Project 1: Sierpinski Gasket

Adrian Tudur, Omar Elsayd, and Haley Hansen

Grand Canyon University

CST: 310

Professor Citro

September 11, 2022

Project Description:

* Ensure successful installation of linux, g++, and OpenGl. (See figure 1,2 and 3 for proof of installation)
* Test and redner a graphics test program using OpenGl API (Please check figure 6)
* Running and testing graphics rendering of Sierpinski Gasket 2D shape using OpenGl (figure 5)
* Running and testing graphics rendering of 3D Sierpinski Gasket shape using OpenGl (figure 4)
* Explaining the Methodology and Approach
* Explaining Algorithm behind the code.
* Delivering a working project with documentation (Please check the ReadMe file) on how to install the essential libraries and how to run the programs

Methodology/Approach:

OpenGL (Open Graphics Library) is a cross-language and cross-platform API (application programming interface) for rendering 2D and 3D vector graphics. That being said, API helps to interact with the graphics processing unit to get the hardware rendering. Now to create the first shape as a test (figure 6), we create the corners of the square first using an implementation between glBegin(GL\_POLYGON) and glEnd(). Then we usie glVertex3f(x,y,z) to create the vertices and essentially we use the x,y axis to draw our shape in 2D or x,y,z in 3D. Then we use glFlush() to flush up the graphics and be ready for display.

The next block of code is essentially will be common in most scripts as it is responsible for displaying the windows :

{

glutInit(&argc, argv); // initialize the toolkit

glutInitDisplayMode(GLUT\_SINGLE); // set display mode

glutInitWindowSize(400, 300); // set window size

glutInitWindowPosition(100, 100); // set window position on screen

glutCreateWindow("Hello world!"); // open the screen window

glutDisplayFunc(displayMe); // register redraw function

glutMainLoop(); // go into a perpetual loop

return 0;

}

**Creating 2D Sierpinski Gasket:**

First we construct 2 points (X,Y) using Glfloat and contractor the two point as showing below the we compute the midpoint

struct Point {

GLfloat x, y; // The x and y coordinates of the point

Point(GLfloat x = 0, GLfloat y = 0): x(x), y(y) {} // Constructor

Point midpoint(Point p) {return Point((x + p.x) / 2.0, (y + p.y) / 2.0);} // Midpoint

};

Now we will have a display function

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT); // Clear the screen

static Point vertices[] = {Point(0, 0), Point(200, 500), Point(500, 0)}; // The vertices of the triangle

// Compute and plot 100000 new points, starting (arbitrarily) with one of

// the vertices. Each point is halfway between the previous point and a

// randomly chosen vertex.

static Point p = vertices[0]; // Start with one of the vertices

glBegin(GL\_POINTS); // Begin plotting points

for (int k = 0; k < 100000; k++) { // Plot 100000 points

p = p.midpoint(vertices[rand() % 3]); // Compute the midpoint

glVertex2f(p.x, p.y); // Plot the new point

}

glEnd(); // Done plotting points

glFlush(); // Finish rendering

}

Essentially we will create the vertices of the triangle and then start with one point. We will loop over 1000 points and then compute the midpoints.

void init() {

// Set a deep purple background and draw in a greenish yellow.

glClearColor(0.25, 0.0, 0.2, 1.0); // Set the background color

glColor3f(0.6, 1.0, 0.0); // Set the drawing color

// Set up the viewing volume: 500 x 500 x 1 window with origin lower left.

glMatrixMode(GL\_PROJECTION); // Set up a projection

glLoadIdentity(); // Initialize the projection

glOrtho(0.0, 500.0, 0.0, 500.0, 0.0, 1.0); // Set the viewing volume

}

Now we will Set the background color, Set the drawing color, Set up a projection, Initialize the projection and Set the viewing volume as shown in the code above.

Following these steps, and creating the same window we did for the test OpenGl, we will have a 2D Sierpinski Gasket (Figure 5).

**Creating 3D Sierpinski Gasket:**

The same methodology applies just as we deal now with 3 axes which are x,y, and z.

struct Point {

GLfloat x, y, z; // The x, y, and z coordinates of the point

Point(GLfloat x, GLfloat y, GLfloat z): x(x), y(y), z(z) {} // Constructor

Point midpoint(Point p) {return Point((x+p.x)/2, (y+p.y)/2, (z+p.z)/2);} // Midpoint

};

As shown in the code, we added the z axis then we get the midpoints. Then we will initialize the same projection as we did for 2D. Also, we will guide the points and then we created tetrahedron

static Point vertices[4] = { // The vertices of the tetrahedron

Point(-250, -225, -200), // Lower left front

Point(-150, -225, -700), // Lower right front

Point(250, -225, -275), // Lower right back

Point(0, 450, -500) // Upper middle

};

Then we will have our loop to plot the points :

glBegin(GL\_POINTS); // Start drawing points

for (int i = 0; i <= 500; i++) { // Loop 500 times

lastPoint = lastPoint.midpoint(vertices[rand() % 4]); // Pick a random vertex

GLfloat intensity = (700 + lastPoint.z) / 500.0; // Compute the intensity

glColor3f(intensity, intensity, 0.25); // Set the color

glVertex3f(lastPoint.x, lastPoint.y, lastPoint.z); // Draw the point

}

glEnd(); // Done drawing points

glFlush(); // Finish rendering

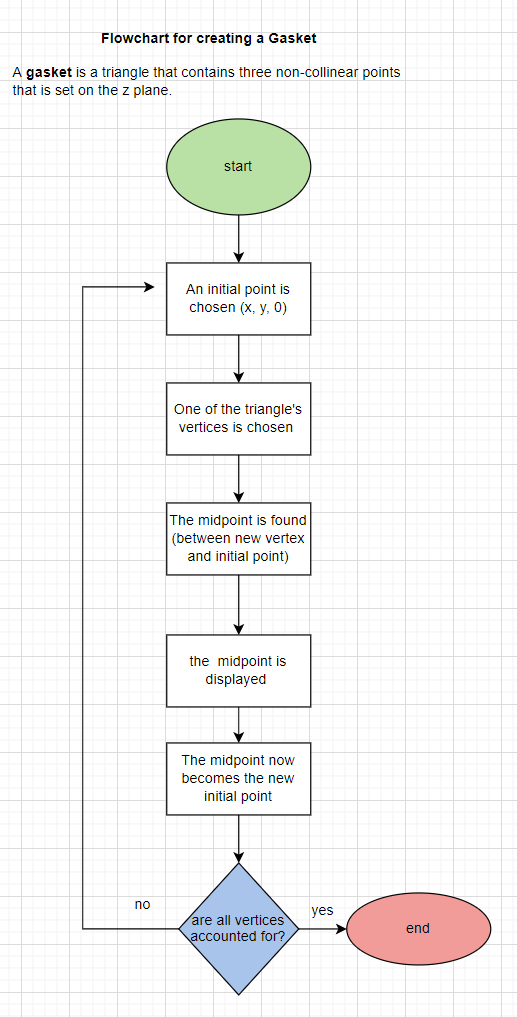
}

Now we have 3D Sierpinski Gasket (Figure 4)

Gasket Algorithm:

The gasket algorithm is based off of a loop. The loop draws 500 points which makes the gasket. In the loop it first finds the last point which uses the function “lastPoint.midpoint(vertices[rand() % 4]);” this function finds the last point. After the last point is created the coloring is done. It takes the z coordinate and adds 700 to the last point and then divides it by 500 which gives the coloring a value from 0 to 1 which allows it to be colored correctly. Lastly the point x y and z vertex are set using the last point. The function is “glVertex3f(lastPoint.x, lastPoint.y, lastPoint.z);”. The algorithm is repeated 500 times which creates the gasket in openGL.

Flowchart:



Screenshots:

Figure 1

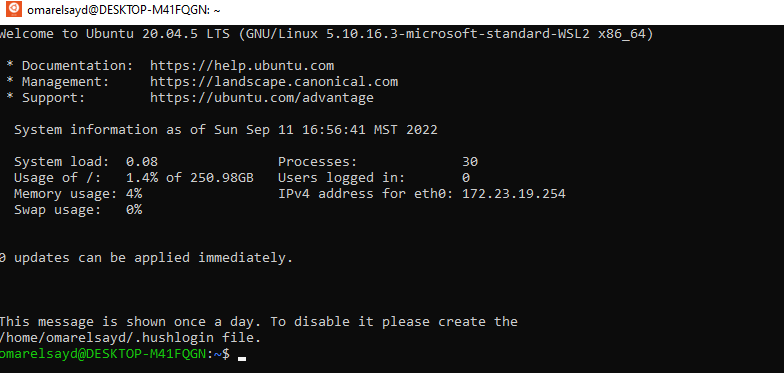


Figure 2

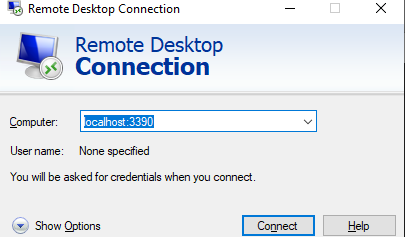


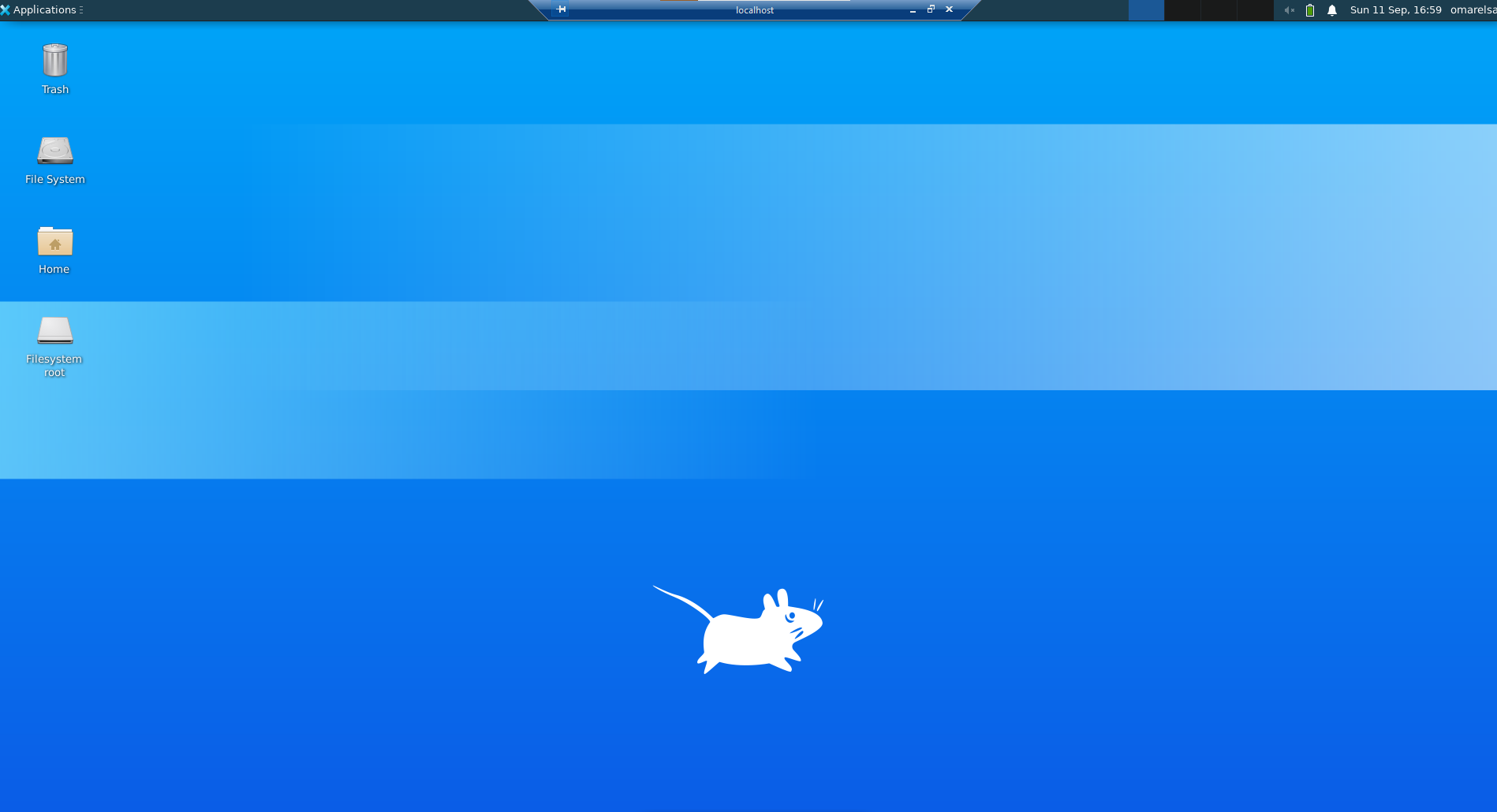
Figure 3

Figure 4

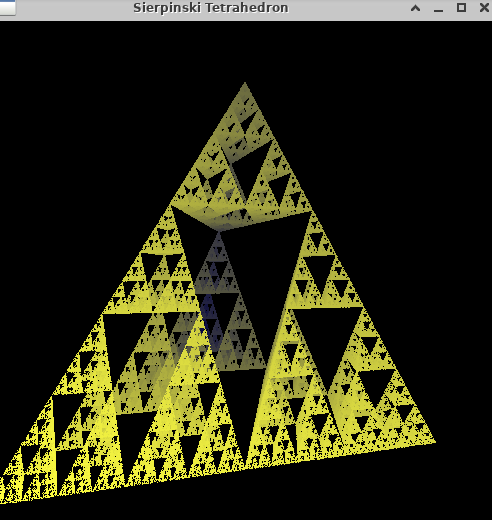


Figure 5

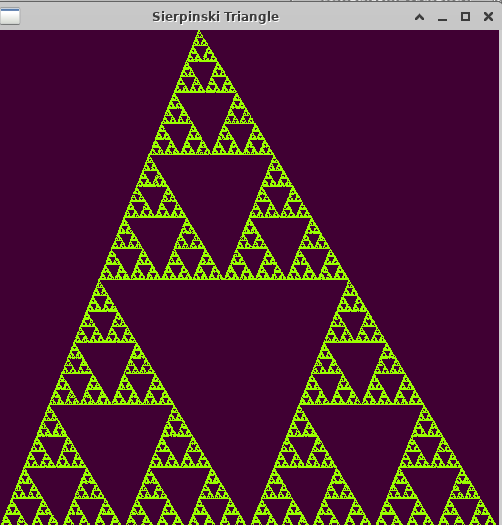
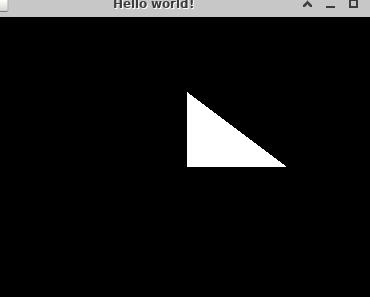


Figure 6



**References:**

<http://www.sci.brooklyn.cuny.edu/~weiss/cisc3620/Lectures/02-Chapter02.html>