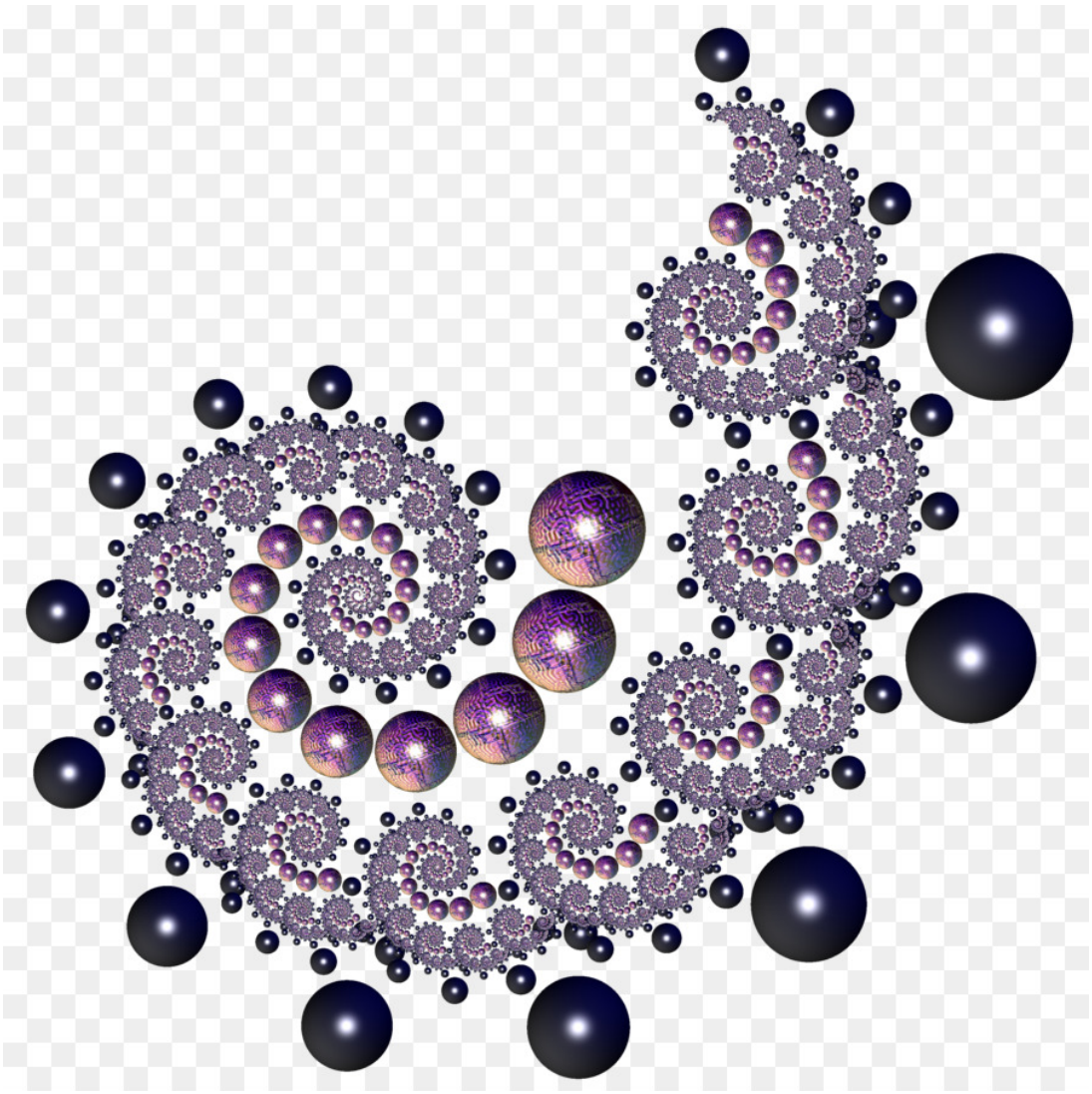


Рисунок 1 - Фрактал

Выполнение работы.

В классе mainWindow в self.stack (QStackedWidget) добавлен виджет класса glWidgetFractal, который наследуется от glWidget0. В классе mainWindow добавлен атрибут self.lblrecursiondepth, который принадлежит классу QLabel. Он содержит текст для ползунка с регулировкой глубины рекурсии. Ползунок self.sliderrecdepth принадлежит классу QSlider в горизонтальном виде с помощью параметра Qt.Orientation.Horizontal. Заданы минимальное и максимальное значения для

ползунка с помощью методов setMinimum и setMaximum у self.sliderrecdepth. Двигая ползунок, с помощью метода valueChanged у self.sliderrecdepth, который передает значение ползунка, и метода connect, который связывает ползунок с методом update_recdepth, метод update_recdepth в классе mainWindow получает значения с ползунка. В методе для всех виджетов в self.stack, обращение к которым осуществляется с помощью метода widget, обновляется значение атрибута self.level, который добавлен в класс glWidget0, от которого наследуются класс glWidgetFractal. Происходит обновление виджетов с помощью метода updateGL. В слой buttonsLayout добавлены виджеты self.lblrecursiondepth, self.sliderrecdepth.

В виджете класса glWidgetFractal в методе paintGL осуществляется смещение координат с помощью функции glTranslatef(). Запускается рекурсивный метод self.draw_spiral(0, 1, 1, []), добавленный в класс glWidgetFractal. В него переданы начальный уровень рекурсии level, начальное масштабирование по x scalex, начальное масштабирование по y scaley, начальный список с точками xascop. В методе draw_spiral сначала происходит проверка глубины рекурсии. Переданное значение level в метод сравнивается с атрибутом класса self.level, которое управляется ползунком. Если значения совпадают, то происходит возврат из метода. Логарифмическая спираль задается с помощью следующих уравнений: $x(t) = r\cos t = ae^{bt}\cos t\,,$ и $y(t) = r\sin t = ae^{bt}\sin t\,,$. Заданы параметры a, b, k (замена t), h шаг для роста k, r для радиуса сферы, count счетчик сфер, countx счетчик смещения по оси x, county счетчик смещения по оси y, xy1 список для точек центров сфер. Запускается цикл, который выполняется до тех пор, пока k не достигнет определенного значения. В цикле по формулам рассчитываются координаты сфер x и y, которые добавляются в список xy1. Смещаются оси координат с помощью функции glTranslatef() для отрисовки сферы. Обновляются

счетчики countx и county сложением с координатами x и y. Чтобы отрисовать сферу используется переменная quadObj, равная возвращаемому значению функции gluNewQuadric(), а также функция gluSphere(), которая принимает переменную quadObj, r радиус, количество подразделений вокруг оси Z, количество подразделений вдоль оси Z. К k прибавляется шаг h, к r прибавляется шаг, счетчик обновляется. Для оптимизации используется функция glFlush(). Перед отрисовкой смещаются оси с помощью функции glTranslatef() на значения -countx и -county. Аналогично первой спирали отрисовывается вторая. У нее свои значения a2, b2, xy2. Вызывается функция glLoadIndentity() для установления единичной матрицы. Список xas содержит середины отрезков между сферами. Список будет принимать значение xascop, переданное в метод draw_spiral(). Переменная count_rotate для регулировки поворота. x_scale и y_scale присваиваются значения scalex и scaley, умноженные на 0.2 для регулировки масштаба. Запускается цикл по длине списка xas. Смещаются оси координат на значения элемента списка xas, содержащего координаты для сфер. Осуществляется поворот с помощью функции glRotate(). Создается список xascopy, в котором перерассчитываются координаты для сфер с учетом масштабирования, поворота относительно точки центра новой системы координат. Происходит масштабирование с помощью функции glScale(x_scale, y_scale, 0). Запускается рекурсивный метод self.draw_spiral(level + 1, x_scale, y_scale, xascopy). Вызывается функция glLoadIdentity() для установления единичной матрицы. Обновляется счетчик для регуляции поворота.

ТЕСТИРОВАНИЕ.

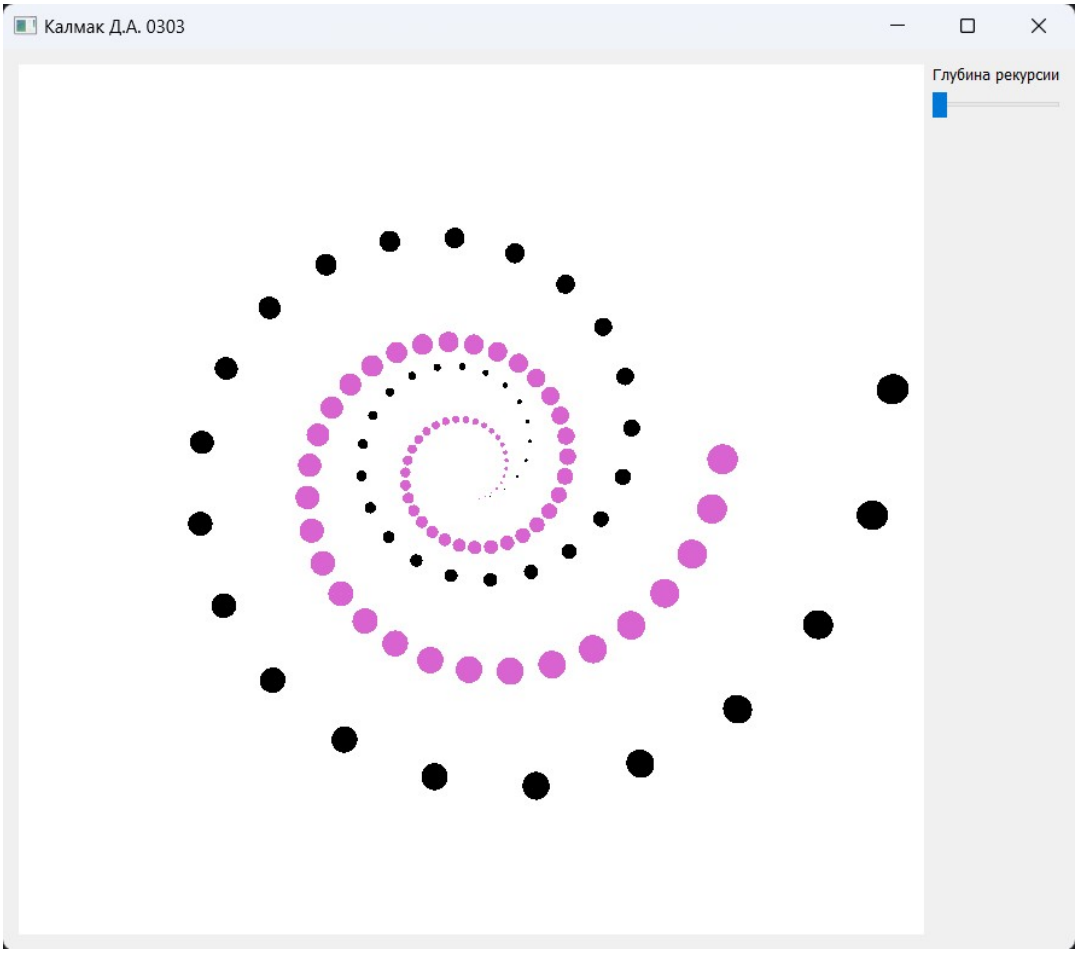


Рисунок 2 – Фрактал начального уровня

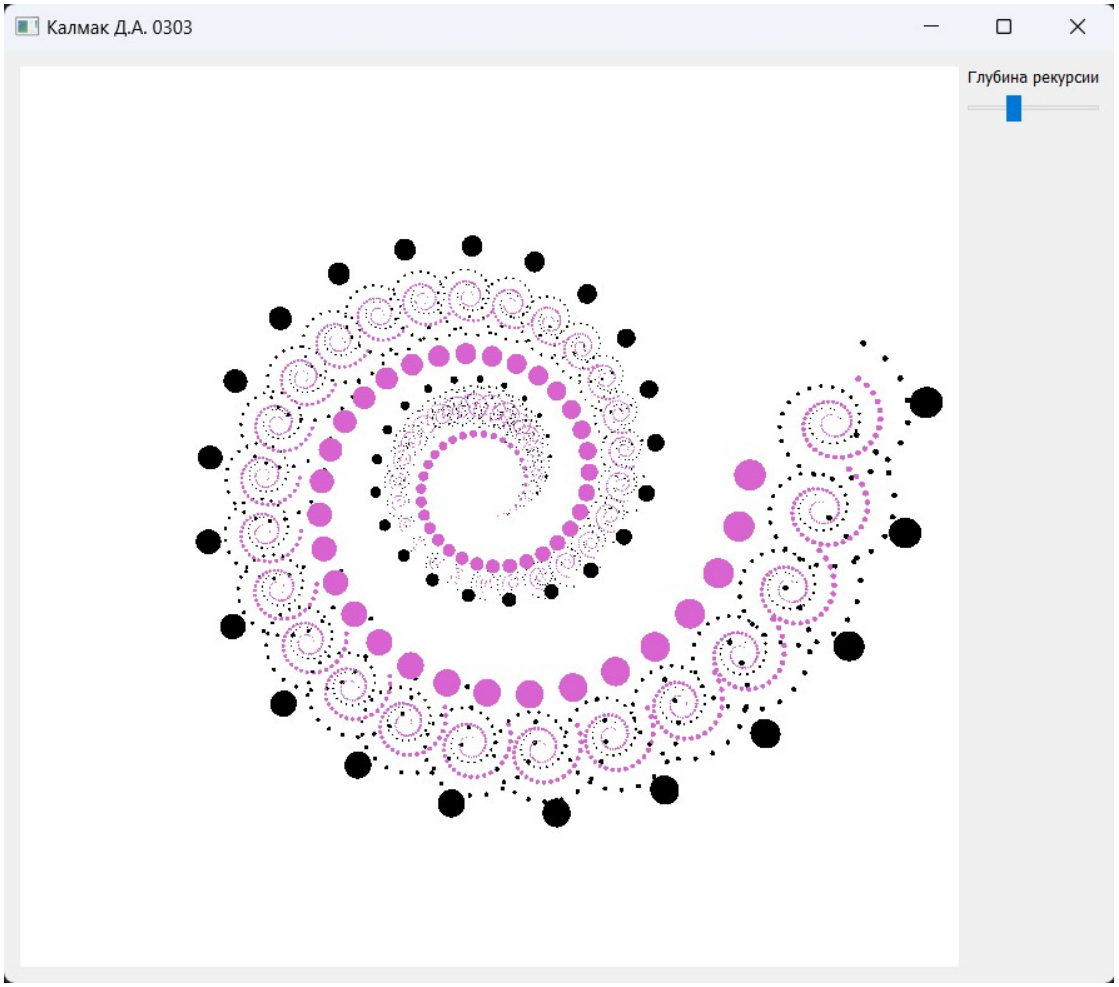


Рисунок 3 – Фрактал с первым уровнем рекурсии

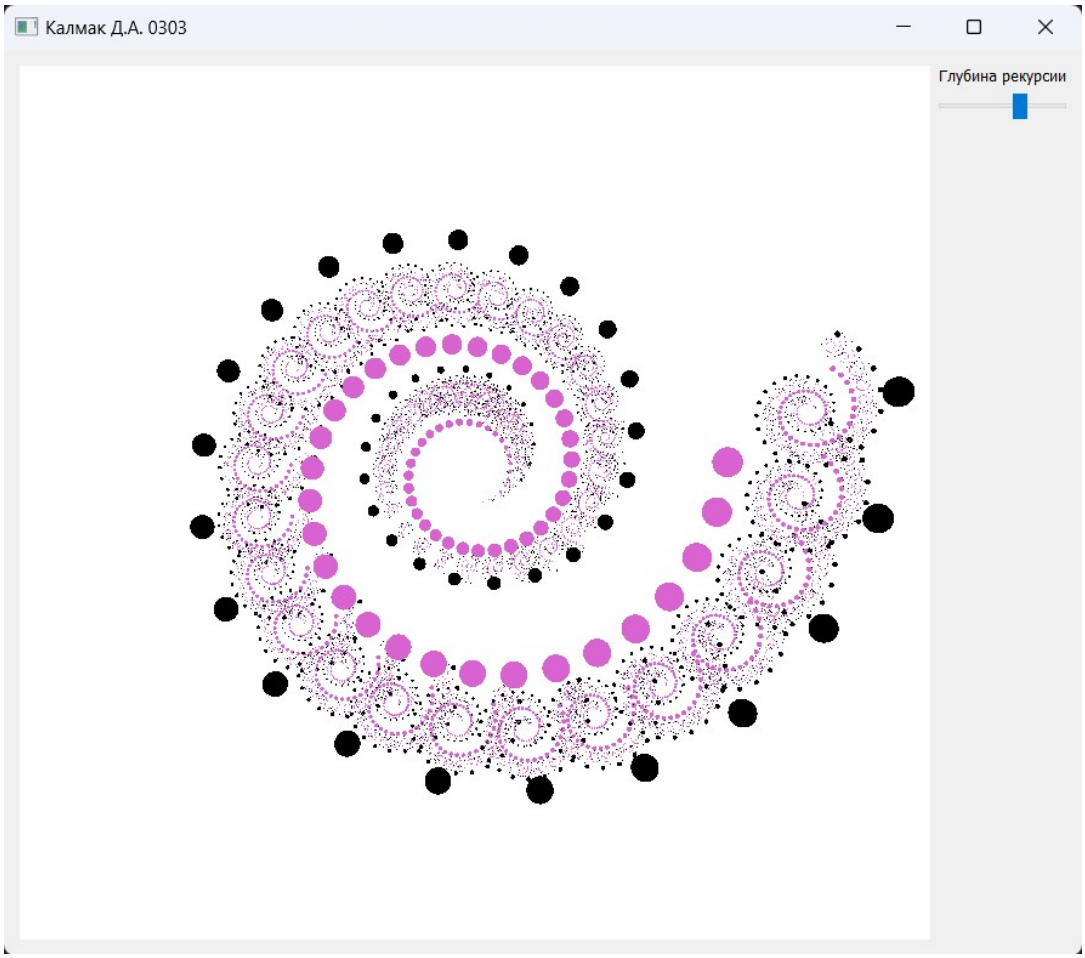


Рисунок 4 – Фрактал со вторым уровнем рекурсии

Вывод.

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

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import math
import sys
from OpenGL.GL import *
from OpenGL.GLU import *
from PyQt5.QtCore import Qt
from PyQt5.QtOpenGL import *
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import (QWidget, QLabel,
                             QComboBox, QStackedWidget, QSlider, QCheckBox)


class mainWindow(QWidget):
    def __init__(self, parent=None):
        super(mainWindow, self).__init__()
        self.stack = QStackedWidget()
        self.stack.addWidget(glWidgetFractal())

        buttonsLayout = QtWidgets.QVBoxLayout()
        self.lblrecursiondepth = QLabel("Глубина рекурсии", self)
        self.sliderrecdepth = QSlider(Qt.Orientation.Horizontal, self)
        self.sliderrecdepth.setMinimum(1)
        self.sliderrecdepth.setMaximum(4)
        self.sliderrecdepth.valueChanged.connect(self.update_recdepth)
        buttonsLayout.addWidget(self.lblrecursiondepth)
        buttonsLayout.addWidget(self.sliderrecdepth)
        buttonsLayout.addStretch()

        mainLayout = QtWidgets.QHBoxLayout()
        widgetLayout = QtWidgets.QHBoxLayout()
        widgetLayout.addWidget(self.stack)
        mainLayout.addLayout(widgetLayout)
        mainLayout.addLayout(buttonsLayout)
        self.setLayout(mainLayout)
        self.setWindowTitle("Калмак Д.А. 0303")

    def update_recdepth(self, value):
        for i in range(self.stack.__len__()):
            self.stack.widget(i).level = value
            self.stack.widget(i).updateGL()


class glWidget0(QGLWidget):
    def __init__(self, parent=None):
        QGLWidget.__init__(self, parent)
        self.setMinimumSize(750, 720)
        self.w = 480
        self.h = 480
        self.level = 1
        self.count = 0

    def initializeGL(self):
```

```
        glClearColor(1.0, 1.0, 1.0, 0.1)
        glClearDepth(1.0)
        glDepthFunc(GL_LESS)
        glEnable(GL_DEPTH_TEST)
        glShadeModel(GL_SMOOTH)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        gluPerspective(45.0, 1, 0.1, 100.0)
        glMatrixMode(GL_MODELVIEW)

    def paintGL(self):
        pass

    def resizeGL(self, w, h):
        self.w = w
        self.h = h
        glViewport(0, 0, w, h)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        aspect = w / h
        gluPerspective(45.0, aspect, 0.1, 100)
        glMatrixMode(GL_MODELVIEW)


class glWidgetFractal(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(0, 0, -4.0)
        self.draw_spiral(0, 1, 1, [])
        self.count = 0

    def draw_spiral(self, level, scalex, scaley, xascop):
        if level == self.level:
            self.count += 1
            print(self.count)
            return
        a = 0.01
        b = 0.15
        k = 0
        h = 0.05
        r = 0.001
        count = 0
        countx = 0
        county = 0
        xy1 = []
        while k < 4.5 * math.pi:
            x = a * pow(2.718281, b * k) * math.cos(k)
            y = a * pow(2.718281, b * k) * math.sin(k)
            xy1.append([x + countx, y + county])
            glTranslatef(x, y, 0.0)
            countx += x
            county += y
            glColor4f(0.0, 0.0, 0.0, 0.5)
            quadObj = gluNewQuadric()
            if count % 6 == 0:
                gluSphere(quadObj, r, 50, 50)
```

```
            k += h
            r += 0.0002
            count += 1
            glFlush()
        glTranslatef(-countx, -county, 0)
        a2 = 0.006
        b2 = 0.15
        k = 0
        h = 0.05
        r = 0.001
        count = 0
        countx = 0
        county = 0
        xy2 = []
        while k < 4.5 * math.pi:
            x = a2 * pow(2.718281, b2 * k) * math.cos(k)
            y = a2 * pow(2.718281, b2 * k) * math.sin(k)
            xy2.append([x + countx, y + county])
            glTranslatef(x, y, 0.0)
            countx += x
            county += y
            glColor4f(0.85098, 0.38823, 0.81568, 0.5)
            quadObj = gluNewQuadric()
            if count % 4 == 0:
                gluSphere(quadObj, r, 50, 50)
            k += h
            r += 0.0002
            count += 1
            glFlush()
        glLoadIdentity()
        glTranslatef(0, 0, 0)
        xas = [[(xy1[i][0] + xy2[i][0]) / 2, (xy1[i][1] + xy2[i][1]) / 2] for i in
range(len(xy1) - 1, -1, -1)]
        if level > 0:
            xas = xascop
        count_rotate = 0
        x_scale = scalex * 0.2
        y_scale = scaley * 0.2
        check = 1
        check_x = 0
        for i in range(0, len(xas), 5):
            glTranslatef(xas[i][0], xas[i][1], -4)
            glRotate(-60+count_rotate, xas[i][0], xas[i][1], -4)
            xascopy = [0 for j in range(0, len(xas))]
            for k in range(0, len(xas)):
                x_new = xas[i][0] + xas[k][0] * x_scale
                y_new = xas[i][1] + xas[k][1] * y_scale
                xascopy[k] = [(x_new - xas[i][0]) * math.cos((60-count_rotate) *
math.pi / 180) - (y_new - xas[i][1]) * math.sin((60-count_rotate) * math.pi / 180) +
xas[i][0],
                              (x_new - xas[i][0]) * math.sin((60-count_rotate) *
math.pi / 180) + (y_new - xas[i][1]) * math.cos((60-count_rotate) * math.pi / 180) +
xas[i][1]]
            glScale(x_scale, y_scale, 0)
            self.draw_spiral(level + 1, x_scale, y_scale, xascopy)
            glLoadIdentity()
            count_rotate += 5
```

```
            if check == 1:
                x_scale -= 0.005 * scalex
                y_scale -= 0.005 * scaley
            if check == 0:
                x_scale += 0.005 * scalex
                y_scale += 0.005 * scaley
            elif check == -1:
                x_scale -= 0.007 * scalex
                y_scale -= 0.007 * scaley
            if x_scale < 0.025 * scalex and check_x == 0:
                check = 0
                check_x = 1
            if x_scale > 0.07 * scalex and check_x == 1:
                check = -1
                check_x = 2


if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    qWindow = QtWidgets.QMainWindow()
    window = mainWindow(qWindow)
    window.show()
    sys.exit(app.exec_())
```