KATHMANDU UNIVERSITY

Computer Science
2021

Fractal Generator

Students:
Bishesh Raj Khanal (26)
Manish Pokharel (42)

Supervisor:
Dhiraj Shrestha
Department of Computer Science and Engineering
Kathmandu University

June 15, 2024

Contents

List of Figures

1 Introduction

The visualization of complex and intricate patterns has long captivated both mathematicians and artists alike. In the realm of computational mathematics and graphics, the Mandelbrot and Julia sets stand as iconic examples of such patterns. These sets, originating from the study of complex dynamics, exhibit fractal properties that manifest visually in stunning and infinitely detailed forms.

This project delves into the exploration and visualization of Mandelbrot and Julia sets through the lens of OpenGL and C++. By leveraging the computational power of modern graphics processing units (GPUs), we aim to render these sets in real-time, allowing for interactive exploration and discovery of their intricate structures.

1.1 Objectives and Goals

The objectives of this project are:

- To implement algorithms for generating Mandelbrot and Julia sets using C++.
- To develop a real-time visualization application using OpenGL for rendering these sets.
- To enable interactive exploration of Mandelbrot and Julia sets, including zooming and panning functionalities.
- To optimize the rendering process for efficient performance, leveraging GPU capabilities.

1.2 Applications and Significance

Fractals are not just theoretical constructs; they have practical applications in various fields:

1. **Nature:** Fractals describe natural phenomena such as coastlines, mountains, clouds, and plants.
2. **Computer Graphics:** Fractal algorithms are used to generate realistic landscapes and textures.
3. **Signal and Image Processing:** Fractal compression techniques are used for efficient storage of images.
4. **Mathematics and Physics:** Fractals provide insights into chaotic systems and complex dynamics.

2 Background and Theory

2.1 Introduction to Fractals

Fractals are intricate geometric shapes that can be split into parts, each of which is a reduced-scale copy of the whole. This property is known as self-similarity. Fractals are used in various fields including physics, computer science, art, and nature, where they describe complex structures such as coastlines, mountain ranges, clouds, and plants.

2.1.1 Basic Properties of Fractals

1. **Self-Similarity:** Fractals exhibit self-similarity, which means their shape looks similar at any scale.
2. **Fractional Dimension:** Unlike traditional geometric shapes that have integer dimensions (1D line, 2D square, 3D cube), fractals have non-integer dimensions. This is known as their fractal dimension.
3. **Formation by Iteration:** Many fractals are formed by iterative processes, where a simple process is repeated over and over.

2.1.2 Example: The Koch Snowflake

The Koch Snowflake is a classic example of a fractal. It is constructed by starting with an equilateral triangle and recursively adding smaller triangles to each side. The perimeter of the snowflake increases infinitely, while its area remains finite.

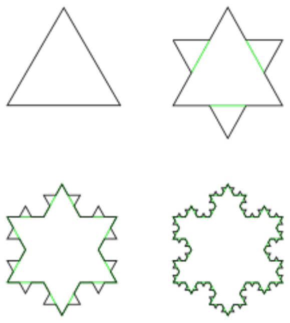

Figure 1: The Koch Slowflake

2.2 The MandelBrot Set

The Mandelbrot set is one of the most well-known fractals and is defined in the complex plane. It is named after Benoît B. Mandelbrot, who studied and popularized it.

2.2.1 Mathematical Definition

The Mandelbrot set M is defined as the set of complex numbers c for which the sequence $\{z_n\}$ remains bounded, where $z_0 = 0$ and $z_{n+1} = z_n^2 + c$. Formally, this can be written as:

$$M = \{c \in C \mid \limsup_{n\to\infty} |z_n| < \infty \text{ where } z_0 = 0 \text{ and } z_{n+1} = z_n^2 + c\}$$

2.2.2 Properties

- **Boundedness**: A complex number c is in the Mandelbrot set if, when iterating $z_{n+1} = z_n^2 + c$ starting from $z_0 = 0$, the absolute value of z_n never exceeds a certain threshold.
- **Fractal Dimension**: The boundary of the Mandelbrot set has a fractal dimension of 2, meaning it is as complex as a two-dimensional surface.

2.2.3 Visualization

The Mandelbrot set M is a famous fractal in complex dynamics. It is visualized by assigning colors to points c in the complex plane based on whether the sequence $z_{n+1} = z_n^2 + c$ remains bounded or tends to infinity, starting from $z_0 = 0$.

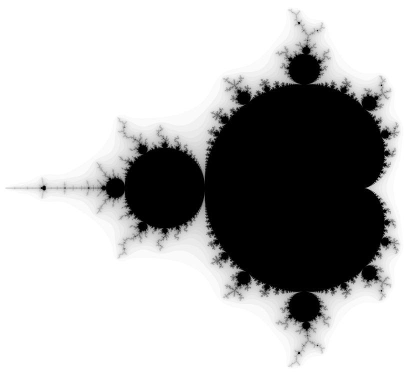

Figure 2: Visualization of the Mandelbrot set

In figure, points colored black are in the set M; other colors represent how quickly points escape to infinity.

2.3 Julia Sets

Julia sets are closely related to the Mandelbrot set and are also defined in the complex plane. However, instead of varying the parameter c, Julia sets are generated by fixing c and varying the initial value z_0.

2.3.1 Mathematical Definition

Given a fixed complex number c, the Julia set $J(c)$ is defined as:

$$J(c) = \{z_0 \in C \mid \limsup_{n\to\infty} |z_n| < \infty \text{ where } z_{n+1} = z_n^2 + c\}$$

2.3.2 Visualization

The Julia set $J(c)$ for a fixed complex number c is visualized by computing the set of points $z_0 \in C$ such that the sequence $z_{n+1} = z_n^2 + c$ remains bounded. Here's an example of the visualization:

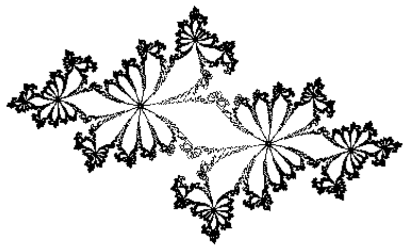

Figure 3: Visualization of the Julia set for $c = -0.69 + 0.28i$

2.3.3 Relationship to the Mandelbrot Set

The nature of the Julia set $J(c)$ is determined by the parameter c. if c is in the Mandelbrot Set, the corresponding Julia set is connected. if c is outside the Mandelbrot set, the Julia set is a Cantor set (totally disconnected).

3 Implementation: Visualizing Mandelbrot and Julia Sets using OpenGL in C++

3.1 Setup and Initialization

- **OpenGL and GLFW Initialization:** The project utilizes GLFW for window and input management, and OpenGL for rendering. GLFW is initialized to create a window and set up a rendering context.
- **Window Dimensions and View Setup:** The initial window dimensions are set to 1920x1080 pixels. OpenGL's orthographic projection is configured to map the screen coordinates to the complex plane for rendering Mandelbrot and Julia sets.

3.2 Drawing Mandelbrot Set

- **Function `draw_mandelbrot`:** This function calculates and renders the Mandelbrot set using OpenGL.
- **Algorithm:**

Algorithm 1 Drawing Mandelbrot Set

```
procedure DRAW_MANDELBROT(window_width, window_height, max_iterations)
    Input:
        window_width : Width of the window in pixels
        window_height : Height of the window in pixels
        max_iterations : Maximum number of iterations to determine divergence
    Output:
        Renders the Mandelbrot set using OpenGL immediate mode
    Begin:
        glBegin(GL_POINTS)
    for $x \leftarrow 0$ to window_width $- 1$ do
        for $y \leftarrow 0$ to window_height $- 1$ do
            $zx \leftarrow 1.5 \times (x -$ window_width$/2.0)/(0.5 \times$ zoom $\times$ window_width$) +$ offset_x
            $zy \leftarrow (y -$ window_height$/2.0)/(0.5 \times$ zoom $\times$ window_height$) +$ offset_y
            $c \leftarrow$ complex$(zx, zy)$
            $z \leftarrow$ complex$(0.0, 0.0)$
            $iteration \leftarrow 0$
            while $|z| \leq 2.0$ and iteration $<$ max_iterations do
                $z \leftarrow z \times z + c$
                iteration $\leftarrow$ iteration $+ 1$
            end while
            $color \leftarrow$ iteration/max_iterations
            glColor3f$(color, color, color)$
            glVertex2f$(x -$ window_width$/2.0, y -$ window_height$/2.0)$
        end for
    end for
    glEnd()
end procedure
```

- **Rendering:** OpenGL's immediate mode (`glBegin(GL_POINTS)` and `glVertex2f`) is used to render each pixel of the Mandelbrot set, with colors determined by the number of iterations.

3.3 Drawing Julia Set

- **Function `draw_julia`:** This function calculates and renders a Julia set based on a user-defined complex number c.
- **Algorithm:**

Algorithm 2 Drawing Julia Set

```
procedure DRAW_JULIA(window_width, window_height, $c$, max_iterations)
    Input:
        window_width : Width of the window in pixels
        window_height : Height of the window in pixels
        $c$ : Complex number representing the constant for the Julia set
        max_iterations : Maximum number of iterations to determine divergence
    Output:
        Renders the Julia set using OpenGL immediate mode with the given constant $c$
    Begin:
        glBegin(GL_POINTS)
    for $x \leftarrow 0$ to window_width $- 1$ do
        for $y \leftarrow 0$ to window_height $- 1$ do
            $zx \leftarrow 1.5 \times (x - \text{window\_width}/2.0)/(0.5 \times \text{zoom} \times \text{window\_width}) + \text{offset\_x}$
            $zy \leftarrow (y - \text{window\_height}/2.0)/(0.5 \times \text{zoom} \times \text{window\_height}) + \text{offset\_y}$
            $z \leftarrow \text{complex}(zx, zy)$
            iteration $\leftarrow 0$
            while $|\text{z}| \leq 2.0$ and iteration $<$ max_iterations do
                $z \leftarrow z \times z + c$
                iteration $\leftarrow$ iteration $+ 1$
            end while
            $color \leftarrow$ iteration/max_iterations
            glColor3f($color, color, color$)
            glVertex2f($x - \text{window\_width}/2.0, y - \text{window\_height}/2.0$)
        end for
    end for
    glEnd()
end procedure
```

- **Rendering:** OpenGL's immediate mode is used to render each pixel of the Julia set, with colors based on the number of iterations until divergence.

3.4 User Interaction: Zooming and Panning

- **Scroll Callback (`scroll_callback`):** Handles zooming functionality by adjusting the zoom factor (`zoom`) based on mouse scroll events. The zoom center is adjusted to zoom towards the cursor position, maintaining focus on the selected region.

3.5 Main Loop and Rendering

- **Main Loop (`main`):** The main program loop continuously clears the screen, sets up the view using OpenGL's projection and modelview matrices, and renders either the Mandelbrot or Julia set based on user selection.

- **Rendering Loop:** Renders the selected set using the respective drawing function (`draw_mandelbrot` or `draw_julia`), swaps buffers, and polls for events.

4 Results

1. **Mandelbrot Set visualization**

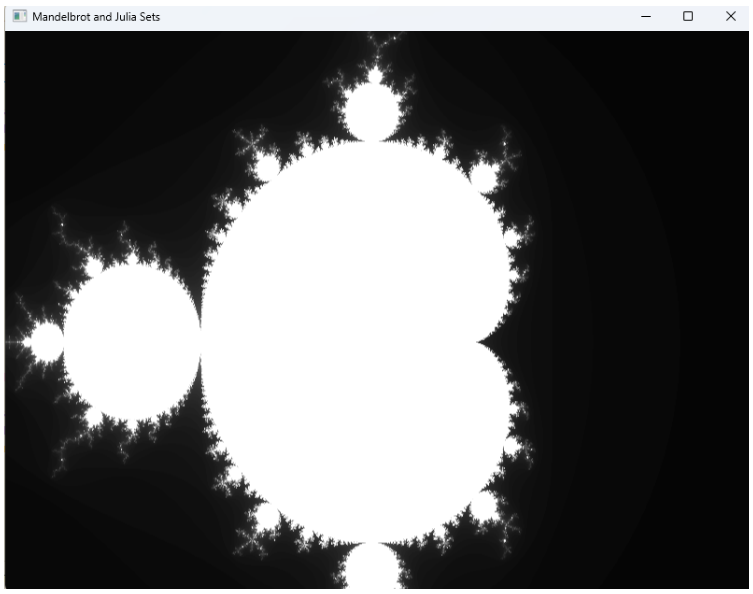

Figure 4: Visualization of the Mandelbrot Set

2. **Julia Set Visualization for c = -0.7 + 0.27015i**

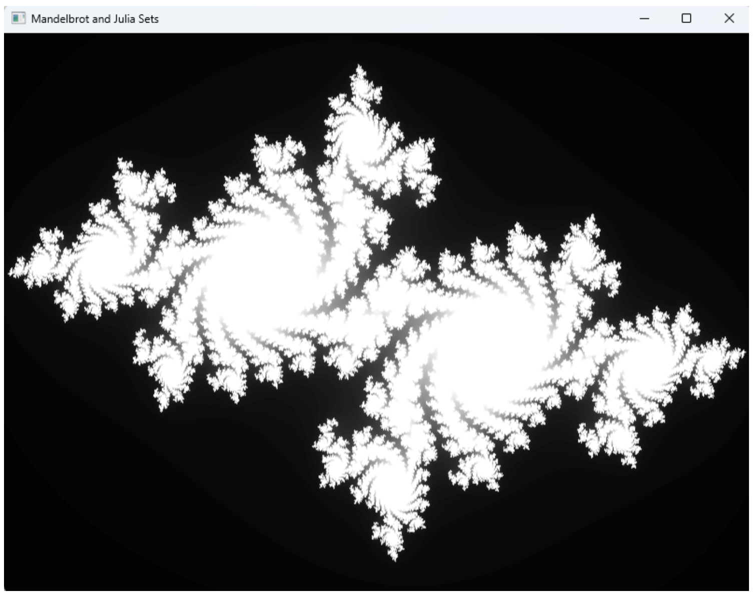

Figure 5: Visualization of Julia Set

5 Performance Analysis

5.1 Performance Issues

- **Iterative Pixel Calculation:**
  - **Issue:** Recalculating Mandelbrot or Julia sets for each pixel on every frame is computationally expensive.
  - **Impact:** High CPU usage and slow rendering, especially noticeable during zoom operations.
- **Immediate Mode Rendering (GL_POINTS):**
  - **Issue:** Using `GL_POINTS` in immediate mode for rendering can be inefficient for large datasets.
  - **Impact:** Bottleneck in rendering performance, causing lag during continuous updates.
- **Lack of Multithreading:**
  - **Issue:** Single-threaded execution leads to CPU-bound operations, affecting responsiveness.
  - **Impact:** UI freezes or lags during computational tasks like recalculating sets.
- **GPU and OpenGL Optimization:**
  - **Issue:** Suboptimal utilization of GPU capabilities and OpenGL rendering techniques.
  - **Impact:** Limits rendering throughput and responsiveness, especially for complex scenes.

5.2 Improvement Strategies

- **Algorithmic Optimization:** The calculations can be optimized by implementing caching mechanisms and adaptive sampling to reduce computational load.
- **Enhanced Rendering Techniques:** Efficiency can be improved through batch rendering using vertex buffers and shaders for GPU-based acceleration.
- **Concurrency with Multithreading:** Multithreading can be done to parallelize operations and enhance responsiveness during intensive computations.
- **Utilization of GPU Capabilities:** GPU-specific optimizations and advanced OpenGL techniques can be utilized for improved rendering performance.

6 Conclusion

In this project, we explored the visualization of Mandelbrot and Julia sets using OpenGL in C++. These fractal sets, renowned for their intricate and infinitely complex patterns, provided a fascinating exploration into computational mathematics and graphics.

The implementation leveraged GLFW for window management and OpenGL for real-time rendering, allowing us to interactively explore the fractal landscapes. We developed functions to compute and render both the Mandelbrot set, which showcases self-similar patterns at different scales, and Julia sets, characterized by their parameterized complexity.

This project successfully demonstrated the visualization of Mandelbrot and Julia sets using OpenGL in C++, showcasing the intersection of mathematics, computer graphics, and user interaction. By addressing performance optimizations and exploring advanced rendering techniques, future enhancements can further enrich the exploration and understanding of these captivating fractal structures.