Санкт-Петербургский Национальный Исследовательский
Университет Информационных Технологий, Механики и Оптики
Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа № 1

Построение и визуализация фрактальных множеств по дисциплине ТФКП

Выполнил
Студент группы P3210
Чжун Цзяцзюнь : 407959
Преподаватель:
Богачёв Владимир Александрович

Санкт-Петербург 2025 г.

Оглавление

Задание

1 Докажите свойства 1 и 2 для множества Мандельброта.

2 Напишите программу, которая будет строить визуализацию множества Мандельброта. Выберите разумные ограничения, поварьируйте максимальное количество итераций. Попробуйте приблизить отдельные части множества, чтобы увидеть фрактальную структуру.

3 Напишите программу, которая по заданному с строит заполненное множество Жюлиа. Поварьируйте максимальное количество итераций, попробуйте пронаблюдать фрактальную структуру, рассмотрите множество при разных с. (Например, красиво получается при с = 05251993+i05251993).

4 Найдите какой-нибудь неразобранный фрактал (например, бассейны Ньютона). Опишите его структуру, построение. Нарисуйте визуализации. Будьте готовы выступить с докладом перед своими одногруппниками.

Доказательства свойств для множества Мандельброта

1 Свойство 1

Поскольку $Z_{n+1} = {Z_n}^2 + c,\ Z_0 = 0$

Для любого комплексного числа $c = a + bi, \bar{c} = a - bi$

Нам надо доказать, что $\overline{Z_n(c)} = Z_n(\bar{c})$ по индукции:

$$\overline{Z_{n+1}(c)} = \overline{Z_n(c)^2 + c} = \overline{Z_n(c)^2} + \bar{c} = Z_{n+1}(\bar{c})$$

Следовательно, последовательность ограничена для с тогда и только тогда, когда ограничена для c̄

Поэтому они симметрично относительно вещественной оси.

2 Свойство 2

Пусть $|c| = 2 + a$

Если$|Z_n| > 2$, то $|Z_{n+1}| = \left|{Z_n}^2 + c\right| \geq \left|{Z_n}^2\right| - |c| \geq |Z_n|(|Z_n| - 1) > |Z_n|$

При$|c| > 2$ имеет $|Z_1| = |c| > 2$,

значит последовательность монотонно возрастает и расходится

Кода программ для построения множеств Мандельброта и Жюлиа.

```python
import turtle
import time

def mandelbrot(c, max_iter):
    """
    Вычисляет количество итераций для точки c в множестве Мандельброта
    c: комплексная точка
    max_iter: максимальное количество итераций
    возвращает: количество итераций до убегания
    """
    z = 0
    for n in range(max_iter):
        if abs(z) > 2:  # условие убегания
            return n
        z = z*z + c    # формула итерации Мандельброта
    return max_iter     # если не убежало, возвращаем max_iter

def draw_mandelbrot_complete():
    # Настройка холста
    screen = turtle.Screen()
    screen.setup(800, 800)
    screen.title("Множество Мандельброта ")
    screen.bgcolor("black")
    screen.tracer(0, 0)  # отключаем анимацию

    # Создаем черепаху
    t = turtle.Turtle()
    t.speed(0)
    t.hideturtle()
    t.penup()

    # Определяем разные области просмотра (для увеличения)
    views = [
        # (x_min, x_max, y_min, y_max, описание, рекомендованное количество
итераций)
        (-2.0, 1.0, -1.5, 1.5, "Полный вид", 50),
        (-0.5, 0.5, -0.5, 0.5, "Увеличение центра", 100),
    ]

    # Определяем разные максимальные количества итераций
    max_iters = [20, 50, 100]

    print("Программа визуализации множества Мандельброта")

    # Рисуем изображения для каждой комбинации вида и количества итераций

```

```
    for view_idx, (x_min, x_max, y_min, y_max, description,
recommended_iter) in enumerate(views):
        print(f"\nВид {view_idx+1}: {description}")
        print(f"Диапазон координат: x[{x_min:.2f}, {x_max:.2f}],
y[{y_min:.2f}, {y_max:.2f}]")

        for iter_idx, max_iter in enumerate(max_iters):
            print(f"  Рисование... Количество итераций: {max_iter}")
            start_time = time.time()

            # Очищаем холст
            t.clear()
            screen.bgcolor("black")

            # Устанавливаем разрешение (настраиваем по размеру вида)
            if (x_max - x_min) < 1.0:
                width, height = 400, 400  # для увеличенных видов
используем большее разрешение
            else:
                width, height = 300, 300  # для полного вида используем
меньшее разрешение

            # Масштабные коэффициенты
            x_scale = (x_max - x_min) / width
            y_scale = (y_max - y_min) / height

            # Проходим по каждому пикселю
            for x_pixel in range(width):
                for y_pixel in range(height):
                    # Преобразуем координаты пикселя в комплексные
координаты
                    x = x_min + x_pixel * x_scale
                    y = y_min + y_pixel * y_scale
                    c = complex(x, y)

                    # Вычисляем количество итераций
                    iter_count = mandelbrot(c, max_iter)

                    # Выбираем цвет в зависимости от количества итераций
                    if iter_count == max_iter:
                        color = "black"  # точки внутри множества
Мандельброта
                    else:
                        # Создаем цветовой эффект
                        intensity = iter_count / max_iter
                        if intensity < 0.2:
```

```
                            color = "dark blue"
                        elif intensity < 0.4:
                            color = "blue"
                        elif intensity < 0.6:
                            color = "cyan"
                        elif intensity < 0.8:
                            color = "yellow"
                        else:
                            color = "red"

                    # Рисуем точку
                    screen_x = (x_pixel - width/2) * 800/width
                    screen_y = (height/2 - y_pixel) * 800/height
                    t.goto(screen_x, screen_y)
                    t.dot(800//width, color)

                # Обновляем экран после каждой строки
                if x_pixel % 50 == 0:
                    screen.update()

            screen.update()
            end_time = time.time()
            print(f"    Завершено! Время: {end_time - start_time:.2f}
секунд")

            # Ждем, пока пользователь посмотрит
            print("    Нажмите любую клавишу для продолжения...")
            input()

    print("\nВсе виды отрисованы!")
    screen.exitonclick()

# Запускаем программу
if __name__ == "__main__":
    draw_mandelbrot_complete()
```

```python
import turtle
import time

def mandelbrot(c, max_iter):
    """
    Вычисляет количество итераций для точки c в множестве Мандельброта
    c: комплексная точка
    max_iter: максимальное количество итераций
    возвращает: количество итераций до убегания
    """
    z = 0
    for n in range(max_iter):
        if abs(z) > 2:  # условие убегания
            return n
        z = z*z + c    # формула итерации Мандельброта
    return max_iter     # если не убежало, возвращаем max_iter

def draw_mandelbrot_complete():
    # Настройка холста
    screen = turtle.Screen()
    screen.setup(800, 800)
    screen.title("Множество Мандельброта - полная версия")
    screen.bgcolor("black")
    screen.tracer(0, 0)  # отключаем анимацию

    # Создаем черепаху
    t = turtle.Turtle()
    t.speed(0)
    t.hideturtle()
    t.penup()

    # Определяем разные области просмотра (для увеличения)
    views = [
        # (x_min, x_max, y_min, y_max, описание, рекомендованное количество
итераций)
        (-2.0, 1.0, -1.5, 1.5, "Полный вид", 50),
        (-0.5, 0.5, -0.5, 0.5, "Увеличение центра", 100),
    ]

    # Определяем разные максимальные количества итераций
    max_iters = [20, 50, 100]

    print("Программа визуализации множества Мандельброта")

    # Рисуем изображения для каждой комбинации вида и количества итераций

```

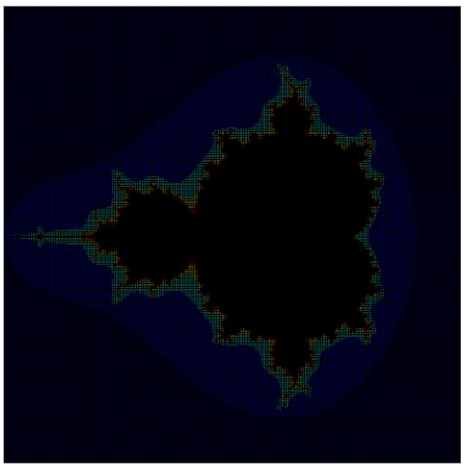
```
    for view_idx, (x_min, x_max, y_min, y_max, description,
recommended_iter) in enumerate(views):
        print(f"\nВид {view_idx+1}: {description}")
        print(f"Диапазон координат: x[{x_min:.2f}, {x_max:.2f}],
y[{y_min:.2f}, {y_max:.2f}]")

        for iter_idx, max_iter in enumerate(max_iters):
            print(f"  Рисование... Количество итераций: {max_iter}")
            start_time = time.time()

            # Очищаем холст
            t.clear()
            screen.bgcolor("black")

            # Устанавливаем разрешение (настраиваем по размеру вида)
            if (x_max - x_min) < 1.0:
                width, height = 400, 400  # для увеличенных видов
используем большее разрешение
            else:
                width, height = 300, 300  # для полного вида используем
меньшее разрешение

            # Масштабные коэффициенты
            x_scale = (x_max - x_min) / width
            y_scale = (y_max - y_min) / height

            # Проходим по каждому пикселю
            for x_pixel in range(width):
                for y_pixel in range(height):
                    # Преобразуем координаты пикселя в комплексные
координаты
                    x = x_min + x_pixel * x_scale
                    y = y_min + y_pixel * y_scale
                    c = complex(x, y)

                    # Вычисляем количество итераций
                    iter_count = mandelbrot(c, max_iter)

                    # Выбираем цвет в зависимости от количества итераций
                    if iter_count == max_iter:
                        color = "black"  # точки внутри множества
Мандельброта
                    else:
                        # Создаем цветовой эффект
                        intensity = iter_count / max_iter
                        if intensity < 0.2:
```

```
                            color = "dark blue"
                        elif intensity < 0.4:
                            color = "blue"
                        elif intensity < 0.6:
                            color = "cyan"
                        elif intensity < 0.8:
                            color = "yellow"
                        else:
                            color = "red"

                    # Рисуем точку
                    screen_x = (x_pixel - width/2) * 800/width
                    screen_y = (height/2 - y_pixel) * 800/height
                    t.goto(screen_x, screen_y)
                    t.dot(800//width, color)

                # Обновляем экран после каждой строки
                if x_pixel % 50 == 0:
                    screen.update()

            screen.update()
            end_time = time.time()
            print(f"    Завершено! Время: {end_time - start_time:.2f}
секунд")

            # Ждем, пока пользователь посмотрит
            print("    Нажмите любую клавишу для продолжения...")
            input()

    print("\nВсе виды отрисованы!")
    screen.exitonclick()

# Запускаем программу
if __name__ == "__main__":
    draw_mandelbrot_complete()
```

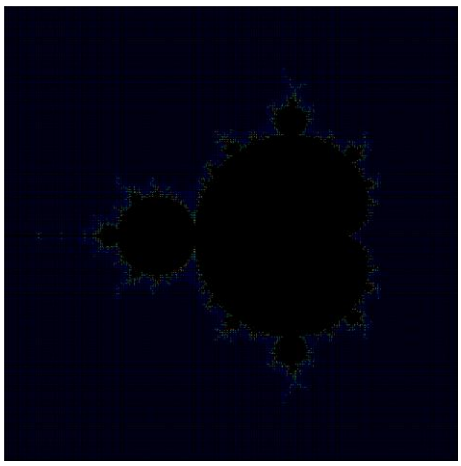
Набора изображений, построенных при разном числе итераций и приближении.

1. Множество Мандельброта

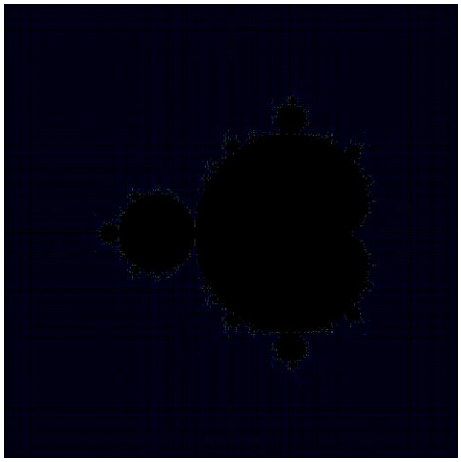
Максимальное количество итераций - 20



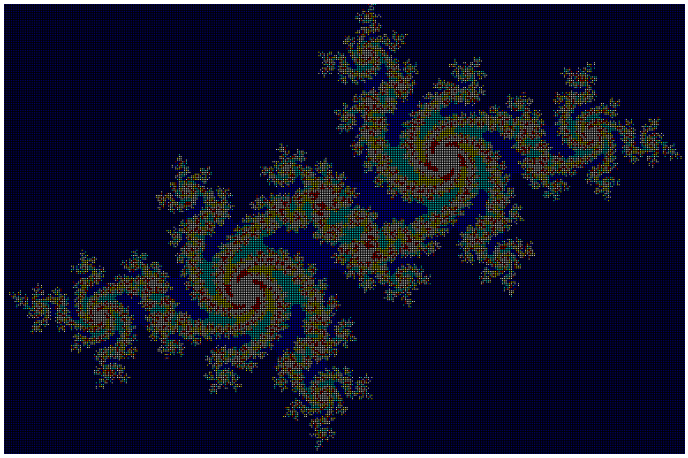
Максимальное количество итераций - 50



Максимальное количество итераций - 100



2 . Множество Жюлиа



Анализ фрактала "Снежинка Коха" (Koch Snowflake)

Снежинка Коха - это классический фрактальный объект, впервые описанный шведским математиком Хельге фон Кохом в 1904 году. Это один из самых ранних и известных фракталов.

Математическое построение:

1. **Начальная фигура**: Равносторонний треугольник (нулевой порядок)
2. **Рекурсивный процесс**:
    - Каждая сторона делится на 3 равные части
    - На средней части строится новый равносторонний треугольник
    - Средняя часть удаляется
3. **Процесс повторяется** для каждой новой стороны бесконечно

Код для создания визуального изображения:

```python
import time
import turtle

# function of making koch snowflake
# size is the total length of the 0-order straight line, n is the order
def koch(size, n):
    if n == 0:
        turtle.fd(size)
    else:
        # The angle the turtle turns when drawing the one order
        for angle in [0, 60, -120, 60]:
            turtle.left(angle)
            koch(size/3, n-1)

def main(size, n):
    turtle.setup(800, 400)
    turtle.pensize(2)
    turtle.penup()
    turtle.goto(-300, 100)
    turtle.pendown()
    turtle.pencolor('red')
    koch(size, n)
    turtle.right(120)
    turtle.pencolor('green')
    koch(size, n)
    turtle.right(120)
    turtle.pencolor('blue')
    koch(size, n)
    turtle.right(120)
    turtle.hideturtle()

size, n = eval(input('Please enter the size, n: '))
main(size, n)
print("Program Ended")
time.sleep(5)
```

Size ,n: (100,0), (100,1), (100,2), (100,3)