Interactive 2D IFS Fractal Generator

Team 05

SAUMIK SHASHWAT

1 INTRODUCTION

Nature is full of intricate yet orderly details, for example, trees, clouds, and mountains. In computer graphics, geometric modelling of complex structures is very intensive [4], which is where fractals come in handy, as they are a convenient way to represent those intricacies in a compressed manner [2][5].

Iterated function systems, or IFSs, were introduced in 1981 by John E. Hutchinson and popularised by the book 'Fractals Everywhere' by Michael Barnsley. IFSs are one of the many methods to generate fractals; they can be of any dimension and are often computed and drawn in 2D [1].

In this project, I aim to develop an interactive 2D IFS fractal generator which enables interactive and iterative affine transformations of generators (a series of rectangles) to generate orderly n-iterative fractals. The generators can be created and linearly transformed (translated, scaled, rotated) to create fractals in multiple iterations.

2 LITERATURE REVIEW

The Macintosh IFS Manual, written by Paul Bourke [3], provides a rough overview of the project, orderly explaining how the program is supposed to work, with comprehensive visuals showcasing each iteration for the fractal generation from the generator rectangles. The manual initially describes a way of describing affine transformations for IFS. After explaining the flow of the program, it highlights some of the applications of this project, mainly data reduction for complex geometric model files.

The emphasis on the data reduction application for this project is further articulated in the excerpt from "A Better Way to Compress Images" by Michael F. Barnsley and Alan D. Sloan. [2], where image compression takes a central role in designing their IFS fractal generator application. The authors explain iterated function systems and then discuss the Random Iteration Algorithm, which decodes an arbitrary IFS code. Lastly, they discuss the Collage Theorem, a systematic method for finding the affine transformations that produce the IFS encoding of an image.

In "An Interactive Application for Modeling Two-Dimensional IFS Fractals" [5], Elena Hadzieva and Jovan Petkoski discuss the related work done in this field, including some of the other papers I have discussed in this section. They further provide a mathematical description of their algorithm, in which they discuss the representation of the points that constitute the fractal image by barycentric coordinates with respect to the three pointwise independent points. Their implementation of the fractal generator is written in C programming language in the Microsoft Visual Studio IDE.

In "Construction of Fractal Objects with Iterated Function Systems" [4], Stephen Demko, Laurie Hodges and Bruce Naylor discuss the previously used techniques for modelling natural objects and their deficiencies, revolving around texture techniques such as colour modulation, geometric techniques, and old fractal-based methods based on fractional Brownian motion. Then they discuss IFS by explaining the Cantor Set as an example and talk about the constructive and qualitative aspects of Attractors, followed by its computation.

Returning to the Macintosh IFS manual [3], the core literature for this project, let's delve deep into the flow of the program. The main idea behind the process is to replace one set of rectangles with another series of suitably transformed rectangles (the generator set) with each iteration. An example is shown below, with Fig. 1. being of the generator set of rectangles and Fig. 2. being of the result four iterations later.

Author's address: Saumik Shashwat, saumik20404@iiitd.ac.in.

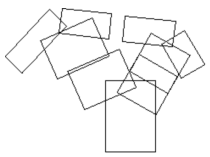


Fig. 1. Generator Set

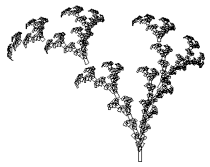
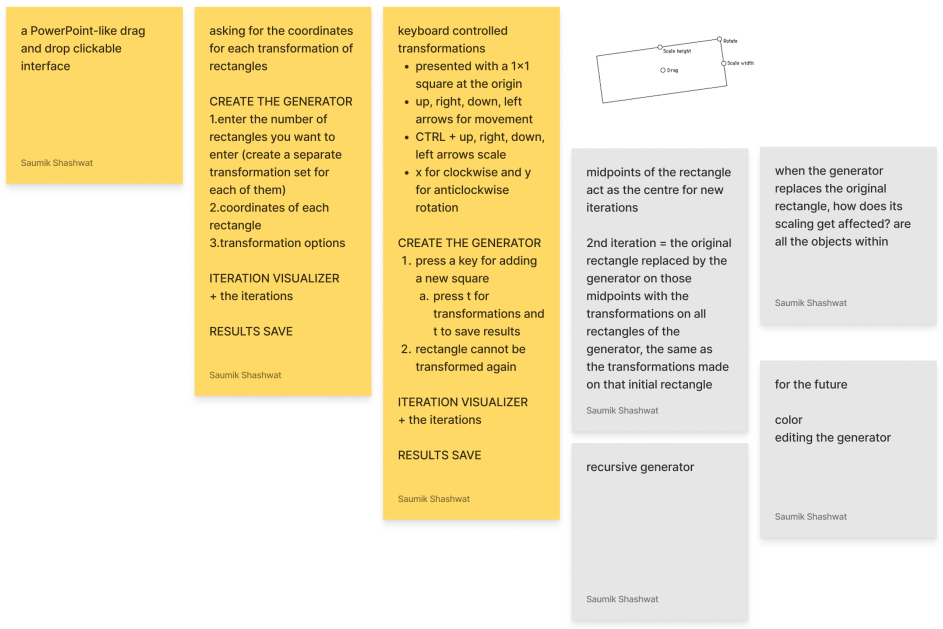


Fig. 2. After fourth iteration



a PowerPoint-like drag and drop clickable interface

Saumik Shashwat

asking for the coordinates for each transformation of rectangles

CREATE THE GENERATOR
1.enter the number of rectangles you want to enter (create a separate transformation set for each of them)
2.coordinates of each rectangle
3.transformation options

ITERATION VISUALIZER
+ the iterations

RESULTS SAVE

Saumik Shashwat

keyboard controlled transformations

- presented with a 1×1 square at the origin
- up, right, down, left arrows for movement
- CTRL + up, right, down, left arrows scale
- x for clockwise and y for anticlockwise rotation

CREATE THE GENERATOR

1. press a key for adding a new square
    a. press t for transformations and t to save results
2. rectangle cannot be transformed again

ITERATION VISUALIZER
+ the iterations

RESULTS SAVE

Saumik Shashwat

midpoints of the rectangle act as the centre for new iterations

2nd iteration = the original rectangle replaced by the generator on those midpoints with the transformations on all rectangles of the generator, the same as the transformations made on that initial rectangle

Saumik Shashwat

when the generator replaces the original rectangle, how does its scaling get affected? are all the objects within

Saumik Shashwat

recursive generator

Saumik Shashwat

for the future

color
editing the generator

Saumik Shashwat

Fig. 3. Rough Ideation and Strategies for the implementation

3 BASIC FLOW

1. The generator set for the fractal will need to be defined first. This will generally involve the transformation matrices and probability weights for each of the rectangles in the generator set being defined.

2. A function for generating the points for the fractal will need to be created next. This can be done using the Iterated Function System (IFS) algorithm, which involves randomly selecting one of the rectangles in the generator set, having the corresponding transformation matrix applied to a point, and then repeating this process for a large number of iterations.

3. In order to render the fractal using OpenGL, a vertex array containing the points generated by the IFS function will need to be created. This vertex array can then be used to create a vertex buffer object (VBO) and the fractal can be rendered using a simple OpenGL shader program.

4. To create a user interface for the fractal generator, an OpenGL window will need to be set up and user input events such as mouse clicks and key presses will need to be handled. This user input can be used to modify the generator set or other parameters of the fractal in real-time.

4 CODE SNIPPETS

struct for a generic rectangle:

```
struct Rectangle {
    glm::vec2 center;
    glm::vec2 size;
    float angle;
};
```

fractal generator class:

```
class FractalGenerator {
public:
    FractalGenerator() {
        rectangles.resize(5);
    }
    void addRectangle(const Rectangle& rect) {
        rectangles.push_back(rect);
    }
    void removeRectangle(int index) {
        rectangles.erase(rectangles.begin() + index);
    }
    void setRectangle(int index, const Rectangle& rect) {
        rectangles[index] = rect;
    }
    std::vector<glm::vec2> generateFractal(int numIterations) const {
        std::vector<glm::vec2> points;
        for (int i = 0; i < numIterations; i++) {
            for (const auto& rect : rectangles) {
                glm::mat2x2 transform;
                transform = glm::rotate(transform, rect.angle);
                transform = glm::scale(transform, glm::vec2(rect.size));
                transform = glm::translate(transform, rect.center);
                for (auto& point : points) {
                    point = transform * point;
                }
            }
        }
        return points;
    }
private:
    std::vector<Rectangle> rectangles;
};
```

5 MILESTONES COMPLETED

S. No.	Milestone	Member
	Mid evaluation	
1	Understanding the fundamental theories and concepts involved in the development process through extensive literature survey and experimentation.	Saumik Shashwat
2	Designing the information architecture flowchart for the application.	Saumik Shashwat
3	Developing an interactive UI to create and transform (translate, scale, rotate) the generator rectangles.	Saumik Shashwat
	Final evaluation	
4	Adding other customisations for the generators, such as adding colours.	Saumik Shashwat
5	Further developing the iterative interface, allowing users to create n-iterative fractals and save the results.	Saumik Shashwat
6	Finding and solving the corner cases that may arise in the development process	Saumik Shashwat
7	Thoroughly debugging the codebase.	Saumik Shashwat

REFERENCES

[1] 2022. Iterated function system. https://en.wikipedia.org/w/index.php?title=Iterated_function_system&oldid=1114991005 Page Version ID: 1114991005.

[2] Michael F Barnsley and Alan D Sloan. 1988. A Better Way to Compress Images. (Jan. 1988). https://www.cs.cmu.edu/~christos/courses/826-resources/PAPERS+BOOK/CMU-ONLY/ImageCompression.pdf

[3] Paul Bourke. 1990. IFS manual. http://paulbourke.net/fractals/ifs/

[4] Stephen Demko, Laurie Hodges, and Bruce Naylor. 1985. Construction of fractal objects with iterated function systems. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques (SIGGRAPH '85)*. Association for Computing Machinery, New York, NY, USA, 271–278. https://doi.org/10.1145/325334.325245

[5] Elena Hadzieva and Jovan Petkoski. 2015. An Interactive Application for Modeling Two-Dimensional IFS Fractals. In *2015 Second International Conference on Mathematics and Computers in Sciences and in Industry (MCSI)*. 49–54. https://doi.org/10.1109/MCSI.2015.21