PS3 Recursive Graphics (Triangle Fractal)

For this assignment, we were given the task of implementing the Sierpinski triangle project from Princeton. The assignment's main objective was to utilize recursion to create a complex-looking triangle using only a few lines of code. The main program took an integer, N, and used it to change the depth of the recursion. After that, our software would recursively draw triangles within triangles, drawing one triangle at depth 1, four triangles at depth 2, and so on.

Key concepts

Recursion is literally the name of the game here within this project. It allows the triangles to be printed to pascals form without manual input or more tedious loops. Vectors are also a godsend when it comes to memory management, which can be a nightmare if the changing of an objects "size" is needed or leaks are allowed to happen.

What I Accomplished

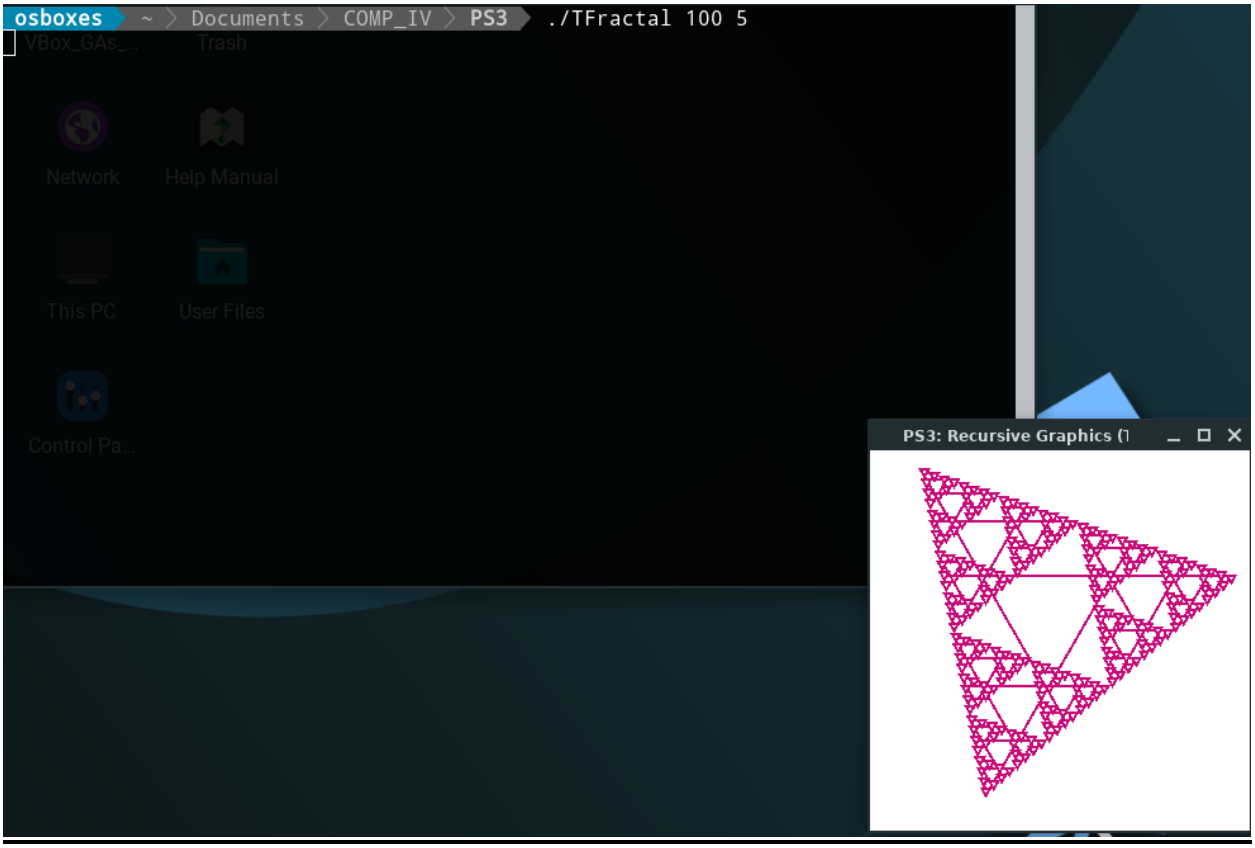
The project contains an object that holds all the information for each individual triangle. Main.cpp then uses a recursive function to print them All out in a pascal triangle. This function is pretty much the whole project. It sets the size, thickness, and color of the triangles to be printed by main.

```
Triangle::Triangle(double initX, double initY, double initL):
      x(initX), y(initY), l(initL) {
   double i = sqrt(.75 * pow(l, 2));
   triangle.setPointCount(3);
   triangle.setPoint(0, Vector2f(static_cast<float>(0),
      static_cast<float>(0)));
   triangle.setPoint(1, Vector2f(static_cast<float>(l),
      static_cast<float>(0)));
   triangle.setPoint(2, Vector2f(static_cast<float>(l/2),
   static_cast<float>(i)));
   sf::Color color((time(0) * 5 + offset1) % 256,
      (time(0) * 5 + offset2) % 256, (time(0) * 5 + offset3) % 256);
   triangle.setOutlineColor(color);
   triangle.setOutlineThickness(2);
   triangle.setPosition(x, y);
```

What I Learned

That recursion is a really powerful tool, but can be extremely taxing if not implemented properly. The memory management can easily get out of hand with a slight miscalculation

Output


osboxes ~ Documents COMP_IV PS3 ./TFractal 100 5
Trash
Network
Help Manual
This PC
User Files
Control Pa...
PS3: Recursive Graphics (1

```
 1: C= g++
 2: CFLAGS= -Wall -Werror -ansi -pedantic
 3: GFLAGS= -lsfml-graphics -lsfml-window -lsfml-system
 4: c17 = -std=c++17
 5:
 6: all: TFractal
 7:
 8: TFractal: TFractal.o Triangle.o
 9:         $(C) TFractal.o Triangle.o -o TFractal $(GFLAGS)
10:
11: TFractal.o: TFractal.cpp
12:         $(C) -c TFractal.cpp -o TFractal.o $(CFLAGS) $(c17)
13:
14: Triangle.o: Triangle.cpp Triangle.hpp
15:         $(C) -c Triangle.cpp -o Triangle.o $(CFLAGS) $(c17)
16:
17: clean:
18:         rm *.o *TFractal
```

```
    1: #include <iostream>
    2: #include <SFML/Graphics.hpp>
    3: #include "Triangle.hpp"
    4:
    5: using std::cout;
    6: using std::endl;
    7: using std::stod;
    8: using sf::RenderWindow;
    9: using std::stoi;
   10:
   11: void triangleFractal(int i, RenderWindow* window,
   12:     double x, double y, double l) {
   13:
   14:     Triangle triangle(x, y, l);
   15:     window->draw(triangle);
   16:     if (i > 0) {
   17:         triangleFractal(i - 1, window, x - (l / 4),
   18:             y - sqrt((3.0/16) * pow(l, 2)), l/2);
   19:         triangleFractal(i - 1, window, x + l, y, l/2);
   20:         triangleFractal(i - 1, window, x, y + sqrt(.75 * pow(l, 2)), l/2)
;
   21:     }
   22:     return;
   23: }
   24:
   25: int main(int argc, char* argv[]) {
   26:     // sets up command line arguments
   27:     if (argc != 3) {
   28:         cout << "Incorrect number of inputs." << endl;
   29:         exit(1);
   30:     }
   31:     int N = stoi(argv[2]);
   32:     double L = stod(argv[1]);
   33:     if (L <= 0 || N <= 0) {
   34:         cout << "Incorrect input range." << endl;
   35:     }
   36:
   37:     RenderWindow window(sf::VideoMode(L * 3, L * 3), "PS3: Recursive Grap
hics (Triangle Fractal)");
   38:     // loop to check if closed
   39:     while (window.isOpen()) {
   40:         sf::Event event;
   41:         while (window.pollEvent(event)) {
   42:             if (event.type == sf::Event::Closed)
   43:                 window.close();
   44:         }
   45:
   46:         // fractal triangle setup
   47:         window.clear(sf::Color::White);
   48:         triangleFractal(N, &window, L * (9.0/10), L, L);
   49:
   50:         window.display();
   51:     }
   52:     return 0;
   53: }
```

```
    1: #ifndef TRIANGLE_HPP
    2: #define TRIANGLE_HPP
    3:
    4: #include <cmath>
    5: #include <SFML/Graphics.hpp>
    6:
    7: using sf::ConvexShape;
    8: using sf::Vector2f;
    9:
   10: class Triangle : public sf::Drawable{
   11:  public:
   12:     Triangle(double initX, double initY, double initL);
   13:  private:
   14:     ConvexShape triangle;
   15:     double x, y;
   16:     double l;
   17:     virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
const;
   18: };
   19:
   20: #endif
```

```
    1: #include "Triangle.hpp"
    2: #include <ctime>
    3: #include <random>
    4:
    5: std::random_device rd;
    6: std::mt19937 mt(rd());
    7: std::uniform_int_distribution<int> dist(0, 255);
    8:
    9: int offset1 = dist(mt);
   10: int offset2 = dist(mt);
   11: int offset3 = dist(mt);
   12:
   13: void Triangle::draw(sf::RenderTarget& target, sf::RenderStates states) co
nst {
   14:     target.draw(triangle, states);
   15: }
   16:
   17: Triangle::Triangle(double initX, double initY, double initL):
   18:         x(initX), y(initY), l(initL) {
   19:     double i = sqrt(.75 * pow(l, 2));
   20:     triangle.setPointCount(3);
   21:     triangle.setPoint(0, Vector2f(static_cast<float>(0),
   22:         static_cast<float>(0)));
   23:     triangle.setPoint(1, Vector2f(static_cast<float>(l),
   24:         static_cast<float>(0)));
   25:     triangle.setPoint(2, Vector2f(static_cast<float>(l/2),
   26:     static_cast<float>(i)));
   27:     sf::Color color((time(0) * 5 + offset1) % 256,
   28:         (time(0) * 5 + offset2) % 256, (time(0) * 5 + offset3) % 256);
   29:     triangle.setOutlineColor(color);
   30:     triangle.setOutlineThickness(2);
   31:     triangle.setPosition(x, y);
   32: }
```