##### **Project 1: Sierpinski Gasket**

Caleb Klinger, Kyungchan Im (Chris)

Grand Canyon University

CST - 435 : Computer Graphics Lecture & Lab

Ricardo Citro

Due September 13,2023

**Description**

This project aims to visualize mathematical gaskets using OpenGL. A gasket is a fractal, a complex structure built from the repetition of simple patterns at every scale. In this project, we focus on two types of gaskets: the 2D Sierpinski Triangle and the 3D Sierpinski Tetrahedron. The purpose of the project is to understand the functionality of the code and scrutinizing the OpenGL libraries.

**2D Sierpinski Gasket**

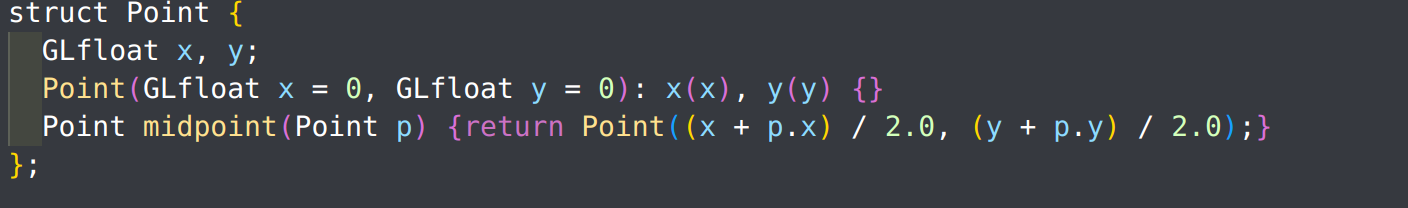
The uses OpenGL and GLUT to draw the Sierpinski triangle, a fractal that's constructed by recursively subdividing an equilateral triangle into smaller equilateral triangles.

Line 3 - 8: In the codes header



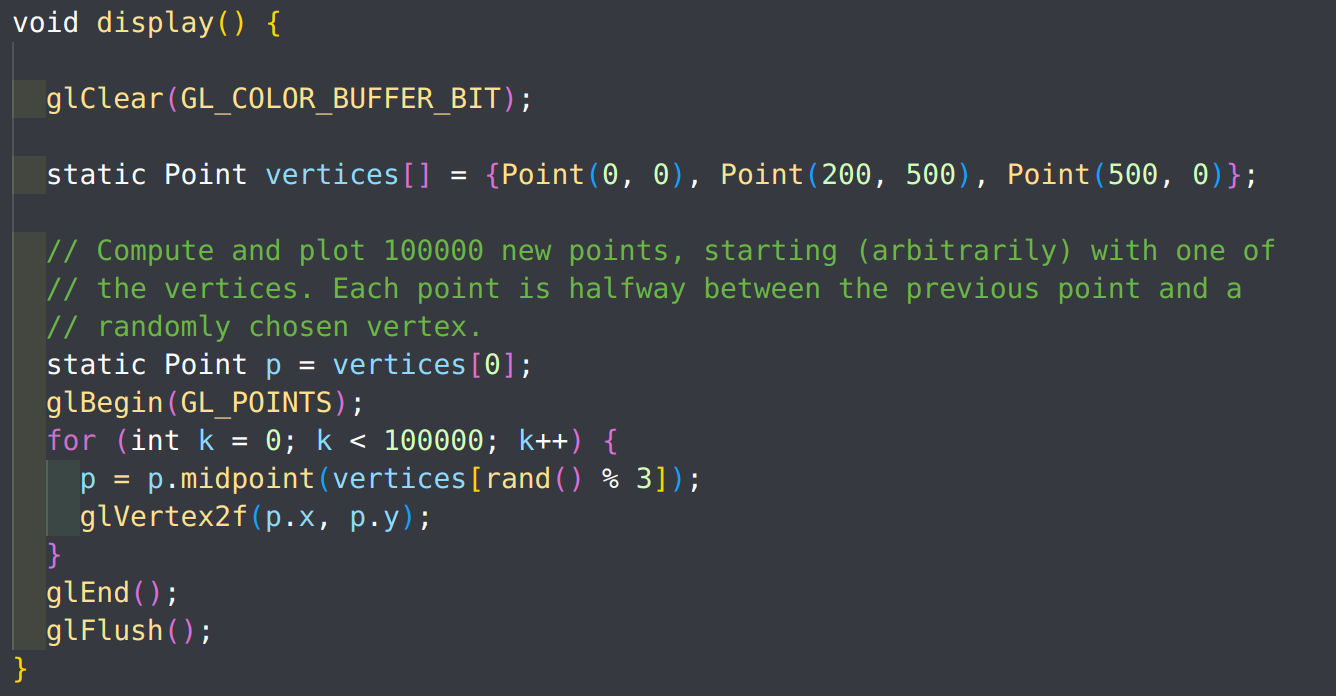
This code checks if the program is being compiled on an Apple platform. If it is, it includes the GLUT header specific to Apple. Otherwise, it includes the standard GLUT header. cstdlib is included for the rand() function.

Line 13 - 17



This is a simple struct to represent a 2D point with x and y coordinates. It also has a function midpoint that calculates the midpoint between the current point and another point p.

Line 22 - 39:



This function is the core of the program. It's responsible for drawing the Sierpinski triangle.

glClear(GL\_COLOR\_BUFFER\_BIT);: Clears the screen.

static Point vertices[] = {...};: Defines the three vertices of the main triangle.

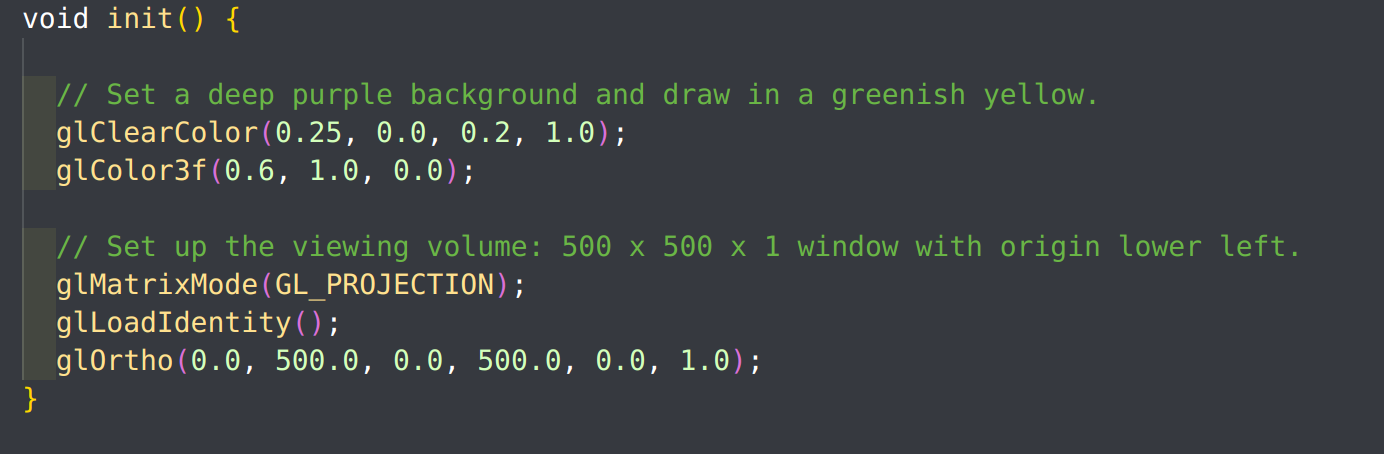
The loop that follows generates 100,000 points. For each point, it calculates the midpoint between the current point and a randomly chosen vertex of the triangle. This is the chaos game method of generating the Sierpinski triangle.

glBegin(GL\_POINTS); and glEnd();: These functions enclose the drawing commands. In this case, the program is drawing points.

glVertex2f(p.x, p.y);: Sets the position of the current vertex in 2D space.

glFlush();: Ensures that all OpenGL commands are executed.

Line 43 - 53



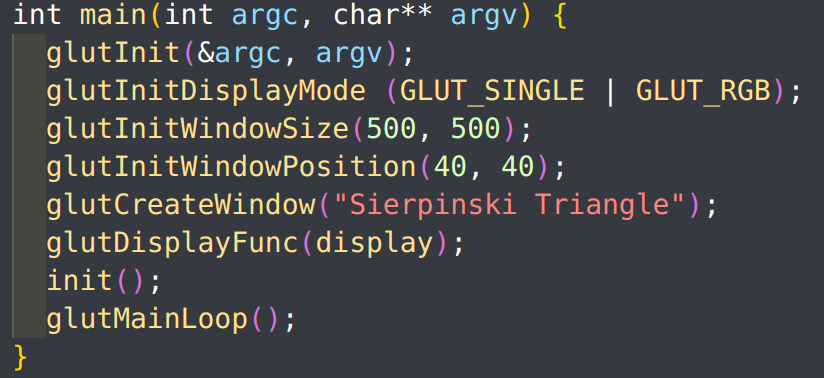
This function sets up the initial OpenGL state:

glClearColor(...);: Sets the background color.

glColor3f(...);: Sets the color for drawing.

The following commands set up an orthographic projection, which is a way of representing 3D objects in 2D.

Line 57 - 66



This function initializes GLUT and sets up the main window:

glutInit(&argc, argv);: Initializes the GLUT library.

glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB);: Sets the display mode to single buffering and RGB color mode.

glutInitWindowSize(500, 500);: Sets the initial window size.

glutInitWindowPosition(40, 40);: Sets the initial window position.

glutCreateWindow("Sierpinski Triangle");: Creates the window with the given title.

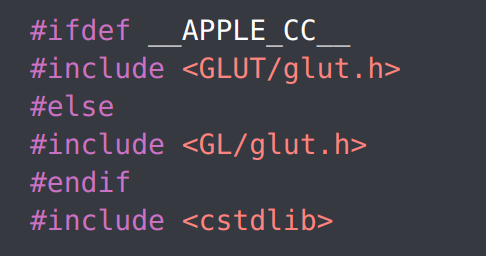
glutDisplayFunc(display);: Registers the display callback function.

init();: Calls the initialization function.

glutMainLoop();: Enters the GLUT main loop, where events are processed.

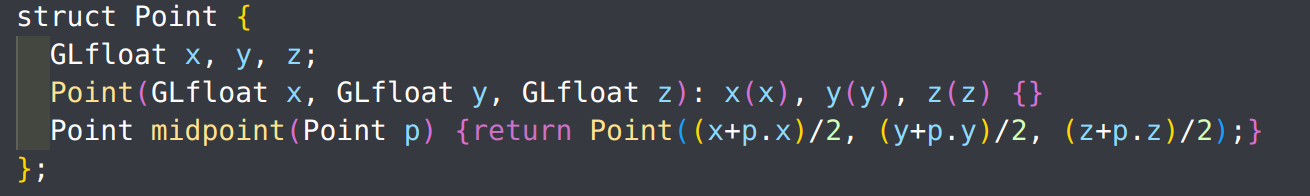
**3D Sierpinski Gasket**

Line 6 - 11



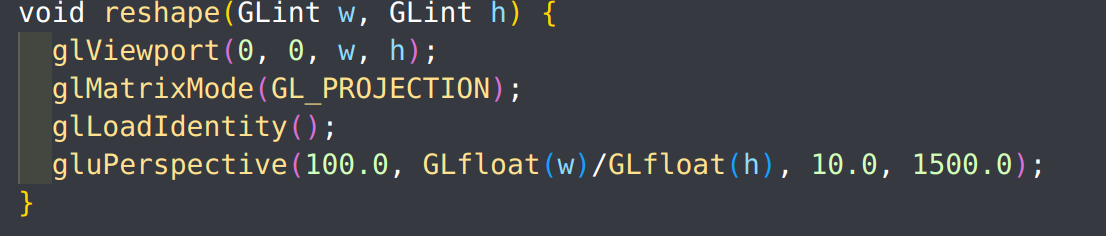
This checks if the program is being compiled on an Apple platform. If it is, it includes the GLUT header specific to Apple. Otherwise, it includes the standard GLUT header. cstdlib is included for the rand() function.

Line 16 - 20



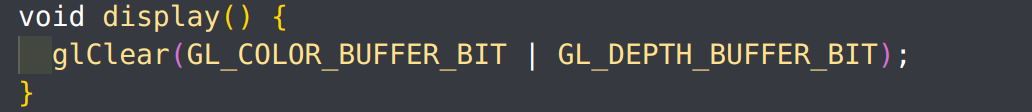
This struct represents a 3D point with x, y, and z coordinates. It also has a function midpoint that calculates the midpoint between the current point and another point p.

Line 27 - 32



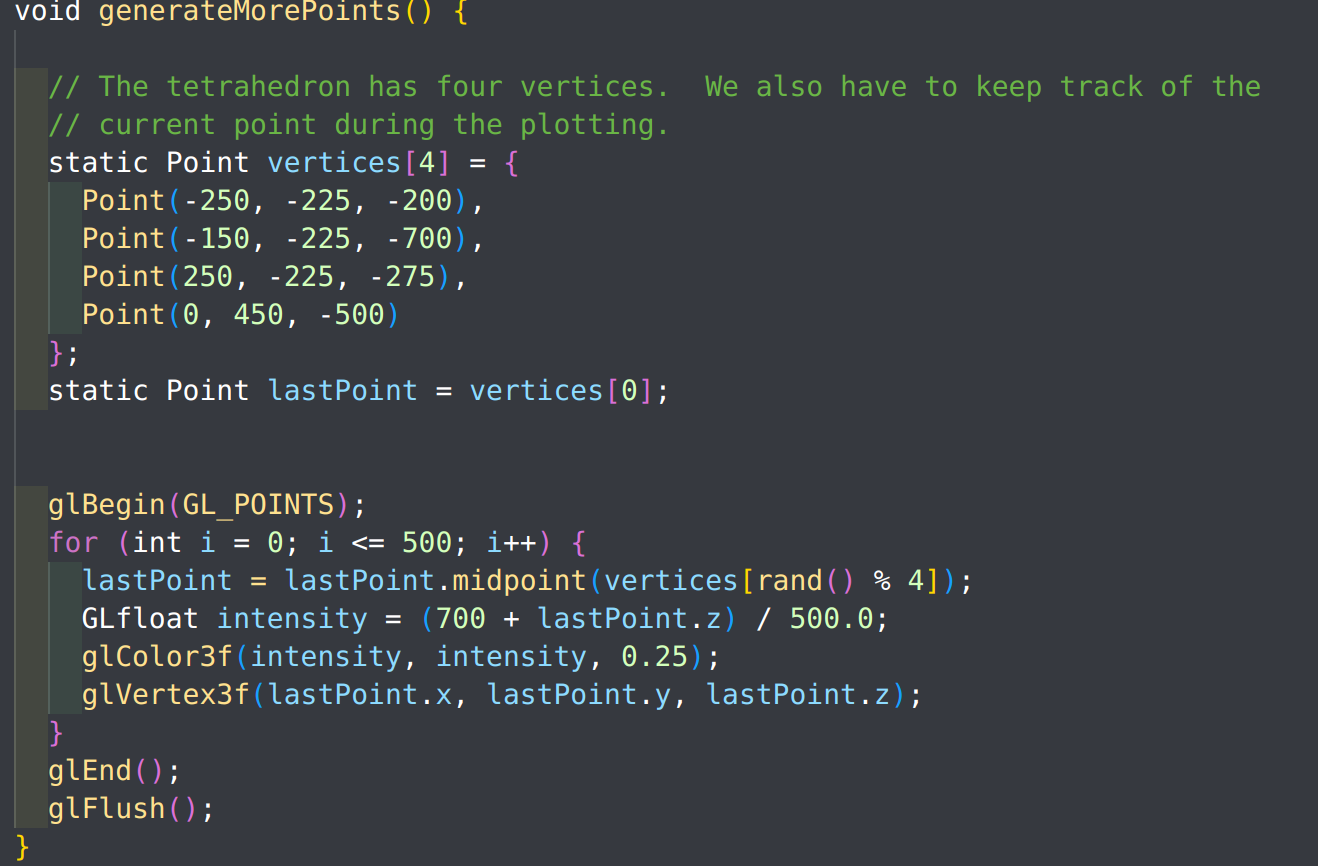
This function is called whenever the window is resized. It sets the viewport to match the window size and sets up a perspective viewing volume. This ensures that the tetrahedron will be displayed correctly regardless of the window's aspect ratio.

Line 36 - 38



This function clears the viewport. The actual drawing is done in the generateMorePoints function, which is set as the idle callback.

Line 43 - 52



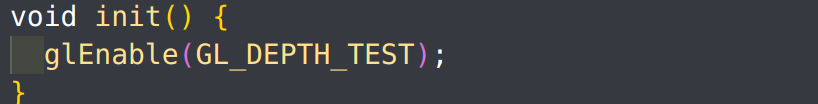
This function generates and plots points for the Sierpinski Tetrahedron:

It defines the four vertices of the main tetrahedron and keeps track of the last plotted point.

The loop generates 500 points. For each point, it calculates the midpoint between the current point and a randomly chosen vertex of the tetrahedron. This is a 3D extension of the chaos game method used for the 2D Sierpinski triangle.

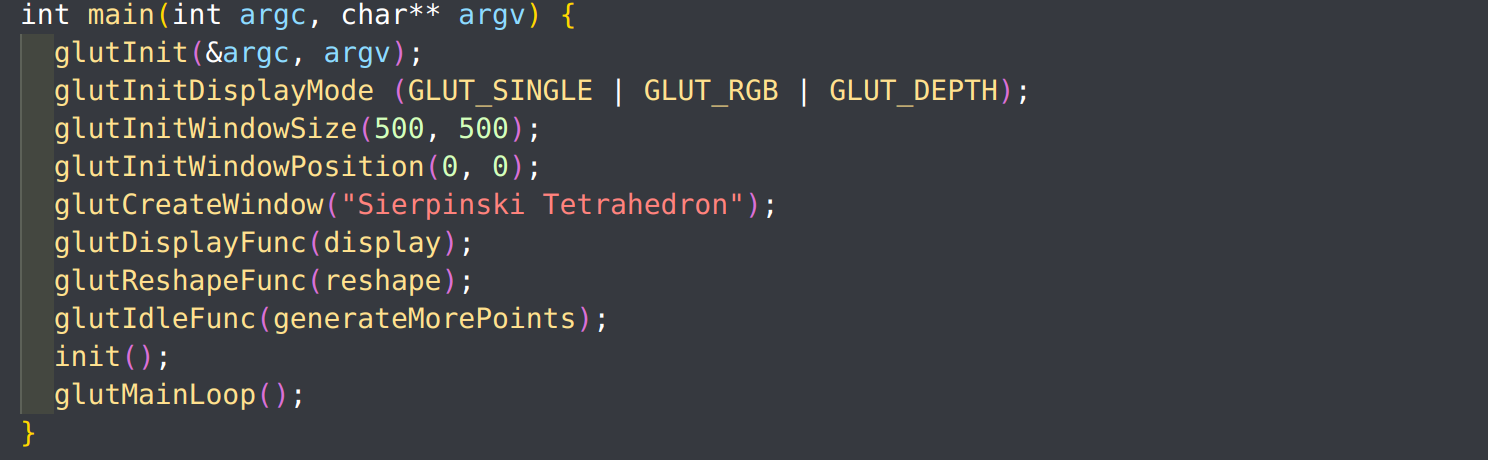
The color of each point is determined by its z-coordinate. Points closer to the camera (higher z-values) are brighter.

Line 73 - 75



This function enables depth buffering in OpenGL, which is necessary for correct rendering of 3D scenes.

Line 79 - 90



This function initializes GLUT, sets up the main window, and registers callback functions:

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);: Sets the display mode to single buffering, RGB color mode, and enables depth buffering.

glutDisplayFunc(display);: Registers the display callback function.

glutReshapeFunc(reshape);: Registers the reshape callback function.

glutIdleFunc(generateMorePoints);: Registers the idle callback function. This function is called whenever there are no events to process, allowing the program to continuously generate and plot points for the tetrahedron.

**Methodology**

The methodology adopted for visualizing the gaskets in OpenGL is based on the "chaos game" approach. This approach is a probabilistic method that involves selecting random points and using them to generate the fractal patterns of the gasket.

**2D Sierpinski Triangle:**

Three vertices of an equilateral triangle are defined.

An arbitrary starting point within the triangle is chosen.

Chaos Game Iteration:

A vertex is randomly chosen from the three vertices of the triangle.

The midpoint between the current point and the chosen vertex is computed.

This midpoint becomes the new current point and is plotted on the screen.

The process is repeated for a large number of iterations (e.g., 100,000 times) to generate the fractal pattern of the Sierpinski Triangle.

**3D Sierpinski Tetrahedron:**

Four vertices of a tetrahedron are defined.

An arbitrary starting point within the tetrahedron is chosen.

Chaos Game Iteration:

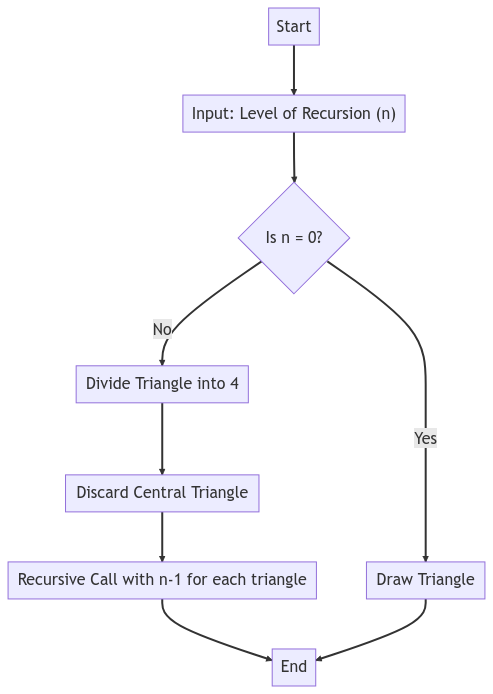
A vertex is randomly chosen from the four vertices of the tetrahedron.

The midpoint between the current point and the chosen vertex is computed.

This midpoint becomes the new current point and is plotted on the screen with a color determined by its depth (z-coordinate).

The process is repeated continuously in the idle function to generate the fractal pattern of the Sierpinski Tetrahedron.

**Algorithm (Flow Chart)**



**2D Sierpinski Triangle:**

Define three vertices of an equilateral triangle.

Choose an initial point P within the triangle.

For i = 1 to N (where N is a large number, e.g., 100,000):

a. Randomly select one of the three triangle vertices, say V.

b. Compute the midpoint M between P and V.

c. Plot the point M.

d. Set P = M.

**3D Sierpinski Tetrahedron:**

Define four vertices of a tetrahedron.

Choose an initial point P within the tetrahedron.

Continuously in the idle function:

a. Randomly select one of the four tetrahedron vertices, say V.

b. Compute the midpoint M between P and V.

c. Determine the color of M based on its z-coordinate (depth).

d. Plot the point M with the determined color.

e. Set P = M.

**ReadME**

---

Ensure you have g++ installed. You can check this by running:

```

g++ --version

```

Ensure you have the required libraries (GLUT, GLU, and GL) installed. On many Linux distributions, you can install them using the package manager. <br>

For example, on Ubuntu:

```

sudo apt-get install freeglut3-dev

```

Navigate to the directory containing your source code :

```

cd home/path

```

Compile the source code using the provided command:

```

g++ [name.cpp] -o [newName] -lglut -lGLU -lGL

```

After successful compilation, you can run the program with:

```

./[newName]

```