**Project 1 - Sierpinski Gasket**

Kyler Harden, Angel Velazquez

Grand Canyon University

CST - 310

Professor Citro

September 5, 2024

**Purpose:**

The purpose of this assignment is to familiarize ourselves with OpenGL. To achieve this, the Sierpinski Gasket has been chosen to teach us how to use and run a 2D and 3D image utilizing the OpenGL API in C++.

**Description:**

There are two programs in this project Sierpinski2D and Sierpinski3D. Both are responsible for creating a Sierpinski triangle also known as Sierpinski Gasket. This is a fractal pattern that represents a triangle with triangles inside of each other in the 2D version, whereas the 3D version is a pyramid with smaller pyramids inside. This is a fractal pattern that

**Methodology:**

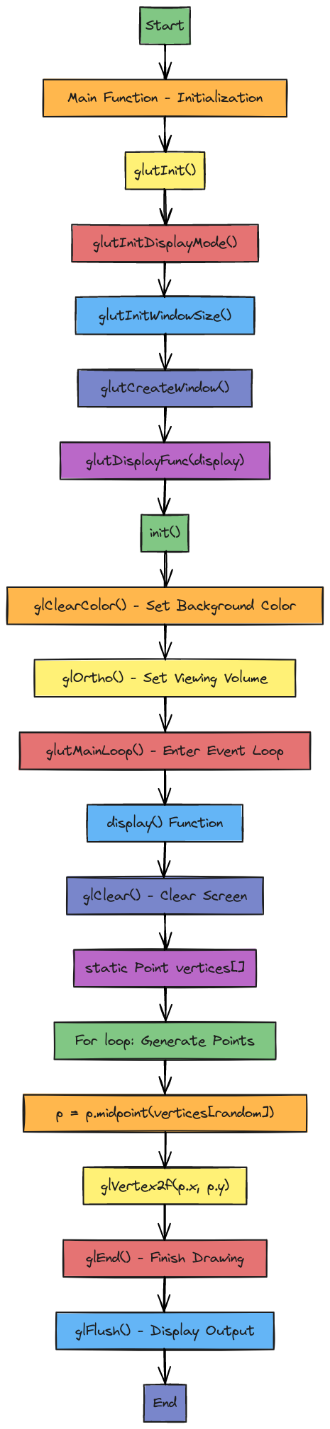
**2D Sierpinski Gasket**

The code for the 2D Sierpinski generates a Sierpinski gasket in 2 dimensional form, which is a fractal pattern, utilizing the OpenGL API. First, we see that a struct is created for each point that we use. This makes the process easier by allowing us to reference points with x and y coordinates instead of using array indicies. Plus, it encapsulates a midpoint function, which is essential to creating the Sierpinski gasket. The following init() function is responsible for the basic OpenGL initialization. It sets the background color to deep purple, specifies the color to be used for the points at a greenish-yellow, and configures the 2D projection to map the 500x500 window. Next, a traingle is defined by three vertices, represented as (0, 0), (200, 500), and (500, 0). These vertices form the main traingle from which the fractal begins. To generate the fractal, we need to utilize random points that start with vertices[0]. In the display function, for every new point created, a vertex is randomly selected, and the midpoint between the selected point and the randomly generated point is chosen. This midpoint is plotted on the display. This process is repeated in a for loop for 100,000 iterations. Last but not least, the main function initializes the OpenGL Utility Toolkit to create a window, set up the display mode, and calls the display function as discussed earlier.

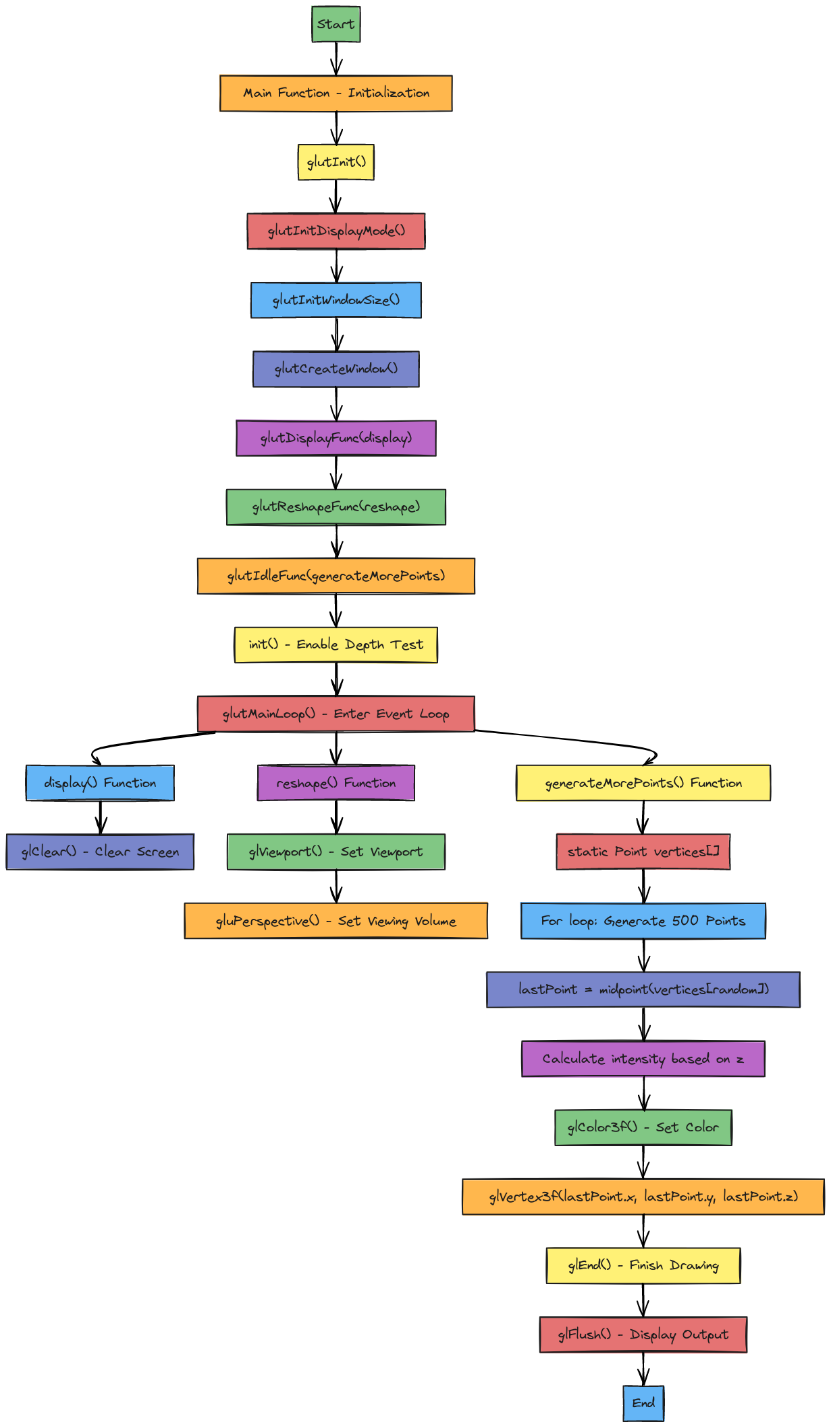
**3D Sierpinski Gasket**

The program for the 3D Sierpinski Gasket generates a 3 dimensional version of the fractal. Essentially, it is simply an extension of the 2D Sierpinski triangle program that was previously discussed. However, this program makes the visualization more appealing to the user by fading the color in order to show depth to the render. Similarly to the 2D program, the 3D program uses a struct to define a point with x, y, and z coordinates. The midpoint function that calculates the midpoint between two 3D points is also included. The reshape function adjusts the OpenGL viewport to make sure that the correct aspect ratio is maintained when the user resizes the window (*Reshape*, 2010). From analyzing the display function, it can be concluded that it acts as a placeholder that clears the display using both color, and depth buffers. Since the rendering is done in the generateMorePoints function, the display function only clears the screen when necessary. The generateMorePoints function implements the midpoint algorithm as discussed previously from the 2D program to plot 500 points of the fractal each time it is called. The function declares four points that form the corners of the 3D structure, chooses an initial point and calculates the midpoint between the current point and a randomly selected point, and plots the points in 3D space and adds a color to them depending on the point’s z value. This last task is important to giving the structure a sense of depth to the user. Finally, the main function initializes the OpenGL Utility Toolkit similarly to the 2D program to set up the display mode, window size, color, and depth buffering.

**Flowchart:**



The flowchart above represents the code flow for the 2D Sierpinski program. To start, we use the main function to initialize all our display and window-based API calls using the Sierpinski program. To start, we utilize the main function to initialize all our display and window-based API calls utilizing the OpenGL Utility Toolkit. After that, we dig into the display function where we begin using the OpenGL API to load clear our color buffer and load our pixels onto the screen. Once that is completed, the init function begins loading our data and sampling the colors we wish to use on our window.



The flowchart above shows the process used to display a 3D Sierpinski triangle. The main function initializes all our window elements as discussed previously utilizing the OpenGL Utility Toolkit. After that, we enter an event loop through GLUT in which three separate branches are created. To begin from the left, the display function is used to clear the screen. The reshape function is used to set the viewport height and width of the window and set the viewing angle in which the 3D prism will be viewed from the viewer. In the generate points function, a point struct is used to make our lives just a little easier. A for loop is used to generate 500 point structs. The way this process works is by selecting a random point inside of an already existing triangle, and putting a point in the middle of that created vector from the corner of the triangle to the point that was randomly chosen. Over time, this creates a Sierpinski triangle.

**Key Coding Elements:**

**Sierpinski 2D:**

The 2D Sierpinski program has a few key coding elements that are worth discussing briefly. To begin, the Point struct is a simple 2D point classification, which uses GLfloat types for OpenGL compatibility. Additionally, the midpoint method that is encapsulated within this struct calculates the midpoint between two points, which is key to forming the fractal. It is also worth mentioning the display callback function which contains the main logic to draw the Sierpinski triangle. It clears the display buffer, defines vertices, iterates through 100,000 points while computing new points halfway between the previous point, and plots each point. The points are generated randomly by selecting one of the triangle’s vertices and plotting a new point halfway between the last plotted point, and the selected point.

**Sierpinski 3D:**

The 3D Sierpinski program also has a few key coding elements that are worth discussing briefly. The Point struct included is a 3D version of the earlier 2D Point struct. However, it now includes a z coordinate to help render the point in 3D space. The midpoint function that is in the struct works similarly to that of the 2D version; it returns the midpoint between two points in 3D space. Additionally, the program includes a reshape callback function which handles window resizing events by setting up a new viewport and projection whenever the user resizes the window.

**Screenshots:**

The following screenshot is obtained from running the Sierpinski2D program. This program creates a Sierpinski triangle in 2D which is a fractal with a shape of a triangle.

A screenshot of a computer screen

Description automatically generated

The following screenshot is obtained from running the Sierpinski3D program. This program creates a Sierpinski triangle in 3D which a fractal with a shape of a pyramid.

A screenshot of a computer

Description automatically generated

[Github Repo](https://github.com/angel-vlzqz/Graphics/tree/main/Project%201)

**References:**

*reshape*. (2010, May 12). Khronos Forums. <https://community.khronos.org/t/reshape/60969>