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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Компьютерная графика»
Тема: Построение фракталов

Студент гр. 0382 Корсунов А.А.

Преподаватель Герасимова Т.В.

Санкт-Петербург
2023

Цель работы.

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

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.

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.

2. *Алгебраические фракталы*

Это самая крупная группа фракталов. Получают их с помощью нелинейных процессов в n-мерных пространствах. Наиболее изучены двухмерные процессы.

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.

Задание.

Разработать программу реализующую фрактал по заданному изображению:

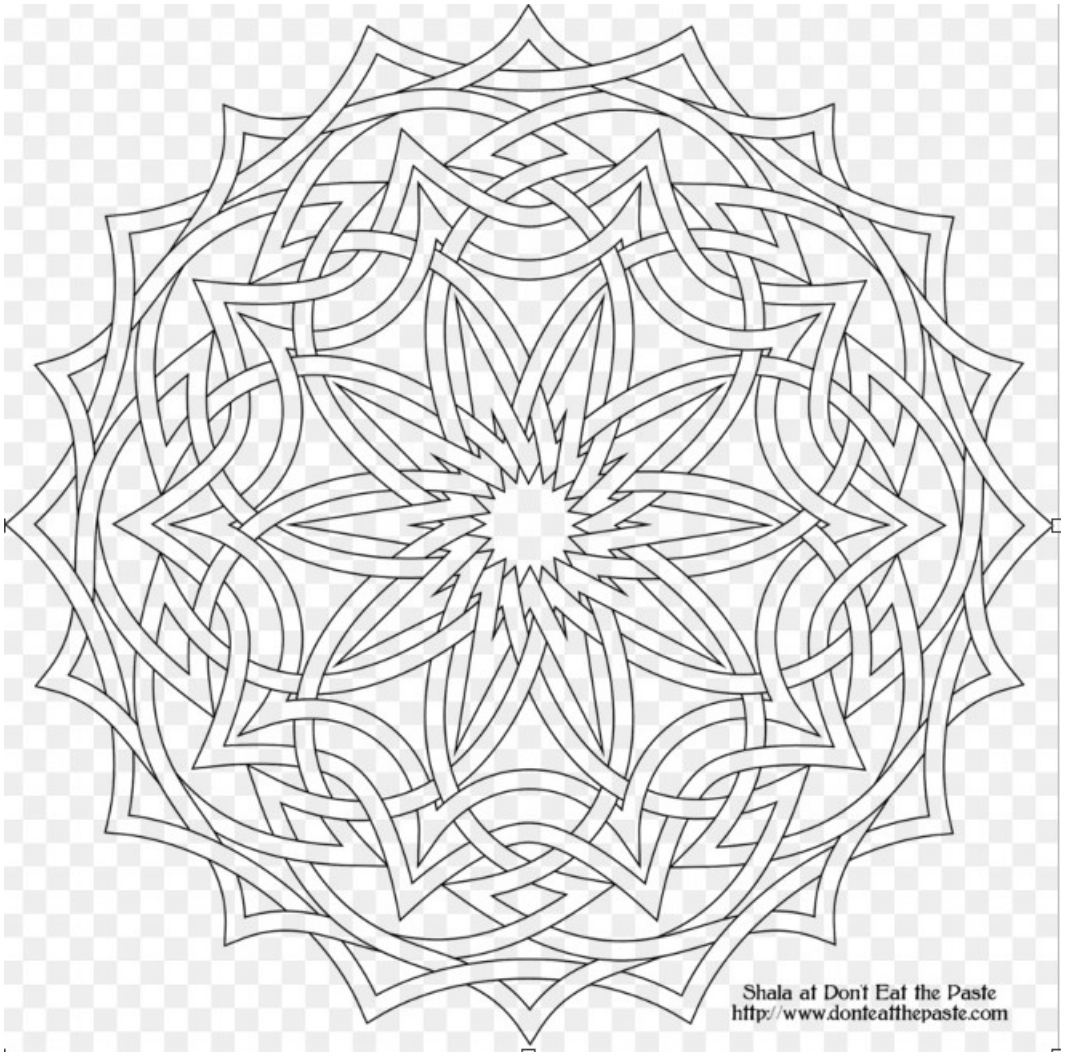

Рисунок 1 — Фрактал для реализации

Ход работы.

Код данной лабораторной работы основан на коде лабораторной работы №2, фрактал главным образом реализуется в новой рекурсивной функции *draw_fractal_main_part.*

1. Отображение окружности

Фрактал (рис.1) изначально можно представить в виде окружности, а в дальнейшем на ее основе вычислять положения изгибов фрактала.

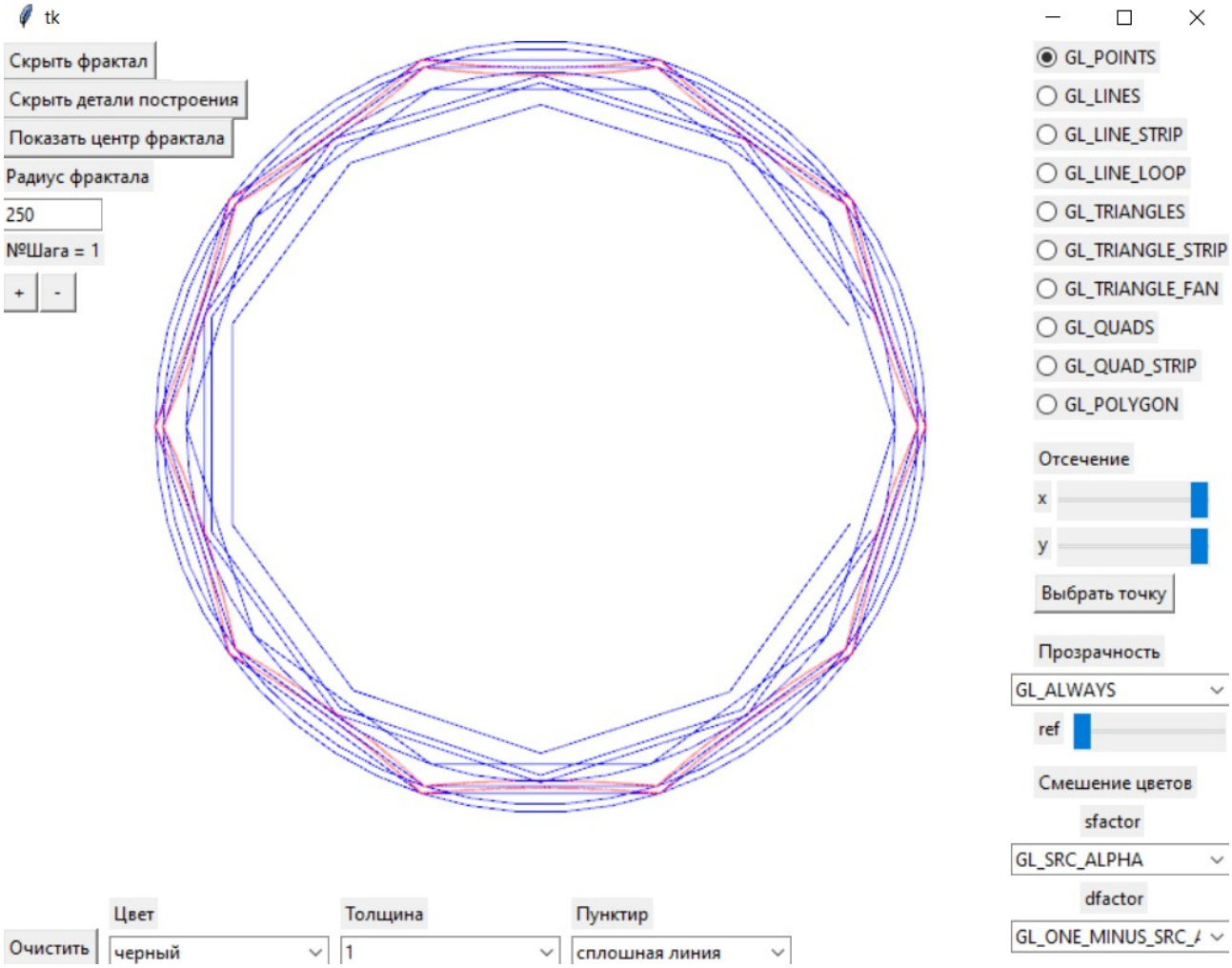

Рисунок 2 — Окружность

Часть функции *draw_fractal_main_part:*

```
...
for i in range(100):
    if turn_show:
        GL.glVertex2f(R * math.cos(2 * math.pi / 50 * i) + 350, R * math.sin(2 * math.pi / 50 * i) + 250)
    if i % 5 == 0:
        angles_points.append((R * math.cos(2 * math.pi / 50 * i) + 350, R * math.sin(2 * math.pi / 50
* i) + 250))

for i in range(10):
    temp_y = (angles_points[i][1] + angles_points[i + 1][1]) / 2
    low_angles_points.append((((angles_points[i][0] + angles_points[i + 1][0]) / 2), temp_y))
GL.glEnd()
...
```

В цикле прорисовывается окружность. В список *angles_points.append* заносятся координаты вершин, соответствующие вершинам, представленных на рисунке ниже.

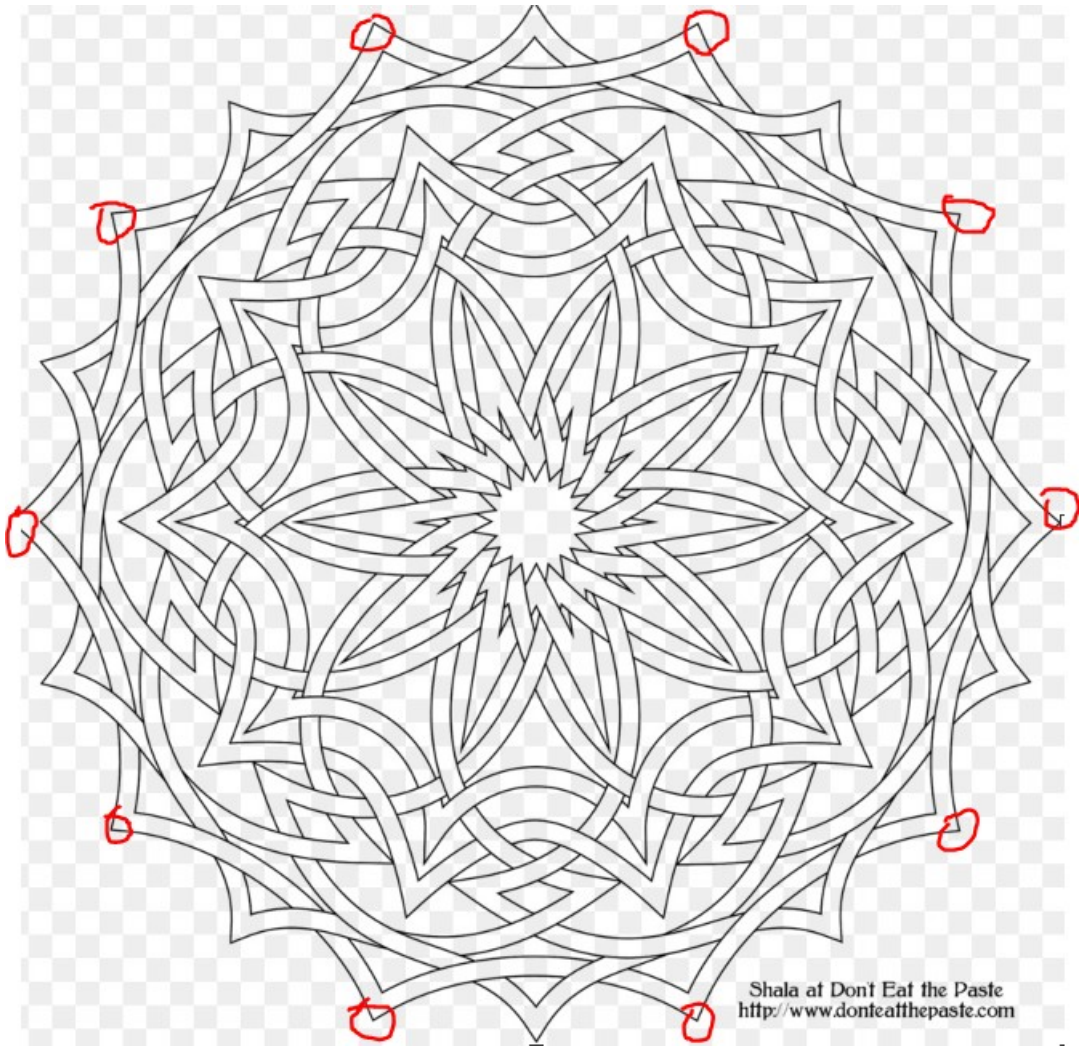

Рисунок 3 — Первые вершины для отображения

После чего в список *low_angles_points.append* заносятся координаты других вершин, на основе предыдущих.

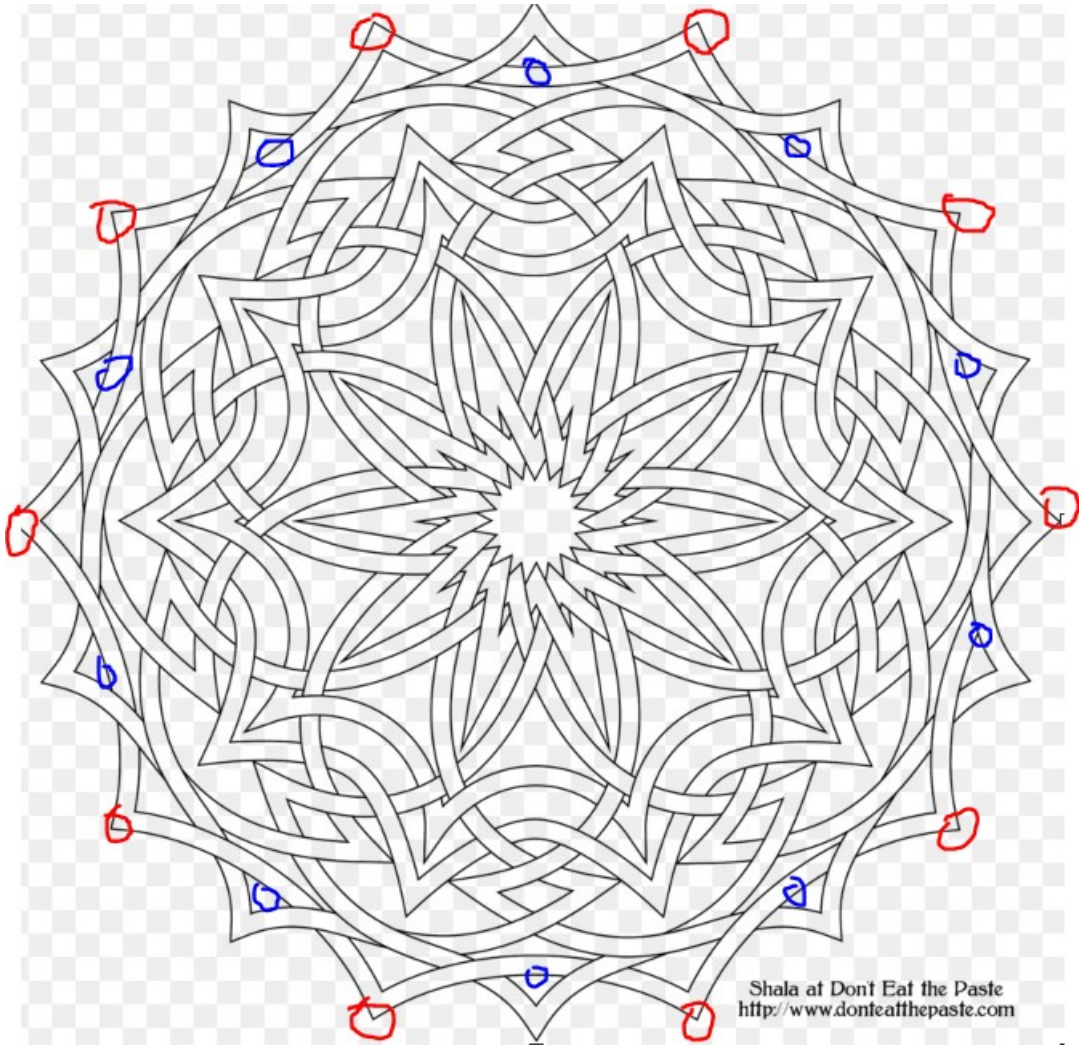

Рисунок 4 — Вторые вершины для отображения

Таким образом можно получить группы по трем точкам, по которым можно построить кривые (изгибы).

2. Отображение кривых.

Для отображения кривых были использованы уравнения Кривых Безье для трех контрольных точек. Уравнения имеют следующий вид:

$$x=(1-t)^2 x_1+2(1-t)tx_2+t^2 x_3$$

$$y=(1-t)^2 y_1+2(1-t)ty_2+t^2 y_3$$

где x_1, y_1, x_2, y_2, x_3, y_3 – контрольные точки, t – переменная, изменяющаяся от 0 до 1

Использовав такие уравнения можно получить следующую фигуру:

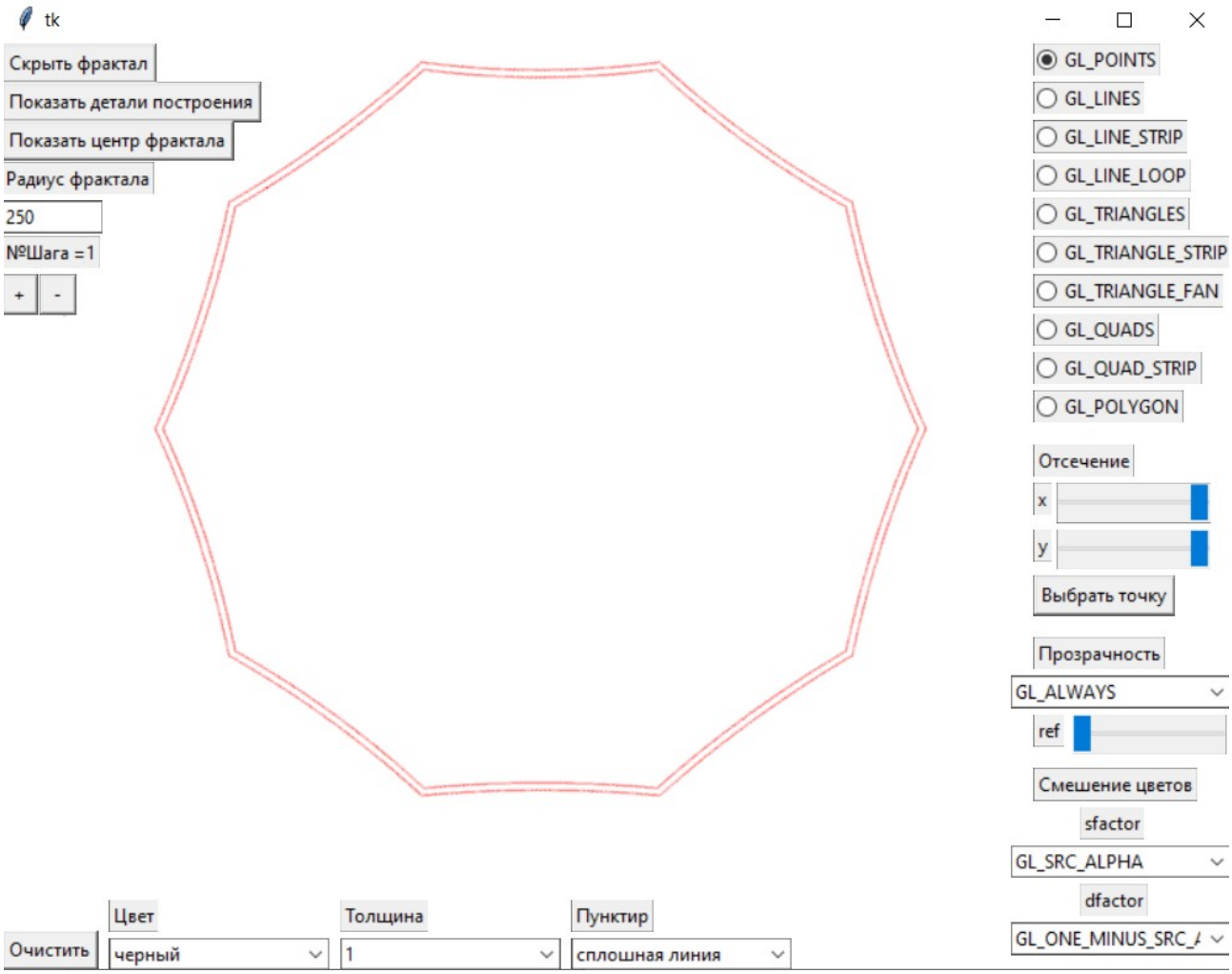

Рисунок 5 — Кривые фигуры

Часть функции *draw_fractal_main_part:*

```
...
for i in range(10):
    control_points = [angles_points[i], low_angles_points[i], angles_points[i + 1]]
    t = 0
```

```
    while t <= 1:
        x = (1 - t) ** 2 * control_points[0][0] + 2 * (1 - t) * t * control_points[1][0] + t ** 2 * \
            control_points[2][0]
        y = (1 - t) ** 2 * control_points[0][1] + 2 * (1 - t) * t * control_points[1][1] + t ** 2 * \
            control_points[2][1]
        t += 0.01
        GL.glVertex3d(x, y, 0)
...
```

В цикле задаются контрольные точки, которые берутся из массивов, созданных в предыдущем пункте, после чего в еще одном цикле прорисовываются вершины, координаты которых изменяются по уравнениям Кривых Безье.

3. Выделение нулевой фазы, на основе которой прорисовывается весь фрактал.

Фигуру, полученную в предыдущем пункте уже можно считать нулевой фазой за исключением некоторых деталей, которые будут описаны позже.

Один из способ получить дальнейший фрактал — сделать функцию *draw_fractal_main_part* рекурсивной и передавать туда уменьшающийся радиус новых окружностей, на основе которых будут строиться новые кривые.

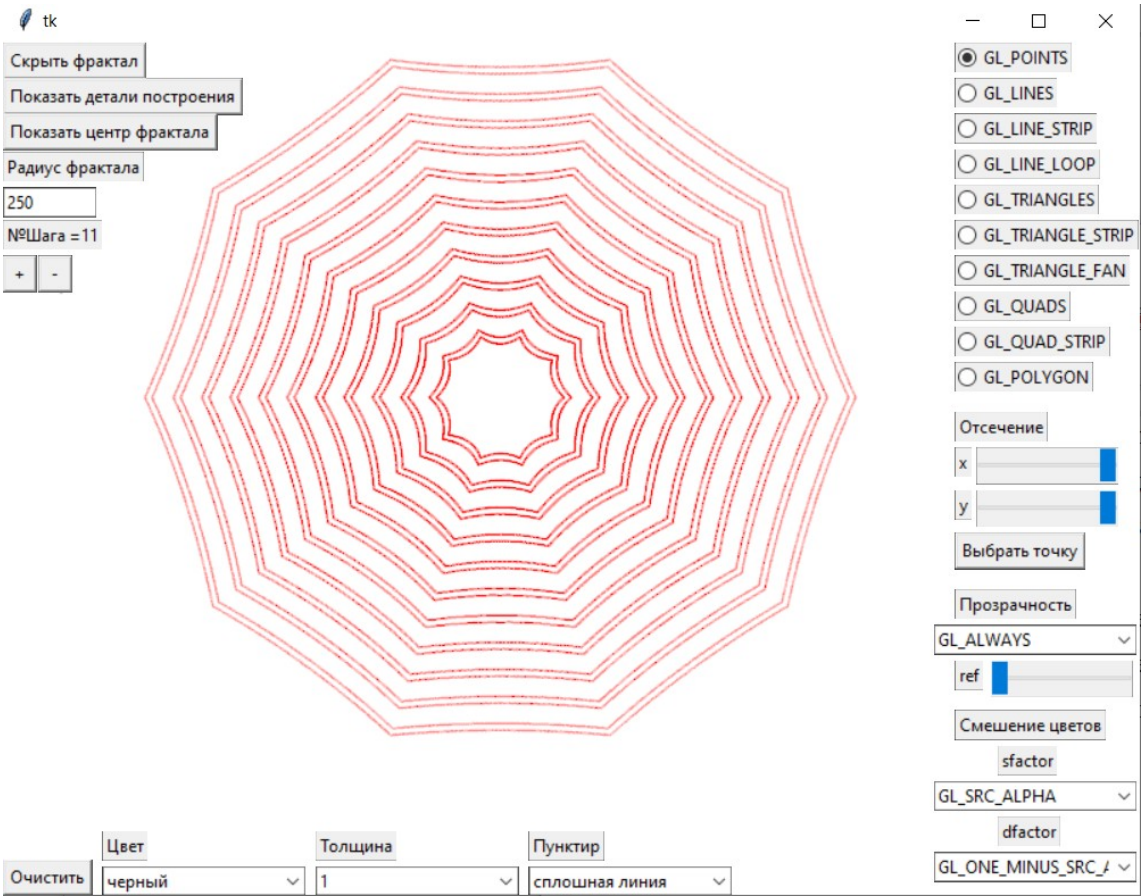

Рисунок 6 — Отображение главной части фрактала

Данная часть соответствует следующей части исходного изображения:

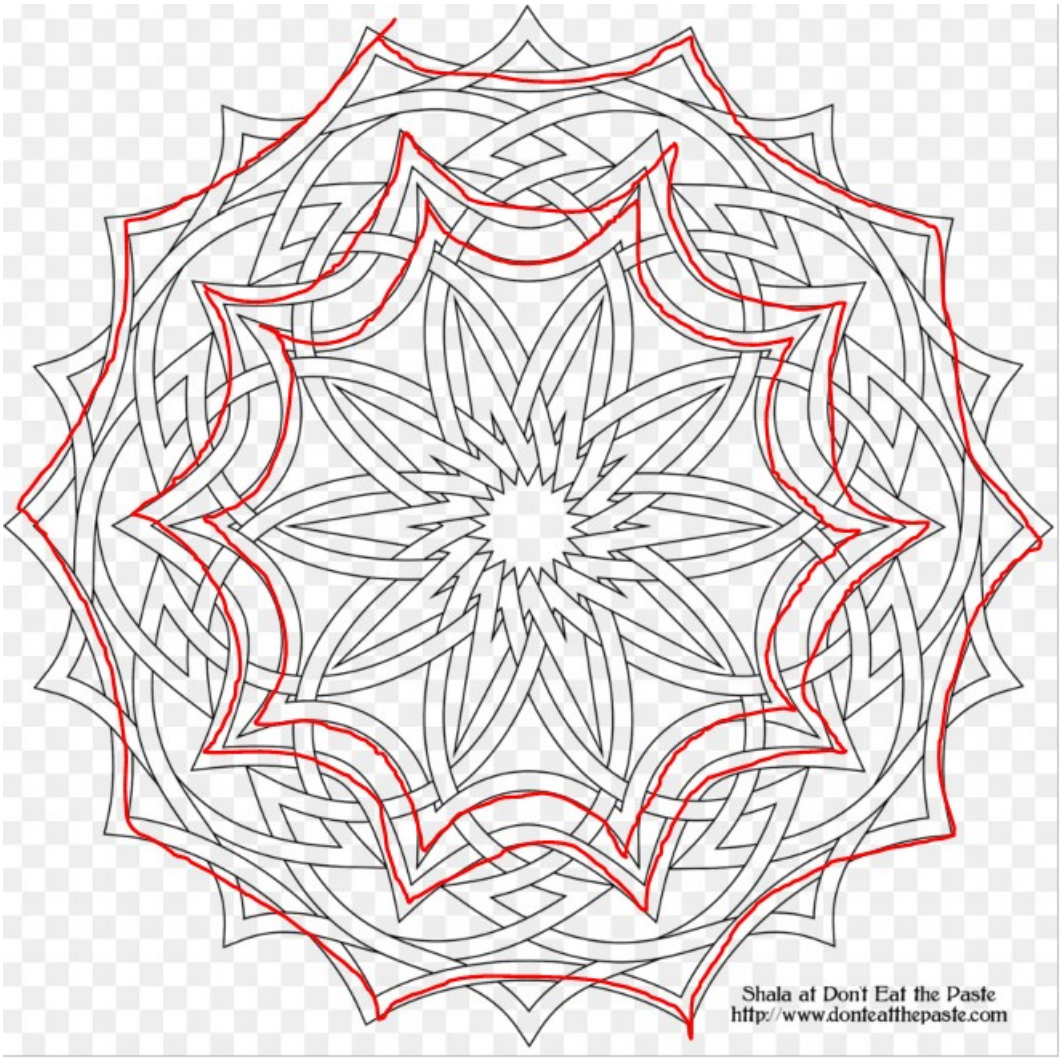

Рисунок 7 — Главная часть на исходном изображении

Данная часть безусловно образует фрактал, потому как подобные друг другу кривые можно отображать бесконечное количество раз, т. к. радиус исходной окружности, по которой строится этот фрактал является вещественным числом.

Часть функции *draw_fractal_main_part:*

...

```
if double and count > 0:
    angles_points.clear()
    low_angles_points.clear()
```

```
    draw_fractal_main_part(R - 5, False, -1, turn_show, add_part_1)

if count > 0 and R > 0:
    angles_points.clear()
    low_angles_points.clear()
    draw_fractal_main_part(R - 20, True, count - 1, turn_show, add_part_1)
...
```

Функция *draw_fractal_main_part* вызывает саму себя в двух случаях:

1) Уже созданной кривой необходимо дорисовать кривую с немного меньшим радиусом, вместе они образуют «толстую кривую»;

2) Заданный пользователем шаг итерации изменяется от n до 0, если в текущем вызове функции шаг итерации не нуль, то необходимо дорисовать кривую с уже более меньшим радиусом;

4. Дополнительная часть фрактала.

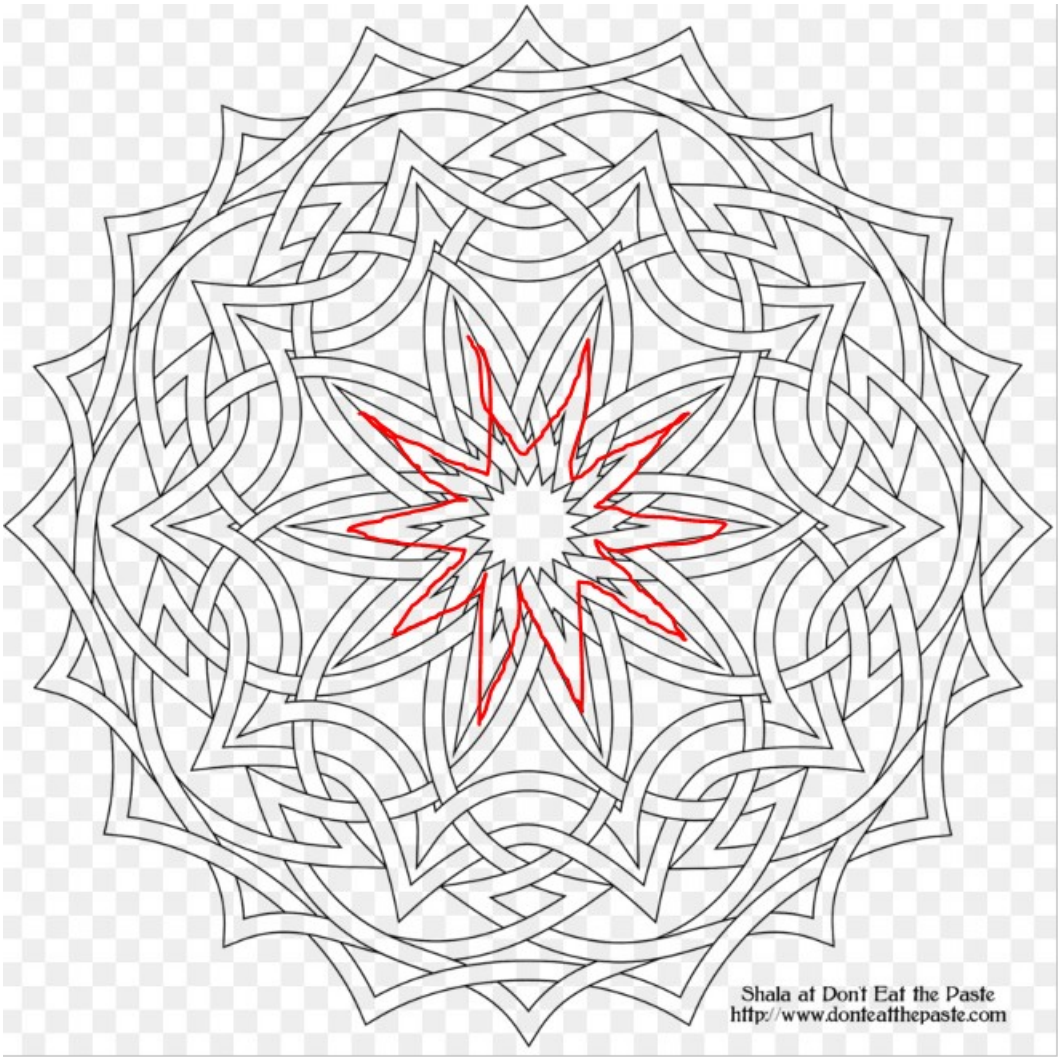

Рисунок 8 — Дополнительная часть фрактала на исходном изображении

Данная часть фрактала на исходном рисунке повторяется лишь один раз в самом центре фрактала, что противоречит свойству самоподобия. Однако, в силу того, что эта часть реализуется проще, чем часть с кривыми, она также была добавлена в программу как дополнительная возможность для пользователя.

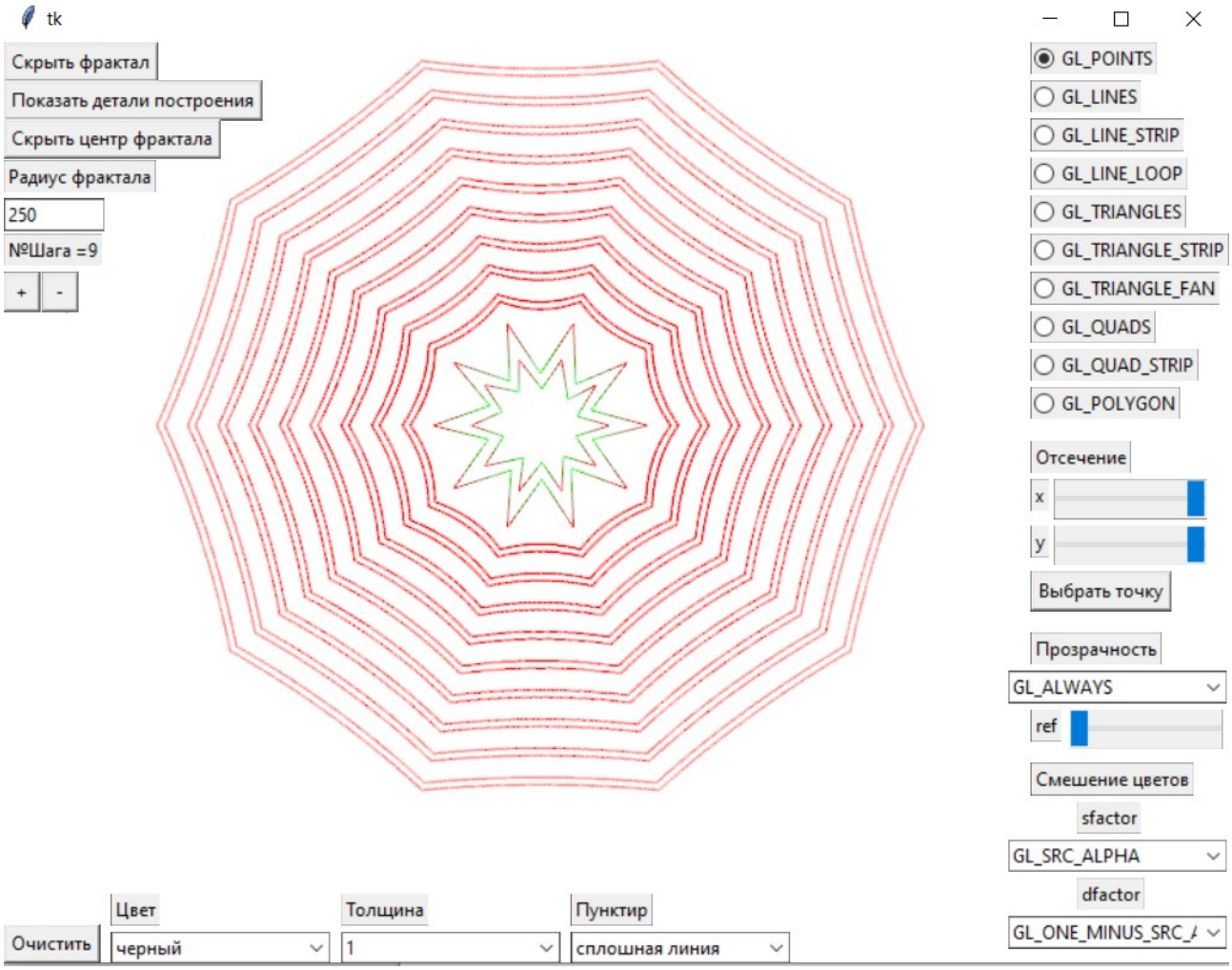

Рисунок 9 — Дополнительная возможность при отображении фрактала

Часть функции *draw_fractal_main_part:*

```
...
for i in range(100):
    if turn_show:
        GL.glVertex2f(45 * math.cos(2 * math.pi / 50 * i) + 350, 45 * math.sin(2 * math.pi / 50 * i) + 250)
    if i % 5 == 0:
```

```
    angles_points.append(
      (45 * math.cos(2 * math.pi / 50 * i) + 350, 45 * math.sin(2 * math.pi / 50 * i) + 250))

GL.glBegin(GL.GL_LINE_LOOP)
for i in range(len(low_angles_points)):
  GL.glColor3d(255, 0, 0)
  GL.glVertex3d(angles_points[i][0], angles_points[i][1], 0)
  GL.glColor3d(0, 255, 0)
  GL.glVertex3d(low_angles_points[i][0], low_angles_points[i][1], 0)
GL.glEnd()

GL.glBegin(GL.GL_LINE_LOOP)

if turn_show:
  for i in range(100):
    GL.glVertex2f(25 * math.cos(2 * math.pi / 50 * i) + 350, 25 * math.sin(2 * math.pi / 50 * i) +
250)

  for i in range(100):
    GL.glVertex2f(35 * math.cos(2 * math.pi / 50 * i) + 350, 35 * math.sin(2 * math.pi / 50 * i) +
250)
GL.glEnd()
...
```

Для отображения этой части были построены три окружности, после чего на основе тех же массивов, что были использованы ранее, были получены координаты точек для отображения прямых линий через примитив GL_LINE_LOOP

5. Последняя часть исходной фигуры.

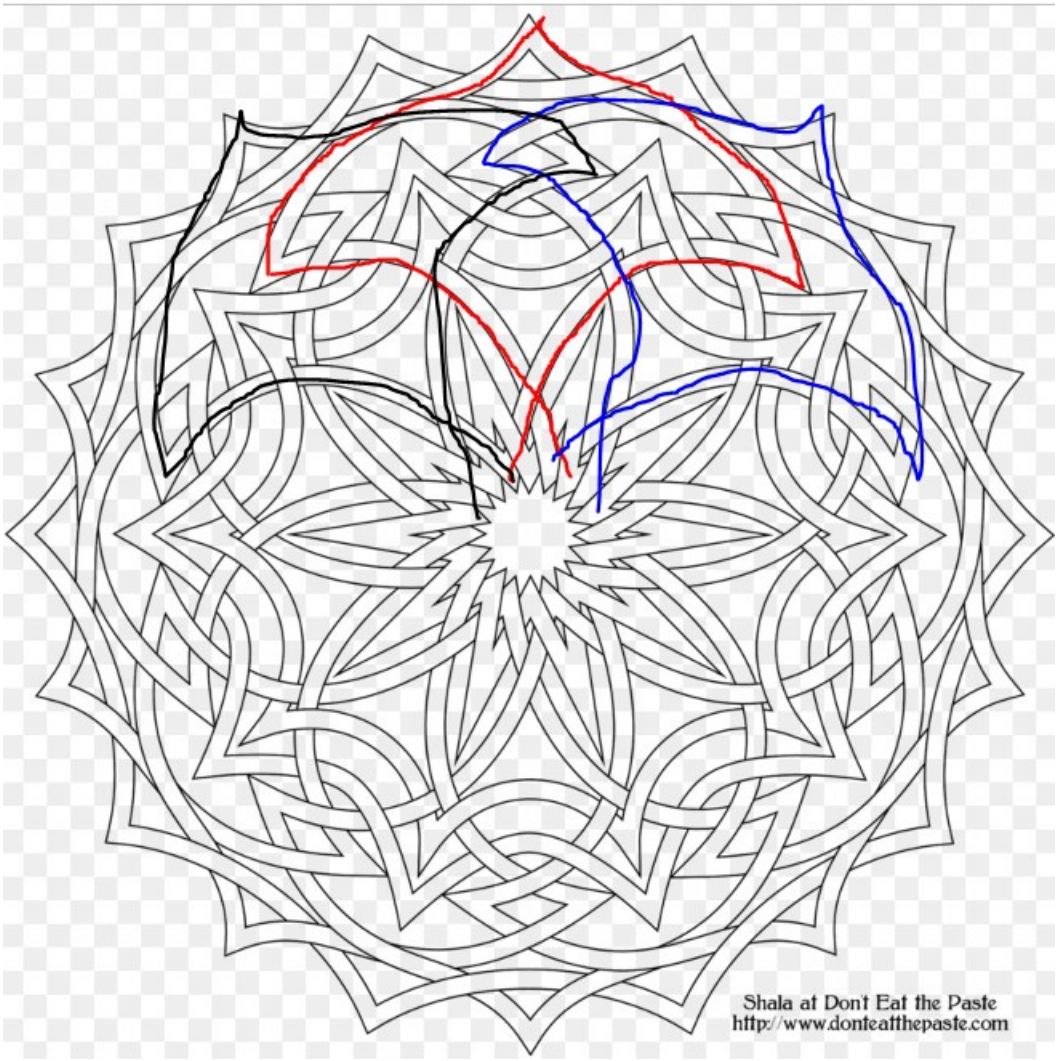

Рисунок 10 — Сложная часть

Последняя часть исходной фигуры также противоречит самоподобию фрактала. Эта часть состоит из «зонтов», которые накладываются друг на друга, располагаясь вокруг центра фрактала. Эта часть состоит из одной непрерывной линии, но очень извилистой, поэтому в силу того, что она не представляет интереса с точки зрения темы фракталов (сколько бы итераций не прошло, эта часть всегда будет с самой первой итерации и единственной), а также из-за сложности реализации (данную фигуру сложно как описать с помощью тех же уравнений Кривых Безье, так и выявить какое-либо другое математическое описание) реализация этой части была отброшена.

6. Пользовательский интерфейс.

Интерфейс программы пополнился управлением отображения фрактала, описанного ранее (рис. 9). Программа предлагает пользователю показать или скрыть фрактал, детали построения (рис. 2), дополнительную часть фрактала, которая была названа его центром. Главным образом пользователь может задавать фрактал через изменение его радиуса (вписав число от 0 до 999 в соответствующее окно (окно предусматривает валидацию), а также через перематывания шагов итерации построения фрактала («+» - на 1 шаг вперед, «-» - на 1 шаг назад).

Примеры работы:

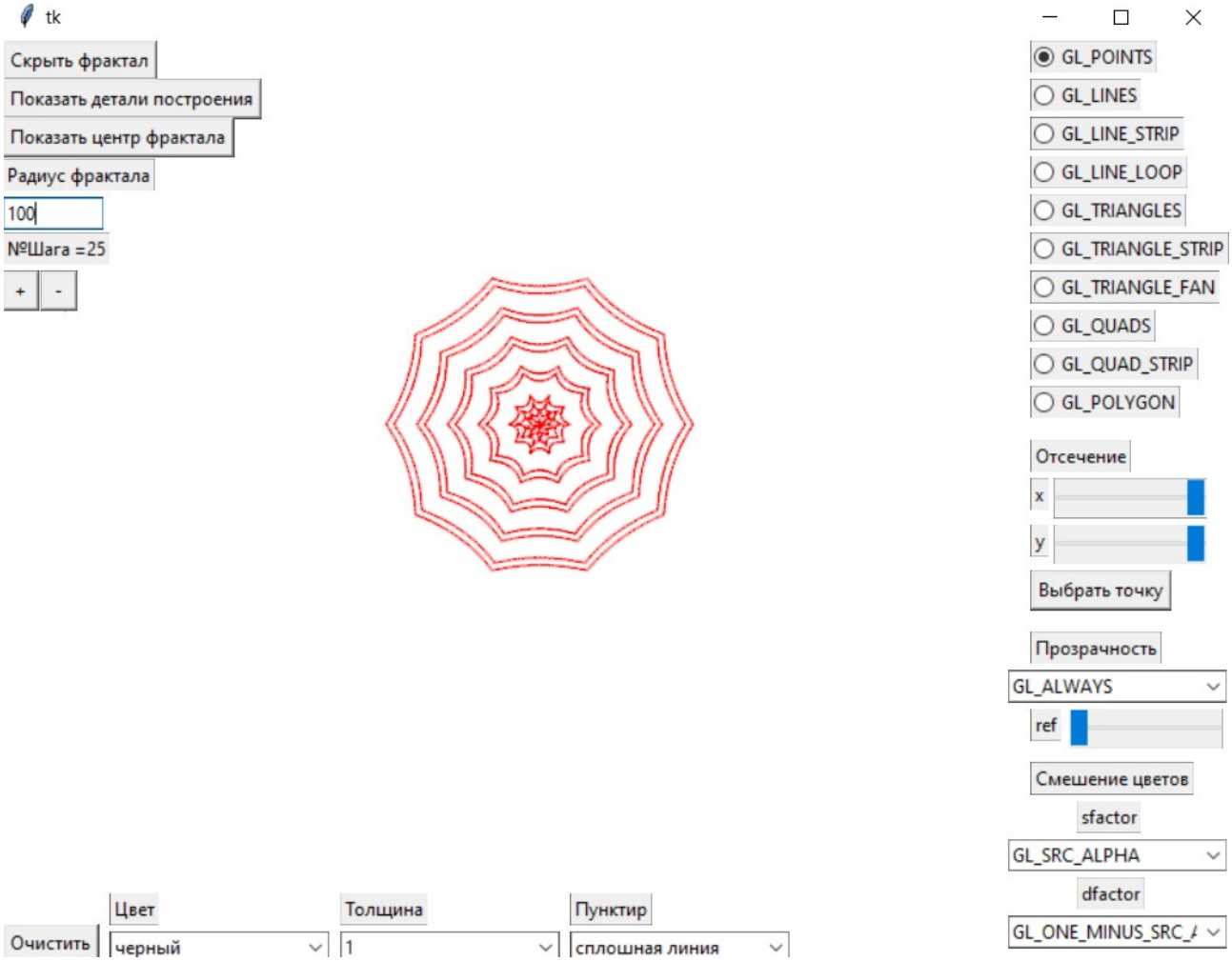

Рисунок 11 — Пример работы №1

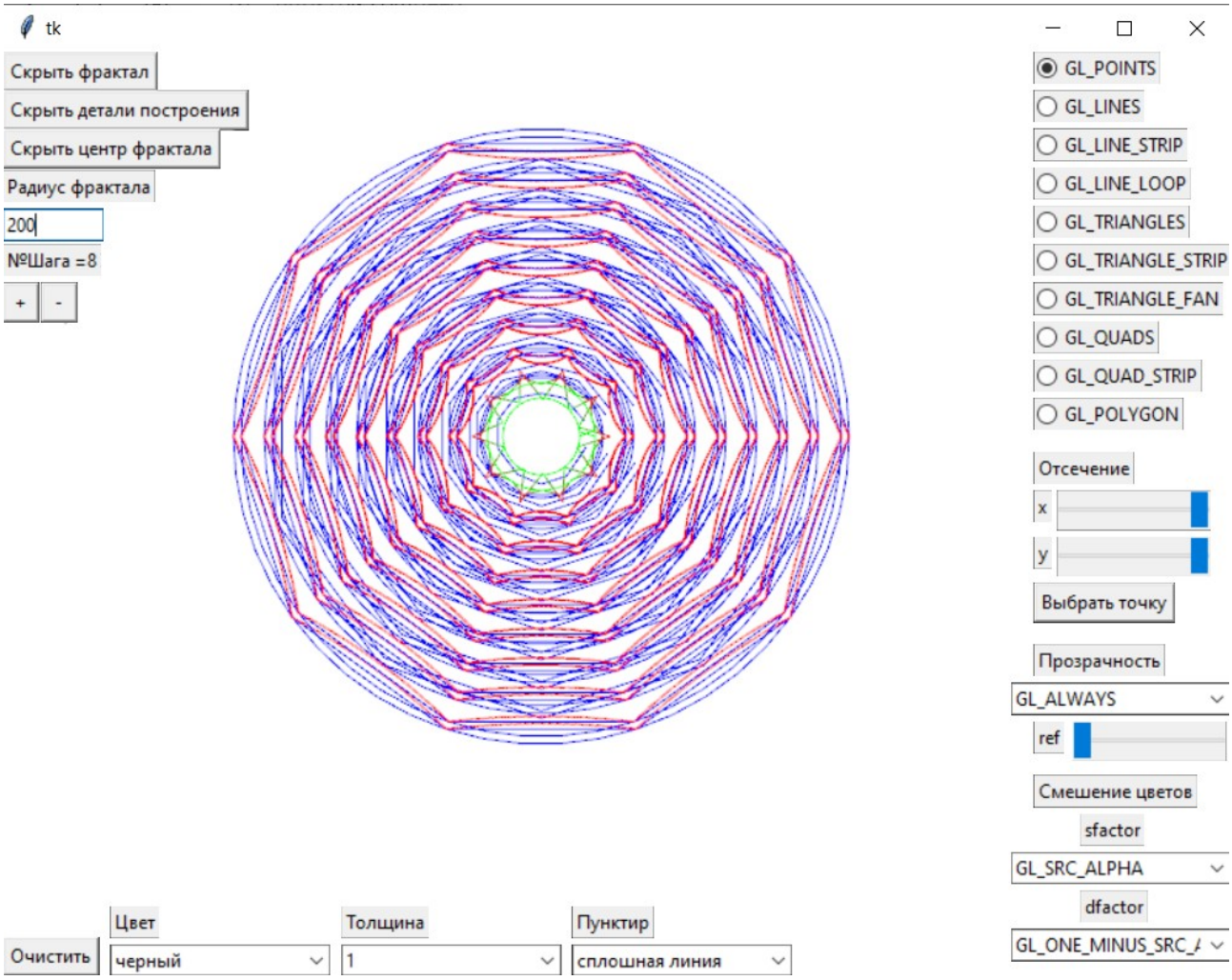

Рисунок 12 — Пример работы №2

Вывод.

На базе предыдущей лабораторной работы была разработана программа реализующая фрактал по индивидуальному заданию.

Приложение А. Исходный код.

Файл main.py

```
import re
from tkinter import *
from pyopengltk import OpenGLFrame
from OpenGL import GL, GLU
from tkinter.ttk import Combobox
from tkinter.ttk import Radiobutton
from tkinter.ttk import Scale
from random import uniform
import math
from tkinter.ttk import Entry


def scissor_click():  # вспомогательная функция для задания начальной точки
    if len(coordinates) > 0:
        start_point_scissor[0] = coordinates[-1][0]
        start_point_scissor[1] = coordinates[-1][1]
        del coordinates[-1]


def scissor_top_left(x, y, w, h):  # вспомогательная функция для теста отсечения
    GL.glScissor(x, window_height - h - y, w, h)


def get_dfactor():  # функция для установки dfactor
    temp_dfactor = combo_blend_dfactor.get()
    if temp_dfactor == "GL_ZERO":
        temp_dfactor = GL.GL_ZERO
    elif temp_dfactor == "GL_ONE":
        temp_dfactor = GL.GL_ONE
    elif temp_dfactor == "GL_SRC_COLOR":
        temp_dfactor = GL.GL_SRC_COLOR
    elif temp_dfactor == "GL_ONE_MINUS_SRC_COLOR":
        temp_dfactor = GL.GL_ONE_MINUS_SRC_COLOR
    elif temp_dfactor == "GL_SRC_ALPHA":
        temp_dfactor = GL.GL_SRC_ALPHA
    elif temp_dfactor == "GL_ONE_MINUS_SRC_ALPHA":
        temp_dfactor = GL.GL_ONE_MINUS_SRC_ALPHA
    elif temp_dfactor == "GL_DST_ALPHA":
        temp_dfactor = GL.GL_DST_ALPHA
    elif temp_dfactor == "GL_ONE_MINUS_DST_ALPHA":
        temp_dfactor = GL.GL_ONE_MINUS_DST_ALPHA
    return temp_dfactor


def get_sfactor():  # функция для установки sfactor
```

```
    temp_sfactor = combo_blend_sfactor.get()
    if temp_sfactor == "GL_ZERO":
        temp_sfactor = GL.GL_ZERO
    elif temp_sfactor == "GL_ONE":
        temp_sfactor = GL.GL_ONE
    elif temp_sfactor == "GL_DST_COLOR":
        temp_sfactor = GL.GL_DST_COLOR
    elif temp_sfactor == "GL_ONE_MINUS_DST_COLOR":
        temp_sfactor = GL.GL_ONE_MINUS_DST_COLOR
    elif temp_sfactor == "GL_SRC_ALPHA":
        temp_sfactor = GL.GL_SRC_ALPHA
    elif temp_sfactor == "GL_ONE_MINUS_SRC_ALPHA":
        temp_sfactor = GL.GL_ONE_MINUS_SRC_ALPHA
    elif temp_sfactor == "GL_DST_ALPHA":
        temp_sfactor = GL.GL_DST_ALPHA
    elif temp_sfactor == "GL_ONE_MINUS_DST_ALPHA":
        temp_sfactor = GL.GL_ONE_MINUS_DST_ALPHA
    elif temp_sfactor == "GL_SRC_ALPHA_SATURATE":
        temp_sfactor = GL.GL_SRC_ALPHA_SATURATE
    return temp_sfactor


def get_Alpha():  # функция для установки прозрачности
    temp_Alpha = combo_Alpha.get()
    if temp_Alpha == "GL_NEVER":
        temp_Alpha = GL.GL_NEVER
    elif temp_Alpha == "GL_LESS":
        temp_Alpha = GL.GL_LESS
    elif temp_Alpha == "GL_EQUAL":
        temp_Alpha = GL.GL_EQUAL
    elif temp_Alpha == "GL_LEQUAL":
        temp_Alpha = GL.GL_LEQUAL
    elif temp_Alpha == "GL_GREATER":
        temp_Alpha = GL.GL_GREATER
    elif temp_Alpha == "GL_NOTEQUAL":
        temp_Alpha = GL.GL_NOTEQUAL
    elif temp_Alpha == "GL_GEQUAL":
        temp_Alpha = GL.GL_GEQUAL
    elif temp_Alpha == "GL_ALWAYS":
        temp_Alpha = GL.GL_ALWAYS
    return temp_Alpha


def get_stipple():  # функция для установки пунктира
    temp_stipple = combo_stipple.get()
    if temp_stipple == "сплошная линия":
        GL.glLineStipple(1, 0xFFFF)
    elif temp_stipple == "точечный пунктир":
        GL.glLineStipple(1, 0x0101)
```

```
    elif temp_stipple == "штриховой пунктир":
        GL.glLineStipple(1, 0x00FF)
    elif temp_stipple == "штрих точка штрих":
        GL.glLineStipple(1, 0x1C47)


def get_color():  # функция для установки цвета
    temp_color = combo_color.get()
    if temp_color == "красный":
        color[0] = 255
        color[1] = 0
        color[2] = 0
        color[3] = uniform(0, 1)
    elif temp_color == "синий":
        color[0] = 0
        color[1] = 0
        color[2] = 255
        color[3] = uniform(0, 1)
    elif temp_color == "зеленый":
        color[0] = 0
        color[1] = 255
        color[2] = 0
        color[3] = uniform(0, 1)
    elif temp_color == "желтый":
        color[0] = 255
        color[1] = 255
        color[2] = 0
        color[3] = uniform(0, 1)
    elif temp_color == "черный":
        color[0] = 0
        color[1] = 0
        color[2] = 0
        color[3] = uniform(0, 1)


def clear():  # очистка списка координат вершин
    coordinates.clear()
    start_point_scissor[0] = 0
    start_point_scissor[1] = 0


def click(event):  # добавление координаты вершины по клику в окне
    if event.x < 670 and event.y < 555:
        get_color()
        temp_color = color.copy()
        coordinates.append((event.x, event.y, temp_color))


def draw():  # отрисовка
```

```
    primitive = GL.GL_POINT  # примитив по умолчанию
    line_width = combo_thickness.get()  # получение значения из виджета толщины
    temp = selected.get()  # получение примитива из выбранного виджета

    if temp == 0:  # установка примитивов для рисования (+толщины и типа линий для
примитивов вида линий)
        primitive = GL.GL_POINTS
    elif temp == 1:
        primitive = GL.GL_LINES
        GL.glLineWidth(int(line_width))
        GL.glEnable(GL.GL_LINE_STIPPLE)
        get_stipple()
    elif temp == 2:
        primitive = GL.GL_LINE_STRIP
        GL.glLineWidth(int(line_width))
        GL.glEnable(GL.GL_LINE_STIPPLE)
        get_stipple()
    elif temp == 3:
        primitive = GL.GL_LINE_LOOP
        GL.glLineWidth(int(line_width))
        GL.glEnable(GL.GL_LINE_STIPPLE)
        get_stipple()
    elif temp == 4:
        primitive = GL.GL_TRIANGLES
    elif temp == 5:
        primitive = GL.GL_TRIANGLE_STRIP
    elif temp == 6:
        primitive = GL.GL_TRIANGLE_FAN
    elif temp == 7:
        primitive = GL.GL_QUADS
    elif temp == 8:
        primitive = GL.GL_QUAD_STRIP
    elif temp == 9:
        primitive = GL.GL_POLYGON

    GL.glEnable(GL.GL_ALPHA_TEST)
    GL.glEnable(GL.GL_SCISSOR_TEST)
    GL.glEnable(GL.GL_BLEND)

    scissor_top_left(start_point_scissor[0], start_point_scissor[1], int(scale_scissor_x.get()),
                     int(scale_scissor_y.get()))
    GL.glAlphaFunc(get_Alpha(), float(scale_Alpha_ref.get()))
    GL.glBlendFunc(get_sfactor(), get_dfactor())

    GL.glBegin(primitive)  # непосредстенно отрисова виджета
    for i in range(len(coordinates)):  # в цикле отрисовка вершин по координатам из списка
        temp_color = coordinates[i][2]
        GL.glColor4f(temp_color[0], temp_color[1], temp_color[2], temp_color[3])  # установка
цвета
```

```
        GL.glVertex3f(coordinates[i][0], coordinates[i][1], 0.0)
    GL.glEnd()

    global is_on_main_fractal
    global is_on_details
    global step
    global is_on_add_part_1
    R = entry_R_fractal.get()
    if R == "" or R == 0:
        R = 250
    if is_on_main_fractal:
        draw_fractal_main_part(float(R), True, step, is_on_details, is_on_add_part_1)

    GL.glFlush()


def draw_fractal_main_part(R, double, count, turn_show=False, add_part_1=False):  # отрисовка фрактала
    GL.glLineWidth(1)
    GL.glColor3d(0, 0, 255)
    angles_points = []
    low_angles_points = []

    GL.glBegin(GL.GL_LINE_STRIP)

    for i in range(100):
        if turn_show:
            GL.glVertex2f(R * math.cos(2 * math.pi / 50 * i) + 350, R * math.sin(2 * math.pi / 50 * i) + 250)
        if i % 5 == 0:
            angles_points.append((R * math.cos(2 * math.pi / 50 * i) + 350, R * math.sin(2 * math.pi / 50 * i) + 250))

    for i in range(10):
        temp_y = (angles_points[i][1] + angles_points[i + 1][1]) / 2
        low_angles_points.append((((angles_points[i][0] + angles_points[i + 1][0]) / 2), temp_y))
    GL.glEnd()

    GL.glBegin(GL.GL_LINE_STRIP)
    for i in range(len(angles_points)):
        if turn_show:
            GL.glVertex3d(angles_points[i][0], angles_points[i][1], 0)
    GL.glEnd()

    GL.glBegin(GL.GL_LINE_STRIP)
    low_angles_points[0] = (low_angles_points[0][0] - 7, low_angles_points[0][1] - 5)
    low_angles_points[1] = (low_angles_points[1][0] - 6, low_angles_points[1][1] - 5)
    low_angles_points[2] = (low_angles_points[2][0], low_angles_points[2][1] - 7)
    low_angles_points[3] = (low_angles_points[3][0] + 7, low_angles_points[3][1] - 6)
```

```
low_angles_points[4] = (low_angles_points[4][0] + 8, low_angles_points[4][1] - 4)
low_angles_points[5] = (low_angles_points[5][0] + 8, low_angles_points[5][1] + 1)
low_angles_points[6] = (low_angles_points[6][0] + 5, low_angles_points[6][1] + 6)
low_angles_points[7] = (low_angles_points[7][0], low_angles_points[7][1] + 10)
low_angles_points[8] = (low_angles_points[8][0] - 7, low_angles_points[8][1] + 6)
low_angles_points[9] = (low_angles_points[9][0] - 8, low_angles_points[9][1] + 2)

for i in range(len(low_angles_points)):
    if turn_show:
        GL.glVertex3d(low_angles_points[i][0], low_angles_points[i][1], 0)
GL.glEnd()

if count > 0 or (not double and count == -1):
    GL.glBegin(GL.GL_POINTS)
    GL.glColor3d(255, 0, 0)
    for i in range(10):
        control_points = [angles_points[i], low_angles_points[i], angles_points[i + 1]]
        t = 0

        while t <= 1:
            x = (1 - t) ** 2 * control_points[0][0] + 2 * (1 - t) * t * control_points[1][0] + t ** 2 * \
                control_points[2][0]
            y = (1 - t) ** 2 * control_points[0][1] + 2 * (1 - t) * t * control_points[1][1] + t ** 2 * \
                control_points[2][1]
            t += 0.01
            GL.glVertex3d(x, y, 0)
    GL.glEnd()

if double and count > 0:
    angles_points.clear()
    low_angles_points.clear()
    draw_fractal_main_part(R - 5, False, -1, turn_show, add_part_1)

if count > 0 and R > 0:
    angles_points.clear()
    low_angles_points.clear()
    draw_fractal_main_part(R - 20, True, count - 1, turn_show, add_part_1)

elif count == 0 and add_part_1:
    low_angles_points[0] = (384, 259)
    low_angles_points[1] = (370, 278)
    low_angles_points[2] = (351, 285)
    low_angles_points[3] = (331, 279)
    low_angles_points[4] = (316, 259)
    low_angles_points[5] = (317, 237)
    low_angles_points[6] = (330, 221)
    low_angles_points[7] = (350, 216)
    low_angles_points[8] = (373, 223)
    low_angles_points[9] = (383, 240)
```

```
    GL.glBegin(GL.GL_LINE_LOOP)
    for i in range(len(low_angles_points)):
        GL.glColor3d(255, 0, 0)
        GL.glVertex3d(angles_points[i][0], angles_points[i][1], 0)
        GL.glColor3d(0, 255, 0)
        GL.glVertex3d(low_angles_points[i][0], low_angles_points[i][1], 0)
    GL.glEnd()

    low_angles_points[0] = (373, 257)
    low_angles_points[1] = (365, 270)
    low_angles_points[2] = (351, 274)
    low_angles_points[3] = (339, 271)
    low_angles_points[4] = (326, 257)
    low_angles_points[5] = (325, 241)
    low_angles_points[6] = (336, 228)
    low_angles_points[7] = (351, 226)
    low_angles_points[8] = (368, 233)
    low_angles_points[9] = (374, 244)

    angles_points.clear()

    for i in range(100):
        if turn_show:
            GL.glVertex2f(45 * math.cos(2 * math.pi / 50 * i) + 350, 45 * math.sin(2 * math.pi / 50 * i)
+ 250)
        if i % 5 == 0:
            angles_points.append(
                (45 * math.cos(2 * math.pi / 50 * i) + 350, 45 * math.sin(2 * math.pi / 50 * i) + 250))

    GL.glBegin(GL.GL_LINE_LOOP)
    for i in range(len(low_angles_points)):
        GL.glColor3d(255, 0, 0)
        GL.glVertex3d(angles_points[i][0], angles_points[i][1], 0)
        GL.glColor3d(0, 255, 0)
        GL.glVertex3d(low_angles_points[i][0], low_angles_points[i][1], 0)
    GL.glEnd()

    GL.glBegin(GL.GL_LINE_LOOP)

    if turn_show:
        for i in range(100):
            GL.glVertex2f(25 * math.cos(2 * math.pi / 50 * i) + 350, 25 * math.sin(2 * math.pi / 50 * i)
+ 250)

        for i in range(100):
            GL.glVertex2f(35 * math.cos(2 * math.pi / 50 * i) + 350, 35 * math.sin(2 * math.pi / 50 * i)
+ 250)
    GL.glEnd()
```

```
def switch_button_main_fractal():
    global is_on_main_fractal

    if is_on_main_fractal:
        button_main_fractal.config(text="Показать фрактал")
        is_on_main_fractal = False
    else:
        button_main_fractal.config(text="Скрыть фрактал")
        is_on_main_fractal = True


def switch_button_details_fractal():
    global is_on_details

    if is_on_details:
        button_details_fractal.config(text="Показать детали построения")
        is_on_details = False
    else:
        button_details_fractal.config(text="Скрыть детали построения")
        is_on_details = True


def is_valid_R(newval):
    return re.match("^\d{0,3}$", newval) is not None


def plus_step():
    global step
    if step < 30:
        step += 1
    label_step.config(text="№Шага =" + str(step))


def minus_step():
    global step
    if step > 1:
        step -= 1
    label_step.config(text="№Шага =" + str(step))


def switch_button_fractal_mid():
    global is_on_add_part_1
    if is_on_add_part_1:
        button_add_part_1.config(text="Показать центр фрактала")
        is_on_add_part_1 = False
    else:
        button_add_part_1.config(text="Скрыть центр фрактала")
```

```
        is_on_add_part_1 = True


class DrawingWindow(OpenGLFrame):  # создание класса на основе пакета pyopengltk

    def initgl(self):  # инициализация
        GL.glClear(GL.GL_COLOR_BUFFER_BIT)  # очистка буфферов от цветов ~ очищение
экарана
        GL.glClearColor(1, 1, 1, 0)  # задает цвет, в который окно будет окрашиваться при его
очистке ~ очищение
        # цветопередачи
        GL.glMatrixMode(GL.GL_PROJECTION)  # матрица проекции (для проецирования 3D
прсостранства в 2D)
        GL.glLoadIdentity()  # единичная матрица ~ очистика
        GLU.gluOrtho2D(0, window_width, window_height, 0)

    def redraw(self):  # перерисовка
        GL.glClear(GL.GL_COLOR_BUFFER_BIT)
        draw()  # функция для отрисовки


points = [(100, 100), (150, 150), (50, 50)]

root = Tk()  # главное окно

window_width = 800  # размеры окна (ширина и высота)
window_height = 600
coordinates = []  # (координаты вершин)
color = [0, 0, 0, 0.0]  # (цвет  вершин)

app = DrawingWindow(root, width=window_width, height=window_height)  # создание окна для
отрисови
app.bind('<Button 1>', click)  # биндим ПКМ по созданному окну, чтобы запоминать
координаты клика
app.pack(fill=BOTH, expand=YES)  # отобразить
app.animate = 1  # для отрисовки в реальном времени

selected = IntVar()  # сюда помещается значение выбранного примитива
rad1 = Radiobutton(app, text='GL_POINTS', value=0, variable=selected)  # радиокнопки для
выбора примитива
rad2 = Radiobutton(app, text='GL_LINES', value=1, variable=selected)
rad3 = Radiobutton(app, text='GL_LINE_STRIP', value=2, variable=selected)
rad4 = Radiobutton(app, text='GL_LINE_LOOP', value=3, variable=selected)
rad5 = Radiobutton(app, text='GL_TRIANGLES', value=4, variable=selected)
rad6 = Radiobutton(app, text='GL_TRIANGLE_STRIP', value=5, variable=selected)
rad7 = Radiobutton(app, text='GL_TRIANGLE_FAN', value=6, variable=selected)
rad8 = Radiobutton(app, text='GL_QUADS', value=7, variable=selected)
rad9 = Radiobutton(app, text='GL_QUAD_STRIP', value=8, variable=selected)
rad10 = Radiobutton(app, text='GL_POLYGON', value=9, variable=selected)
```

```
rad1.place(x=670, y=0)  # их отрбражение
rad2.place(x=670, y=25)
rad3.place(x=670, y=50)
rad4.place(x=670, y=75)
rad5.place(x=670, y=100)
rad6.place(x=670, y=125)
rad7.place(x=670, y=150)
rad8.place(x=670, y=175)
rad9.place(x=670, y=200)
rad10.place(x=670, y=225)

btn = Button(app, text="Очистить", command=clear)  # кнопка для очистки
btn.place(x=0, y=575)

label_color = Label(text="Цвет")  # виджеты для выбора цвета вершин
label_color.place(x=70, y=555)
combo_color = Combobox(app)
combo_color['values'] = ("красный", "синий", "зеленый", "желтый", "черный")
combo_color['state'] = 'readonly'
combo_color.current(4)
combo_color.place(x=70, y=580)

label_thickness = Label(text="Толщина")  # виджеты для выбора толщины линий
label_thickness.place(x=220, y=555)
combo_thickness = Combobox(app)
combo_thickness['values'] = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
combo_thickness['state'] = 'readonly'
combo_thickness.current(0)
combo_thickness.place(x=220, y=580)

label_stipple = Label(text="Пунктир")  # виджеты для выбора типа пунктира
label_stipple.place(x=370, y=555)
combo_stipple = Combobox(app)
combo_stipple['values'] = ("сплошная линия", "точечный пунктир", "штриховой пунктир",
"штрих точка штрих")
combo_stipple['state'] = 'readonly'
combo_stipple.current(0)
combo_stipple.place(x=370, y=580)

start_point_scissor = [0, 0]
label_scissor = Label(text="Отсечение")  # виджеты для теста отсечения
label_scissor.place(x=670, y=260)
label_scissor_x = Label(text="x")
label_scissor_x.place(x=670, y=285)
scale_scissor_x = Scale(from_=start_point_scissor[0], to=670, orient=HORIZONTAL, value=670)
scale_scissor_x.place(x=685, y=285)
label_scissor_y = Label(text="y")
label_scissor_y.place(x=670, y=315)
```

```
scale_scissor_y = Scale(from_=start_point_scissor[1], to=555, orient=HORIZONTAL, value=555)
scale_scissor_y.place(x=685, y=315)
scale_button = Button(app, text="Выбрать точку", command=scissor_click)
scale_button.place(x=670, y=345)

label_Alpha = Label(text="Прозрачность")  # виджеты для теста прозрачности
label_Alpha.place(x=670, y=385)
combo_Alpha = Combobox(app)
combo_Alpha['values'] = ("GL_NEVER", "GL_LESS", "GL_EQUAL", "GL_LEQUAL", "GL_GREATER",
"GL_NOTEQUAL", "GL_GEQUAL",
                         "GL_ALWAYS")
combo_Alpha['state'] = 'readonly'
combo_Alpha.current(7)
combo_Alpha.place(x=655, y=410)
label_Alpha_ref = Label(text="ref")
label_Alpha_ref.place(x=670, y=435)
scale_Alpha_ref = Scale(from_=0, to=1, orient=HORIZONTAL, value=0)
scale_Alpha_ref.place(x=695, y=435)

label_blend = Label(text="Смешение цветов")  # виджеты для теста смешения цветов
label_blend.place(x=670, y=470)
label_blend_sfactor = Label(text="sfactor")
label_blend_sfactor.place(x=700, y=495)
combo_blend_sfactor = Combobox(app)
combo_blend_sfactor['values'] = ("GL_ZERO", "GL_ONE", "GL_DST_COLOR",
"GL_ONE_MINUS_DST_COLOR", "GL_SRC_ALPHA",
                         "GL_ONE_MINUS_SRC_ALPHA", "GL_DST_ALPHA",
"GL_ONE_MINUS_DST_ALPHA",
                         "GL_SRC_ALPHA_SATURATE")
combo_blend_sfactor['state'] = 'readonly'
combo_blend_sfactor.current(4)
combo_blend_sfactor.place(x=655, y=520)
label_blend_dfactor = Label(text="dfactor")
label_blend_dfactor.place(x=700, y=545)
combo_blend_dfactor = Combobox(app)
combo_blend_dfactor['values'] = ("GL_ZERO", "GL_ONE", "GL_SRC_COLOR",
"GL_ONE_MINUS_SRC_COLOR", "GL_SRC_ALPHA",
                         "GL_ONE_MINUS_SRC_ALPHA", "GL_DST_ALPHA",
"GL_ONE_MINUS_DST_ALPHA")
combo_blend_dfactor['state'] = 'readonly'
combo_blend_dfactor.current(5)
combo_blend_dfactor.place(x=655, y=570)

is_on_main_fractal = False
button_main_fractal = Button(app, text="Показать фрактал",
command=switch_button_main_fractal)
button_main_fractal.place(x=0, y=0)

is_on_details = False
```

```
button_details_fractal = Button(app, text="Показать детали построения",
command=switch_button_details_fractal)
button_details_fractal.place(x=0, y=25)

is_on_add_part_1 = False
button_add_part_1 = Button(app, text="Показать центр фрактала",
command=switch_button_fractal_mid)
button_add_part_1.place(x=0, y=50)

check_valid_R = (app.register(is_valid_R), "%P")
label_Entry = Label(text="Радиус фрактала")
label_Entry.place(x=0, y=77)
entry_R_fractal = Entry(width=10, validate="key", validatecommand=check_valid_R)
entry_R_fractal.place(x=0, y=102)
entry_R_fractal.insert(0, "250")

step = 1
label_step = Label(text="№Шага = 1")
label_step.place(x=0, y=125)
button_plus_step = Button(app, text="+", command=plus_step, width=2)
button_plus_step.place(x=0, y=150)
button_minus_step = Button(app, text="-", command=minus_step, width=2)
button_minus_step.place(x=25, y=150)

app.mainloop()  # запуск главного цикла
```