**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторным работам №4**
**по дисциплине «Компьютерная графика»**
**Тема: Построение фракталов**

Студентка гр. 1304 _____ Чернякова В.А.

Студентка гр. 1304 _____ Ярусова Т.В.

Преподаватель _____ Герасимова Т.В.

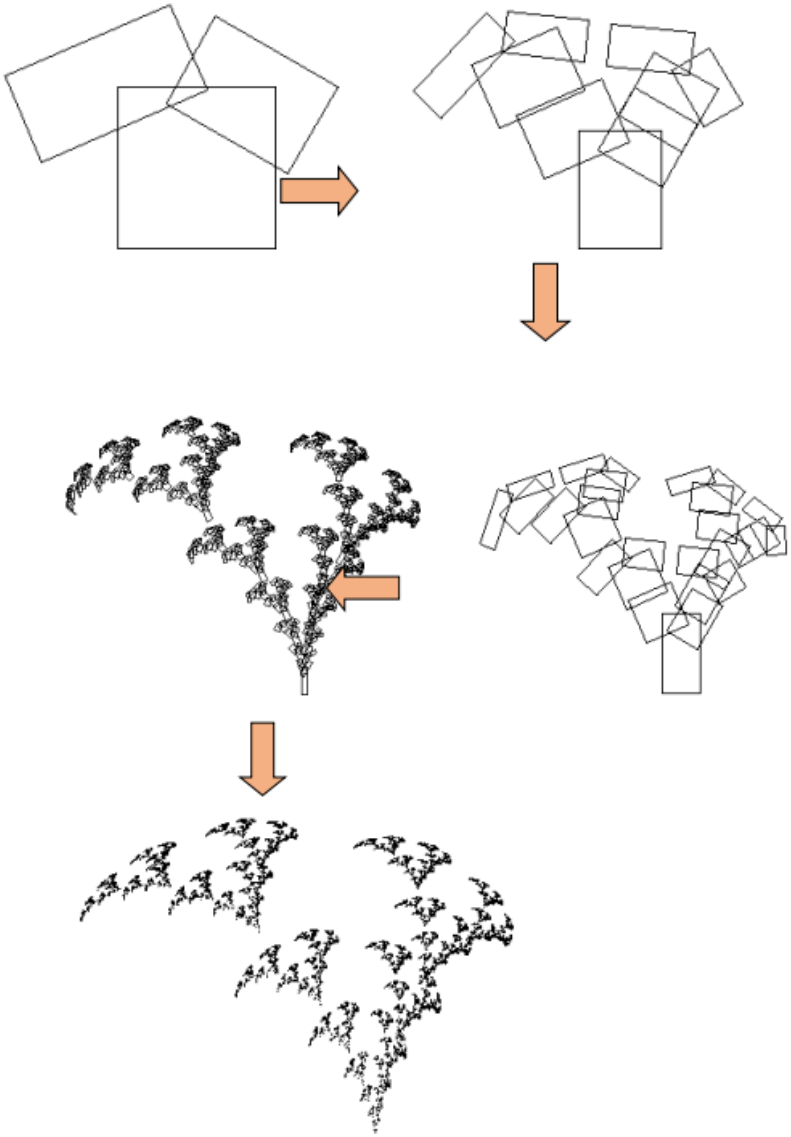
Санкт-Петербург
2024

**Цель работы.**

Разработать программу, реализующую представление заданного фрактала.

**Задание.**

Задание 13. Геометрический фрактал:



**Теоретические положения.**

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.

Геометрические фракталы, основанные на прямоугольниках, представляют собой класс фракталов, которые строятся путем разбиения прямоугольников на более мелкие части с определенными пропорциями и правилами. Одним из известных примеров таких фракталов является дерево Пифагора.

В основе геометрических фракталов, основанных на прямоугольниках, лежит принцип самоподобия - структура фрактала повторяется на различных масштабах и состоит из подобных элементов. При этом каждый следующий шаг построения фрактала определяется определенными правилами, такими как соотношение размеров частей, углы поворота и другие параметры.

Система итерирующих функций (Iterated Function System, IFS) — это математическая конструкция, используемая для описания фрактальных объектов и генерации сложных форм на основе простых правил. IFS состоит из набора функций (обычно конечного), каждая из которых преобразует точки в пространстве (например, двумерное пространство) согласно определенным правилам. Эти функции могут быть применены итеративно, то есть результат применения каждой функции становится входом для следующей функции.

Ключевая идея заключается в том, что при достаточном числе итераций эти функции порождают фрактальные структуры, такие как самоподобные образы, которые могут быть бесконечно детализированы. Структуры, порождаемые IFS, часто обладают высокой степенью симметрии и регулярности, но при этом могут быть достаточно сложными и разнообразными.

Аффинные преобразования — это взаимно однозначные отображения плоскости (пространства) на себя, при которых прямые переходят в прямые.

Примеры аффинных преобразований:

- ортогональное преобразование (представляет собой движение плоскости или пространства или движение с зеркальным отражением);

- преобразования подобия;

- равномерное сжатие.

**Выполнение работы.**

Рассмотрим реализацию построения геометрического фрактала на языке программирования *Python* с применением библиотек *PyQt5* и *OpenGL*.

В файле *MainWindow.py* создан одноименный класс, который наследуется от класса *QMainWindow* из *PyQt5* и содержит методы для инициализации, установки разметки элементов интерфейса и обработки сигналов.

В методе *init* происходит создание виджетов (виджет для отображения графики *GLScene* и виджета для выбора количества итераций *QSpinBox*), настройка их отображения и размещение на главном окне.

Метод *iterationChanged*, который обрабатывает изменение значения спинбокса. Этот метод вызывается при изменении значения в *QSpinBox* и обновляет соответствующий параметр класса *GLScene*.

Метод *connectSignals*, который связывает сигналы и слоты. Этот метод устанавливает соединение между сигналом изменения значения *QSpinBox* и слотом *iterationChanged*.

Таким образом, данный код создает главное окно приложения с возможностью выбора количества итераций через спинбокс и отображения графики на главном экране.

Подключение графической библиотеки осуществляется с помощью виджета *QOpenGLWidget.*

Создан виджет *GLSence* наследуемый от данного класса, в котором с помощью переопределённых методов *initializeGL, resizeGL* и *paintGL*

осуществляется соответственно подготовка кадра, переопределение размеров и отрисовка изображения.

Метод *init* класса *GLSence* инициализирует параметры класса, такие как размеры окна, угол поворота, количество итераций и списки прямоугольников, создает начальный прямоугольник и сохраняет его в списке *old_rectangles*.

```
def __init__(self, parent=None):
    super(GLScene, self).__init__(parent)
    self.frameHeight = None
    self.frameWidth = None
    self.width = 640
    self.height = 640

    self.angle = 0
    self.iterations = 0
    self.old_rectangles = [
        createRectangle(self.width / 2, self.height / 2, self.angle,
ZERO_LEVEL, ZERO_LEVEL)
    ]
    self.new_rectangles = []
    self.steps = {0: self.old_rectangles}
```

Метод *initializeGL* инициализирует *OpenGL*, устанавливает цвет очистки.

Метод *resizeGL* вызывается при изменении размеров окна и устанавливает соответствующие настройки *OpenGL*.

Метод *paintGL* вызывается для отрисовки сцены в *OpenGL*, очищает буферы цвета и глубины. Вызывает функцию *fractalGenerator(self)* для создания фрактала. Далее происходит отрисовка каждого прямоугольника из списка *old_rectangles* с помощью функции *paintRectangle(rectangle)*.

```
def paintGL(self):
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

    fractalGenerator(self)
    for rectangle in self.old_rectangles:
        paintRectangle(rectangle)
```

В файле *FractalGenerator* содержится основная функция *fractalGenerator*, которая генерирует фракталы путем деления прямоугольников на более мелкие фрагменты.

В начале функции проверяется, если количество итераций *(gl.iterations)* уже существует в словаре *gl.steps*. Если да, то текущий список прямоугольников сохраняется в *gl.old_rectangles*, а *gl.new_rectangles* очищается. Если количество итераций еще нет в словаре *gl.steps*, то происходит цикл по каждому прямоугольнику из списка *gl.old_rectangles*: для каждого прямоугольника

вычисляются новые координаты и размеры для трех новых прямоугольников с помощью следующих формул аффинных преобразований:

- Для первого нового прямоугольника:

Смещение по x: x – w + 2 * w * 72 / 240

Смещение по y: y– h + 2 * h * (1 - 55 / 176)

Поворот на угол: angle + np.pi / 12

Коэффициент масштабирования по высоте: h * 67 / 176

Коэффициент масштабирования по ширине: w * 129 / 240

- Для второго нового прямоугольника:

Смещение по x: x - w + 2 * w * 177 / 240

Смещение по y: y - h + 2 * h * (1 - 65 / 176)

Поворот на угол: angle - np.pi / 8

Коэффициент масштабирования по высоте: h * 71 / 176

Коэффициент масштабирования по ширине: w * 102 / 240

- Для третьего нового прямоугольника:

Смещение по x: x - w + 2 * w * 117 / 240

Смещение по y: y - h + 2 * h * (1 - 120 / 176)

Поворот на угол: angle

Коэффициент масштабирования по высоте: h * 117 / 176

Коэффициент масштабирования по ширине: w * 102 / 240

```
def fractalGenerator(gl):
    if gl.iterations in gl.steps:
        gl.old_rectangles = gl.steps[gl.iterations]
        gl.new_rectangles = []
        return

    for rectangle in gl.old_rectangles:
        x = rectangle['x']
        y = rectangle['y']
        h = rectangle['h']
        w = rectangle['w']
        angle = rectangle['angle']

        gl.new_rectangles.append(createRectangle(
            x - w + 2 * w * 72 / 240,
            y - h + 2 * h * (1 - 55 / 176),
            angle + np.pi / 12,
            h * 67 / 176,
            w * 129 / 240
        ))
```

```
        gl.new_rectangles.append(createRectangle(
            x - w + 2 * w * 177 / 240,
            y - h + 2 * h * (1 - 65 / 176),
            angle - np.pi / 8,
            h * 71 / 176,
            w * 102 / 240
        ))

        gl.new_rectangles.append(createRectangle(
            x - w + 2 * w * 117 / 240,
            y - h + 2 * h * (1 - 120 / 176),
            angle,
            h * 117 / 176,
            w * 102 / 240
        ))

    gl.old_rectangles = copy.deepcopy(gl.new_rectangles)
    gl.steps[gl.iterations] = copy.deepcopy(gl.new_rectangles)
    gl.new_rectangles = []
```

После создания новых прямоугольников, список *gl.old_rectangles* обновляется до нового списка прямоугольников *(gl.new_rectangles*). Также текущий список прямоугольников добавляется в словарь *gl.steps* по ключу количества итераций.

Наконец, *gl.new_rectangles* очищается для следующей итерации.

В файле *Rectangle.py* функция *createRectangle* создает прямоугольник с определенными параметрами: координатами центра (x, y), углом поворота angle, высотой h и шириной w. Функция возвращает словарь с этими параметрами.

Функция *paintRectangle* используется для отрисовки прямоугольника, заданного в виде словаря. Внутри функции устанавливается цвет черный, и с помощью *OpenGL* рисуется линейный контур прямоугольника. Координаты вершин прямоугольника вычисляются с учетом угла поворота с помощью тригонометрических функций *cos* и *sin*.

```
def createRectangle(x, y, angle, h, w):
    return {'x': x, 'y': y, 'h': h, 'w': w, 'angle': angle}


def paintRectangle(rectangle):
    x = rectangle['x']
    y = rectangle['y']
    w = rectangle['w']
    h = rectangle['h']
    angle = rectangle['angle']

    glColor3f(0.0, 0.0, 0.0)
    glBegin(GL_LINE_LOOP)
    glVertex2f(x + w * cos(angle) - h * sin(angle), y + w * sin(angle) + h *
```

```
cos(angle))
    glVertex2f(x + w * cos(angle) + h * sin(angle), y + w * sin(angle) - h *
cos(angle))
    glVertex2f(x - w * cos(angle) + h * sin(angle), y - w * sin(angle) - h *
cos(angle))
    glVertex2f(x - w * cos(angle) - h * sin(angle), y - w * sin(angle) + h *
cos(angle))
    glEnd()
```

**Тестирование.**

Рассмотрим геометрический фрактал на разных итерациях:

- Итерация 0. (см. рис. 1)

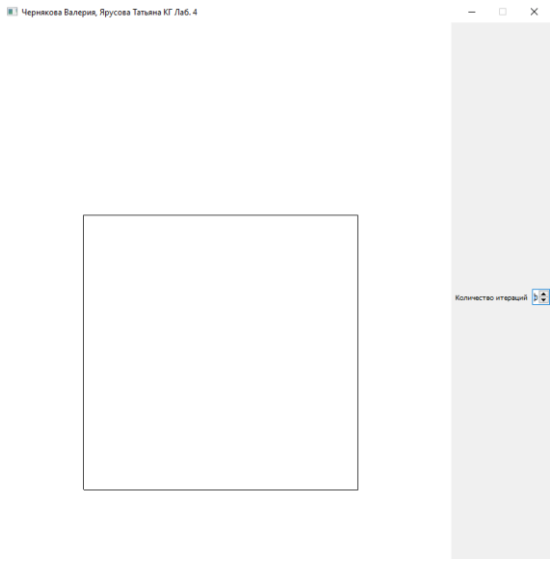



Рисунок 1. Геометрический фрактал на 0 итерации

- Итерация 1. (см. рис. 2)

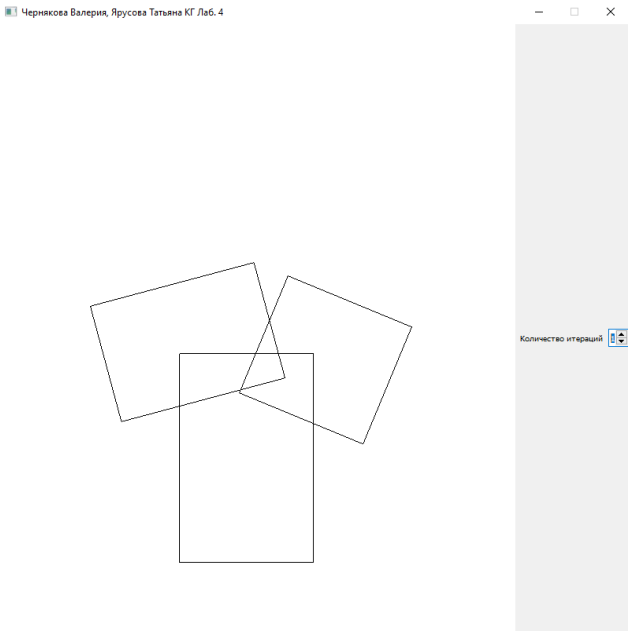



Рисунок 2. Геометрический фрактал на 1 итерации

- Итерация 3. (см. рис. 3)

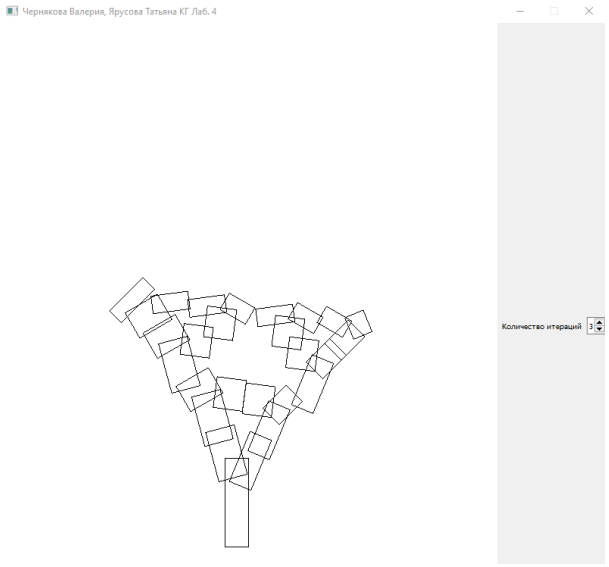



Рисунок 3. Геометрический фрактал на 3 итерации

- Итерация 5. (см. рис. 4)

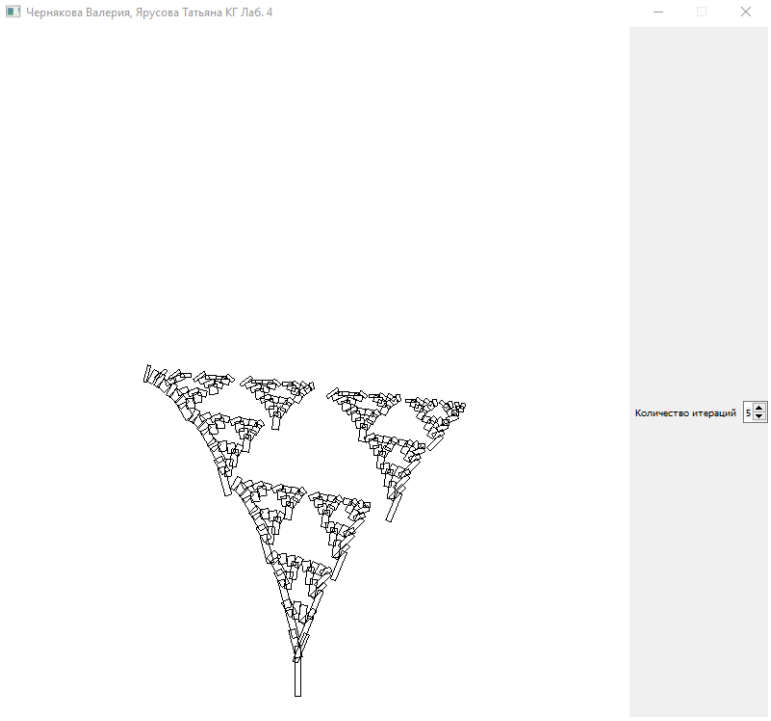



Рисунок 4. Геометрический фрактал на 5 итерации

**Выводы.**

В ходе лабораторной работы были изучены принципы построения геометрического фрактала. Была реализована программа, которая строит заданный фрактал в зависимости от выбранной итерации.

Разработана программа для визуализации геометрического фрактала, которая позволяет наглядно представить этот фрактал и изучить его особенности. В дальнейшем можно будет экспериментировать с различными параметрами, такими как углы поворота и соотношения сторон, что позволит получить разнообразные и красивые варианты дерева.