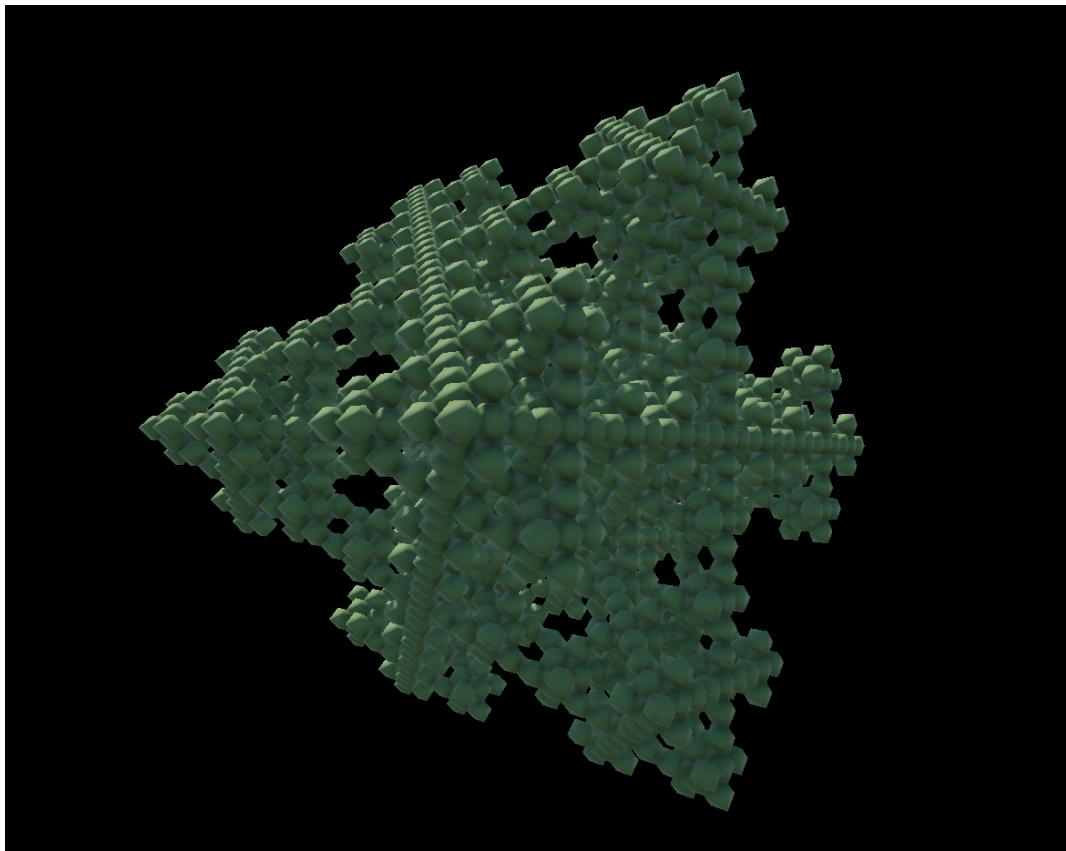


Fractal Generator in Unity

Mathematics for Games and V/AR

Msc Computer Games Programming



Yishan Ding

28/11/2025

Abstract

In this project, I attempt to develop a fractal renderer that allows users to freely generate eight different fractals and manually control parameters such as iteration depth, size, color, and rotation. The goal is to make it easier for users to understand how each fractal evolves through its iterative process. I accomplished the system using Unity and C#, and I believe I have completed the core requirements. In addition, I added optional 3D effects for several of the fractals to enhance visual understanding.

Keywords: Fractals, Unity, C#, Line Renderer, Mesh Generation

1 Features

1.1 Overview of Implemented Fractals

There are eight distinct fractals totally, mainly using FractalManager pipeline, 5 of them are build by line renderer and 3 of them are build by mesh with a 3D feature.

1.1.1 Koch Snowflake (Triangle)

Generated from an equilateral triangle, each line is replaced by same substitution rule at each iteration.

1.1.2 Koch Snowflake (Hexagon)

Generated from an equilateral hexagon, each line is replaced by same substitution rule at each iteration.

1.1.3 Sierpinski Triangle

Generated from an equilateral triangle mesh, each mesh is segmented by 3 triangles same as itself at each iteration.

1.1.4 Vicsek Fractal

Generated from a square mesh, each square is segmented by 5 squares same as itself at each iteration.

1.1.5 Hilbert Curve

The Hilbert Curve is generated through a fixed L-system. The generator expands the axiom ("L") using two substitution rules over multiple iterations, then walks through the final command string to draw the curve step by step.

1.1.6 Circle Packing Fractal

Generated from a circle, 4 smaller tangent circles are inserted between existing one at each iteration.

1.1.7 T-Square Fractal

Generated from a square mesh, 4 smaller squares are inserted between existing one at each iteration, with their anchor points placed on the corners of the base square.

1.1.8 H-Fractal

Generated from a horizontal line, adding 2 lines perpendicular to itself at the end of it which forming a "H" at each iteration.

1.2 UI Features

In designing the User Interface (UI), my primary objective was to abstract the complex mathematical parameters of fractals into an accessible format for non-technical

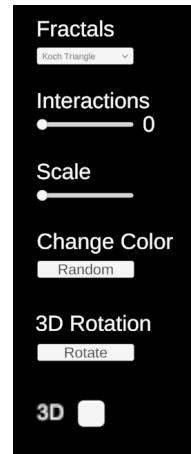


Figure 1: ui interface

users. A lightweight control panel was implemented using Unity UI, allowing real-time modification of fractal parameters. Users can adjust several parameters such as interactions slider, scale slider, rotating animation, random color button and 2D/3D switch.

1.3 configuration

The project adopts a simple external configuration system, where each fractal preset corresponds to an individual .txt file. These configuration files contain all the parameters required to reconstruct a fractal model, including: iteration, baseShape, angle, scale, offsetAngle. When the scene starts, ConfigLoader is responsible for reading the selected configuration file from the project's Resources folder. The loader parses each line, converts the text values into the correct data types, and stores them in a dedicated data class: FractalConfig. Once the configuration object is constructed, it is passed to FractalManager, which uses the parameters to generate the fractal based on the loaded preset. This system allows consistent behaviour across different fractals, easy switching between presets, and supports extension by simply adding new text files without modifying core code.

2 Data Classes

2.1 FractalConfig

FractalConfig serves as a parameter wrapper storing the fractal's configuration, such as type, initial shape, rotation angle, and position. This object is populated by ConfigLoader and later consumed by FractalManager.

2.2 IFractalCell / SquareCell / TriangleCell

Depending on the fractal, generators return either a List<Vector3> (for line-based fractals) or a List<Cell>. These simple containers allow each fractal generator to output its own data structure while keeping FractalManager's rendering pipeline unified.

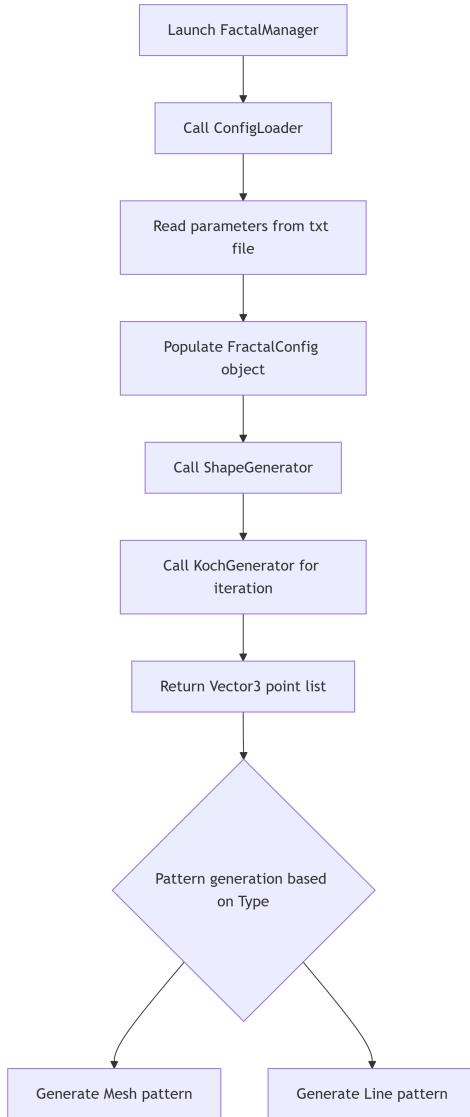


Figure 2: procedural framework

3 Main Structure

The program is organised around a central controller called FractalManager, which coordinates all fractal generation, UI updates, and rendering.

3.1 Configuration Loading

1. When the scene starts to play, ConfigLoader would read default configuration file. The configuration file includes fractal type, iteration, size, angle (only for partial fractals) and if using line renderer, mesh or 3D renderer.
2. The configuration is converted to the data class FractalConfig, and is sent to FractalManager.

3.2 Shape Initialisation

1. Uses ShapeGenerator to create the initial polygon (triangle, square, hexagon, circle seed, etc.).
2. Stores this initial shape as the base shape for all later iterations.

3.3 Selecting a Generator

1. Users select fractal type through UI Dropdown.
2. When selection is done, FractalUIController invoke the function FractalManager.GenerateByType.
3. FractalManager select corresponded generator based on enum:
SierpinskiGenerator
GosperGenerator
KochGenerator
HilbertGenerator
CircleGenerator
HGenerator
VicsekGenerator
TSquareGenerator
4. Each generator receives: the current list of points or cells, the requested iteration count and the required parameters (size, angle offset). Then the generator return either a list of points or a list of Cell data classes.

3.4 Iteration Processing

1. Each time the iteration value changes, FractalManager calls the selected generator again. The resulting data structure is sent to the rendering pipeline.

3.5 Rendering

FractalManager does it together, The rendering pipeline supports three modes:

1. LineRenderer is instantiated, and points are connected sequentially.

2. A polygon mesh is generated from the returned points or cell meshes.
3. 3D generators output CubeCell objects and each cube mesh is instantiated with a shared material.

3.6 UI Integration

1. A dropdown is linked to SetPresetByIndex(). Changing the value loads the corresponding configuration file and regenerates the fractal. Iteration and size slider is bound to public fields in FractalManager.
2. Hotkeys of numbers 1–8 switch between the eight fractals. Up/Down arrows change iteration. Left/Right arrows change size. Middle mouse drag rotates the entire fractal for 3D inspection.

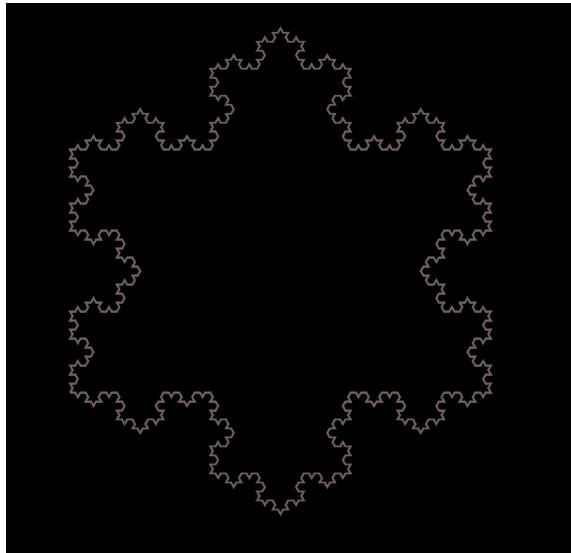
4 Results

The project successfully renders eight different fractals using line rendering, mesh-based filled shapes, and 3D visualization for selected patterns. Each fractal's iterative behavior has been tested and verified against standard mathematical definitions. For example, the Koch, Sierpinski, and Hilbert curves match their expected shapes, and the subdivision rules of the Vicsek and T-Square fractals behave correctly.

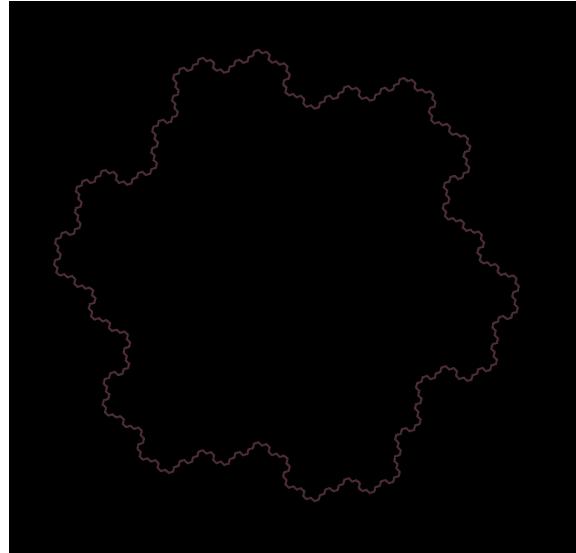
All fractals can be adjusted in real time through the user interface, including iteration level, size, rotation, and colour. Hotkeys also allow users to switch between fractals or modify parameters efficiently. Some fractals, such as Vicsek and T-Square, additionally support a 3D mode to demonstrate the structure in an extra dimension.

One fractal differs slightly from the reference image: the Circle Packing pattern correctly generates recursive circles, but the smaller circles do not visually occlude the outline of the larger circle. This affects only the appearance, not the underlying iteration logic.

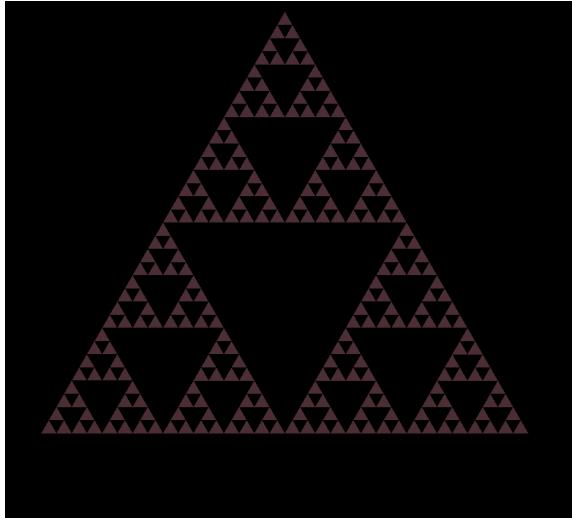
Overall, the system successfully demonstrates eight fractal structures with interactive controls and a clear visualization of the iterative processes.



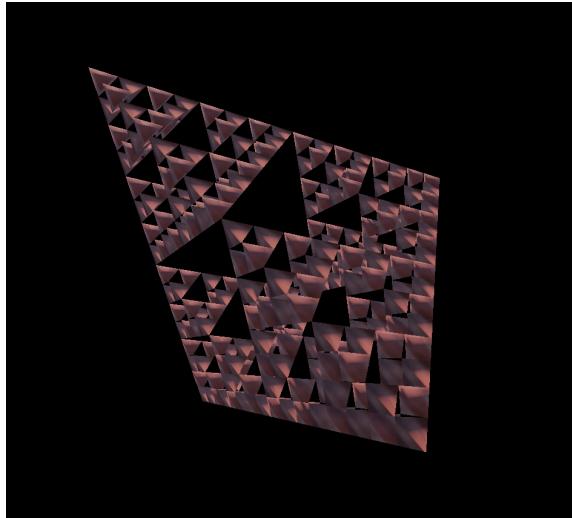
(a) Koch Snowflake (Triangle)



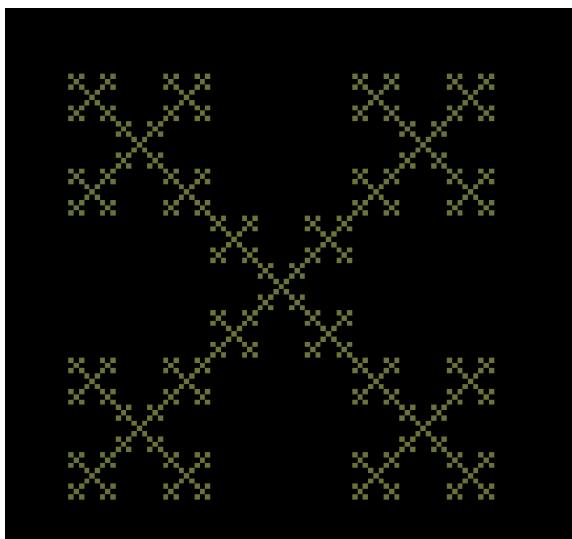
(b) Koch Snowflake (Hexagon)



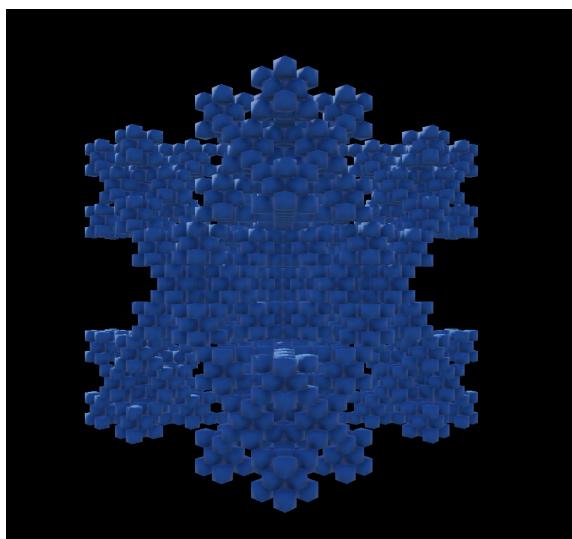
(a) Sierpinski Triangle



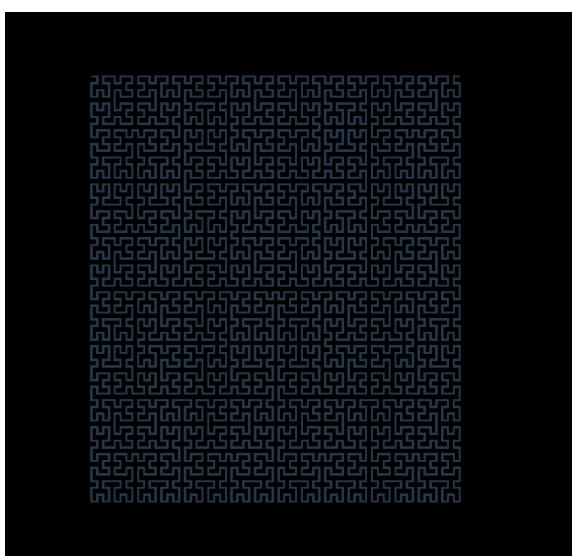
(b) Sierpinski Triangle (3D)



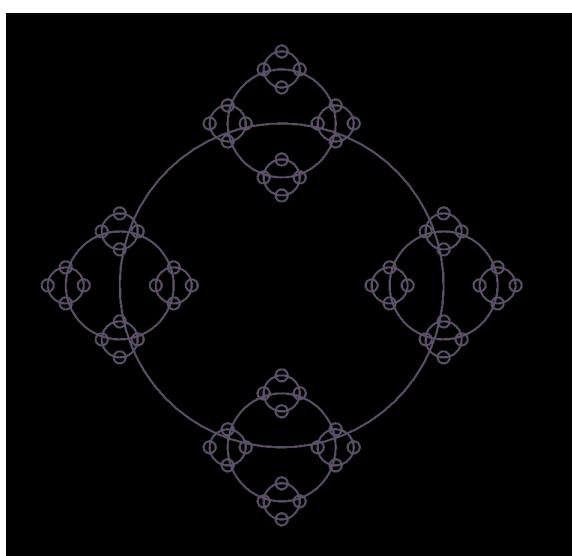
(c) Vicsek Fractal



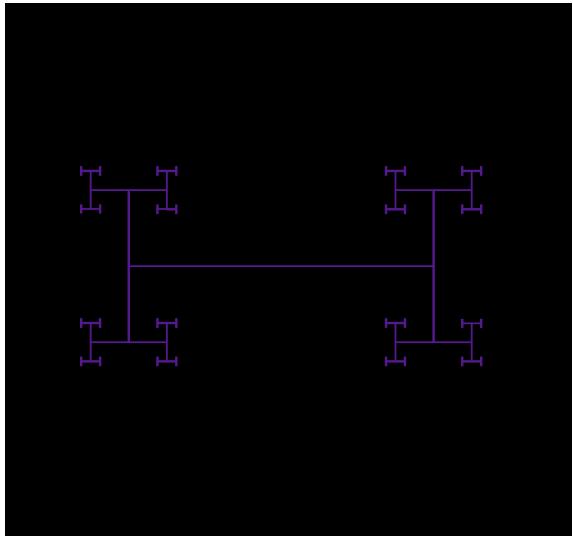
(d) Vicsek Fractal (3D)



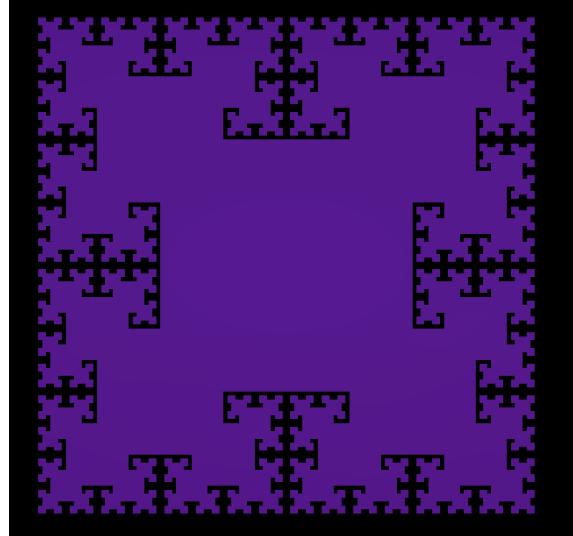
(e) Hilbert Curve



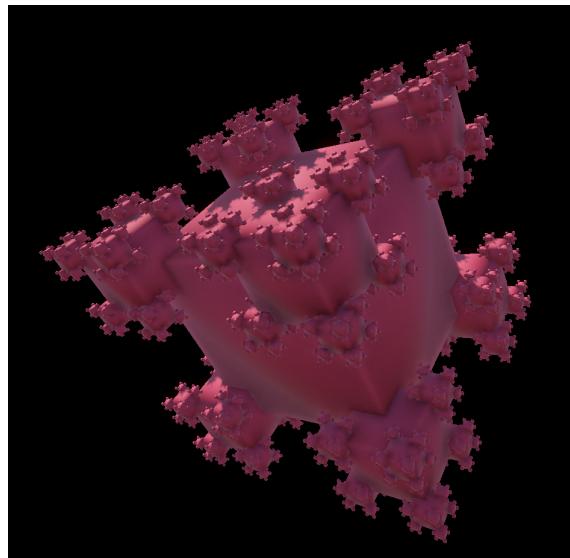
(f) Circle Packing Fractal



(a) H-Fractal



(b) T-Square Fractal



(c) T-Square Fractal (3D)

5 Conclusion

In conclusion, I have accomplished the core objectives in this project. However, several aspects could be improved or expanded upon.

First, the current system relies on geometric subdivision rules, where each interaction replaces a line segment or cells with a predefined pattern. While this method is intuitive, many fractals- such as Mandelbrot set- are better generated using mathematical iteration based on complex dynamics. In addition, Hilbert Curve is generated through a fixed L-system, with the axiom and rules defined inside the generator. The main advantages of L-systems are their high scalability and unified representation.

Second, regarding rendering, I mainly use Unity's LineRenderer and MeshRenderer[1]. For future expansion, shader-based pixel iteration or GPU compute approaches could be added to support

continuous fractals rather than purely geometric ones. shader-based pixel iteration or GPU compute approaches could be added to support continuous fractals rather than purely geometric ones. Furthermore, some fractals could be generated by GPU Instancing. This method is particularly effective for rendering large amounts of repetitive structures, as it significantly reduces Draw Calls and improves overall rendering efficiency.

Finally, while the system includes a 3D mode for the Vicsek fractal, many 2D fractals could also be extended into 3D. For example, the Hilbert curve has a well-established 3D version commonly used in spatial indexing. Adding such variations would significantly enhance the visual complexity and educational value of the system.

Overall, I believe I have achieved my main objective of creating an interactive, multi-mode fractal generator. This project has deepened my understanding of recursion, geometric subdivision, parametric systems, and Unity's rendering pipeline.

References

- [1] Benny Hansen Lifindra and Rianarto Sarno. Visualizing texture of fractal arts in unity3d framework. In *2023 International Conference on Computer Science, Information Technology and Engineering (ICCoSITE)*, pages 6–11, 2023.