PS2: Triangle Fractal

Assignment Overview

This project involved creating a recursive graphical application using SFML to display a triangle-based fractal, inspired by the Sierpiński triangle. The program takes two command-line arguments: the side length of the base triangle (L) and the recursion depth (N). A recursive function was used to draw and position each level of the fractal pattern, and the window was dynamically sized to match the geometry. Extra credit was pursued by adding color and translucency variation based on recursion depth. It is important to add that this program was completed using paired programming practices to stimulate an industry-like situation.

Key Algorithms / Data Structures / OO Designs

The program uses a recursive drawing algorithm where each triangle generates three smaller child triangles at every level, with their size halved and positions calculated relative to the parent triangle. SFML's sf::ConvexShape is used to construct and render each triangle, allowing for dynamic manipulation of points and fill color. Geometric calculations based on the height formula for equilateral triangles (height = root3/2 × side) are used to precisely place each new triangle so that the overall fractal structure maintains its symmetry. Admittedly, the geometry behind our solution was very much achieved through trial and error. It proved extremely challenging to get our child triangles to be placed exactly where we wanted them. The program also accepts command-line input for triangle size and recursion depth, enabling flexible execution. Additionally, the fill color of each triangle is modulated according to recursion depth, creating a visually pleasing and layered pattern that satisfies extra credit requirements for aesthetic enhancement.

What I Learned

Through PS2, I learned how to apply recursive thinking to graphical problems and use SFML graphics and shapes (like ConvexShape) to draw 2D geometric figures. I calculated positions of equilateral triangles using trigonometric relationships and vector math, and dynamically sized and rendered a window to fit complex, scaled content. I also implemented color schemes using logic tied to depth or position and structured a program around command-line arguments for custom program behavior. Most importantly, I learned how to work effectively with a programming partner using paired programming, where we alternated physically coding and debugging positions. This collaborative method allowed us to avoid bugs that would have taken significantly longer to resolve individually.

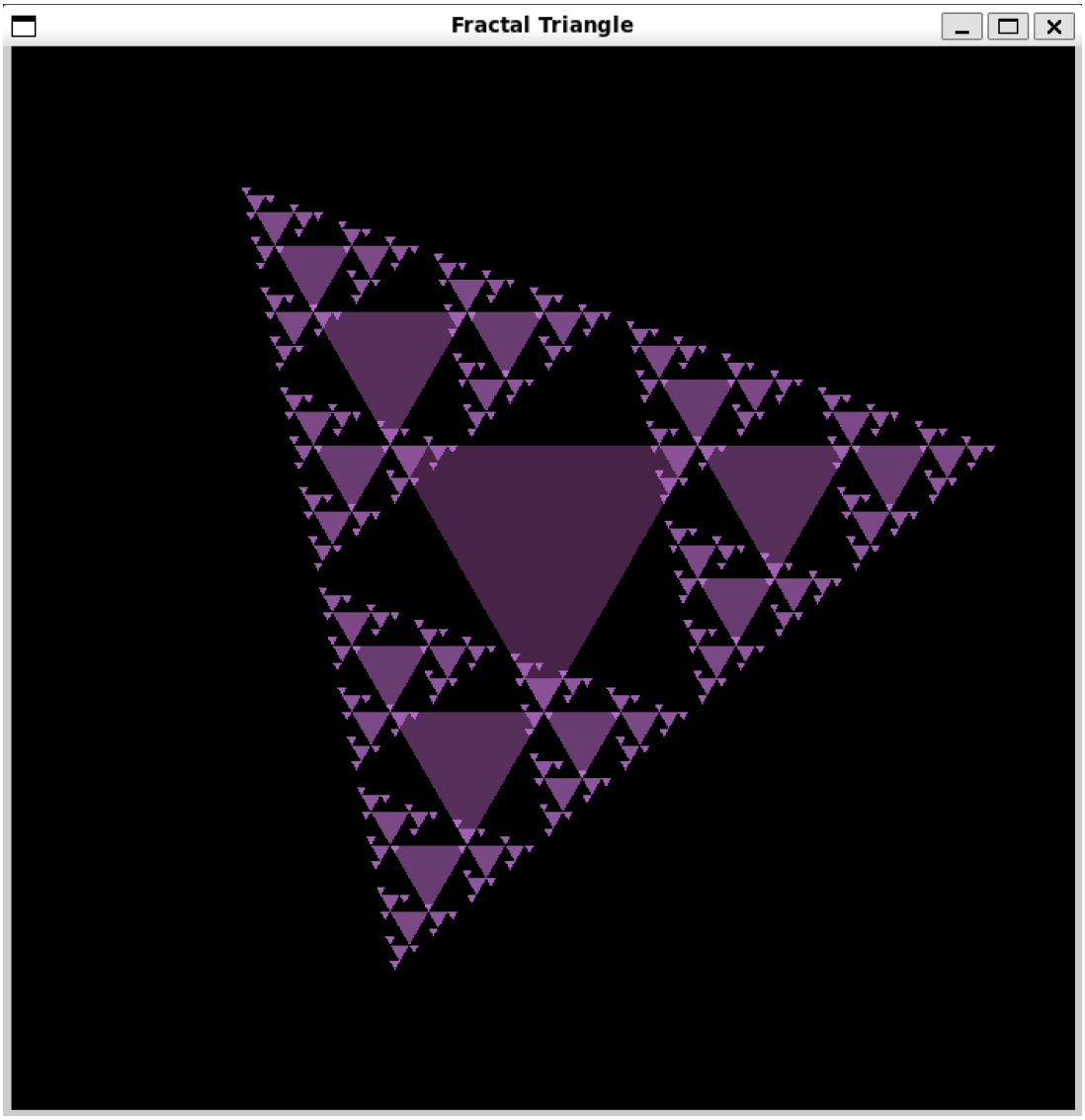

Figure 3: Example Output

Source Code

```
Makefile:
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
# Your .hpp files
DEPS = triangle.hpp
# Your compiled .o files
OBJECTS = triangle.o
# The name of your program
PROGRAM = Triangle

.PHONY: all clean lint

```

```
all: $(PROGRAM)

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $<
$(PROGRAM): main.o $(OBJECTS)
	$(CC) $(CFLAGS) -o $@ $^ $(LIB)
clean:
	rm *.o $(PROGRAM)
lint:
	cpplint *.cpp *.hpp

main.cpp:
// Copyright 2025 Karan Patel & Kaden Gardiner

#include <iostream>
#include <cmath>
#include <SFML/Graphics.hpp>
#include "triangle.hpp"

int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cout << "Usage: ./Triangle L N" << std::endl;
        return 1;
    }

    float L = std::stof(argv[1]);
    int N = std::stoi(argv[2]);

    float width = (sqrt(3) * 2) * L;
    float height = (sqrt(3) * 2) * L;

    sf::RenderWindow window(sf::VideoMode(width, height), "Fractal
       Triangle");
    sf::Vector2f startPoint(width/ 2, height / 2);

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        window.clear(sf::Color::Black);

        fractal(window, N, L, startPoint, 0);

        window.display();
    }

    return 0;
}
```

```

triangle.hpp:
// Copyright 2025 Karan Patel & Kaden Gardiner
#pragma once

#include <SFML/Graphics.hpp>

sf::ConvexShape drawTriangle(sf::RenderTarget& window,
                             float side, sf::Vector2f point);
void fractal(sf::RenderTarget& window, int n, float side,
             sf::Vector2f point, int depth);

triangle.cpp:
// Copyright 2025 Karan Patel & Kaden Gardiner

#include <iostream>
#include <cmath>
#include "triangle.hpp"

sf::ConvexShape drawTriangle(sf::RenderTarget& window,
                             float side, sf::Vector2f point) {
    sf::ConvexShape triangle(3);
    float h = sqrt(3) / 2 * side;
    triangle.setPoint(0, sf::Vector2f(point.x, point.y + h / 2));
    triangle.setPoint(1, sf::Vector2f(point.x - side / 2, point.y - h / 2)
       );
    triangle.setPoint(2, sf::Vector2f(point.x + side / 2, point.y - h / 2)
       );

    return triangle;
}

void fractal(sf::RenderTarget& window, int n, float side,
             sf::Vector2f point, int depth) {
    if (n == 0) {
        sf::ConvexShape triangle(3);
        float h = sqrt(3) / 2 * side;
        triangle.setPoint(0, sf::Vector2f(point.x, point.y + h / 2));
        triangle.setPoint(1, sf::Vector2f(point.x - side / 2, point.y - h
           / 2));
        triangle.setPoint(2, sf::Vector2f(point.x + side / 2, point.y - h
           / 2));

        int r = 100 + (depth * 20) % 156;
        int g = 50 + (depth * 15) % 206;
        int b = 100 + (depth * 25) % 156;
        int alpha = 180 + (depth * 5) % 76;

        triangle.setFillColor(sf::Color(r, g, b, alpha));

        window.draw(triangle);
        return;
```

```
113     }
114
115     sf::ConvexShape triangle(3);
116     float h = sqrt(3) / 2 * side;
117     triangle.setPoint(0, sf::Vector2f(point.x, point.y + h / 2));
118     triangle.setPoint(1, sf::Vector2f(point.x - side / 2, point.y - h / 2)
            );
119     triangle.setPoint(2, sf::Vector2f(point.x + side / 2, point.y - h / 2)
            );
120
121     int r = 100 + (depth * 20) % 156;
122     int g = 50 + (depth * 15) % 206;
123     int b = 100 + (depth * 25) % 156;
124     int alpha = 180 + (depth * 5) % 76;
125
126     triangle.setFillColor(sf::Color(r, g, b, alpha));
127
128     window.draw(triangle);
129
130     float newSide = side / 2;
131     float newH = sqrt(3) / 2 * newSide;
132
133     sf::Vector2f bottom = {point.x - newSide / 2, point.y + h / 2 + newH /
            2};
134     sf::Vector2f left = {point.x - side / 2, (point.y + h / 2
135                         - (newH + newH + newH / 2))};
136     sf::Vector2f right = {(point.x + (side) * 3/4), point.y - h/4};
137
138     fractal(window, n - 1, newSide, bottom, depth + 1);
139     fractal(window, n - 1, newSide, left, depth + 1);
140     fractal(window, n - 1, newSide, right, depth + 1);
141 }
```