Assignment 1

CS4610

Michael Rallo – msr5zb – 12358133

2/14/2017

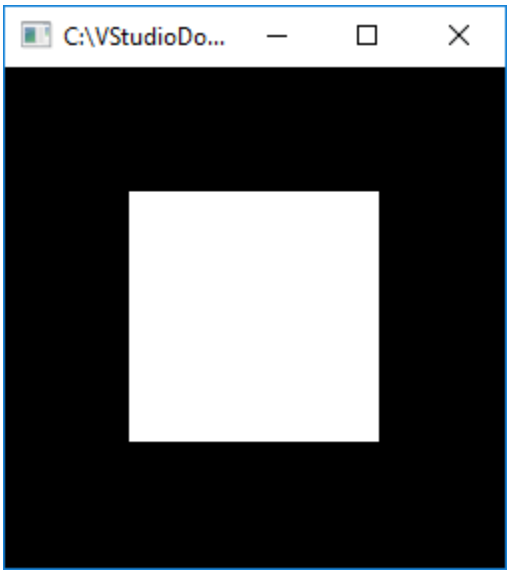
Assignment 1 was broken into 3 parts: A, B, C.

The *purpose* of this assignment was to familiarize ourselves about the basic functions of OpenGL. Such functions include creating and displaying objects, mouse click events, and timers.

Part A:

Objective: Download the program double.c (Example 1-3 from the OpenGL red book), create a VC++ project, compile and run it.

Approach: I downloaded and installed Visual Studio Community 2015, along with OpenGL. I created the VC++ projects and inserted the double.c. Once compiled and built, I was presented with the following:



On left click, the square rotates origin its center. When right clicked, the square would stop rotating. Upon examining the code, I saw that there were many important functions needed to make this project work.

Main

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(250, 250);
    glutInitWindowPosition(100, 100);
    glutCreateWindow(argv[0]);
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMouseFunc(mouse);
    glutMainLoop();
    return 0;   /* ANSI C requires main to return int. */
}
```

I will be explaining the functions after init(), for the ones before it are size=explanatory.

- *glutDisplayFunct(display)* took in a function that built and display the initial square.
- *glutReshapeFunc(reshape)* adjusts the view for the square upon window resize.
- *glutMouseFunc(mouse*) adds a mouse eventListener to the window, which gives us our functionality to decide when to rotate the square.
- *glutMainLoop()* ensures that our program keeps running and updating, letting our shapes be created and manipulated in real-time.

Mouse eventListener

```
void mouse(int button, int state, int x, int y)
{
    switch (button) {
    case GLUT_LEFT_BUTTON:
        if (state == GLUT_DOWN)
            glutIdleFunc(spinDisplay);
        break;
    case GLUT_MIDDLE_BUTTON:
    case GLUT_RIGHT_BUTTON:
        if (state == GLUT_DOWN)
            glutIdleFunc(NULL);
        break;
    default:
        break;
    }
}
```

Here is our eventListener for our mouse. Note we specifically target the left_button and the right_button. We can see that the glutIdleFunc(spinDisplay) causes the spinDisplay function to repeatedly trigger as time goes on (as time idles).

Animating

```
static GLfloat spin = 0.0;

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix();
    glRotatef(spin, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glRectf(-25.0, -25.0, 25.0, 25.0);
    glPopMatrix();

    glutSwapBuffers();
}

void spinDisplay(void)
{
    spin = spin + 2.0;
    if (spin > 360.0)
        spin = spin - 360.0;
    glutPostRedisplay();
}
```

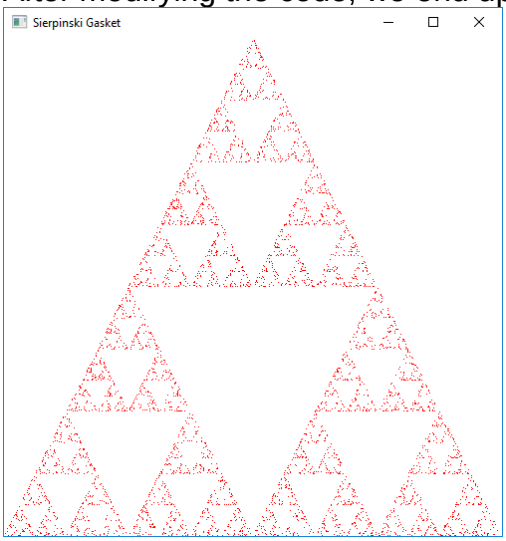
As mentioned before, the spinDisplay is being repeatedly called. Each time this is called, the variable spin gets adjusted. This spin variable is used as a rotation variable to tell how much the square is to be rotated. Note, after each spinDisplay call, we redisplay the shape with glutPostRedisplay().

Part B:

Objective: Run the three sample programs described in Chapter 2 of the textbook (Appendix A): A.1: Sierpinski Gasket Program; A.2: Recursive Sierpinski Gasket; and A.3: Recursive 3D Sierpinski Gasket with the following modification:

a) In A.1, display(), comment out the following line:
int rand();

After modifying the code, we end up with the shape:



Note the way this program functions is very similar to that of double.c. It is how the shape is structured that makes them differ greatly. It is formed based of the Sierpinski algorithm of dividing triangles into triangle and so on. Below is a snippet of the main bit of the algorithm.

```
void divide_tetra(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d, int m)
{

    GLfloat mid[6][3];
    int j;
    if(m>0)
    {
        /* compute six midpoints */

        for(j=0; j<3; j++) mid[0][j]=(a[j]+b[j])/2;
        for(j=0; j<3; j++) mid[1][j]=(a[j]+c[j])/2;
        for(j=0; j<3; j++) mid[2][j]=(a[j]+d[j])/2;
        for(j=0; j<3; j++) mid[3][j]=(b[j]+c[j])/2;
        for(j=0; j<3; j++) mid[4][j]=(c[j]+d[j])/2;
        for(j=0; j<3; j++) mid[5][j]=(b[j]+d[j])/2;

        /* create 4 tetrahedrons by subdivision */

        divide_tetra(a, mid[0], mid[1], mid[2], m-1);
        divide_tetra(mid[0], b, mid[3], mid[5], m-1);
        divide_tetra(mid[1], mid[3], c, mid[4], m-1);
        divide_tetra(mid[2], mid[4], d, mid[5], m-1);

    }
    else(tetra(a,b,c,d)); /* draw tetrahedron at end of recursion */
}
```

These points are so small because we iterate through this 5,000 times!

```
for (k = 0; k<5000; k++)
{
    j = rand() % 3; /* pick a vertex at random */


                    /* Compute point halfway between selected vertex and old point */

    p[0] = (p[0] + vertices[j][0]) / 2.0;
    p[1] = (p[1] + vertices[j][1]) / 2.0;

    /* plot new point */

    glVertex2fv(p);

}
```

Also note how we are working with 2D triangles

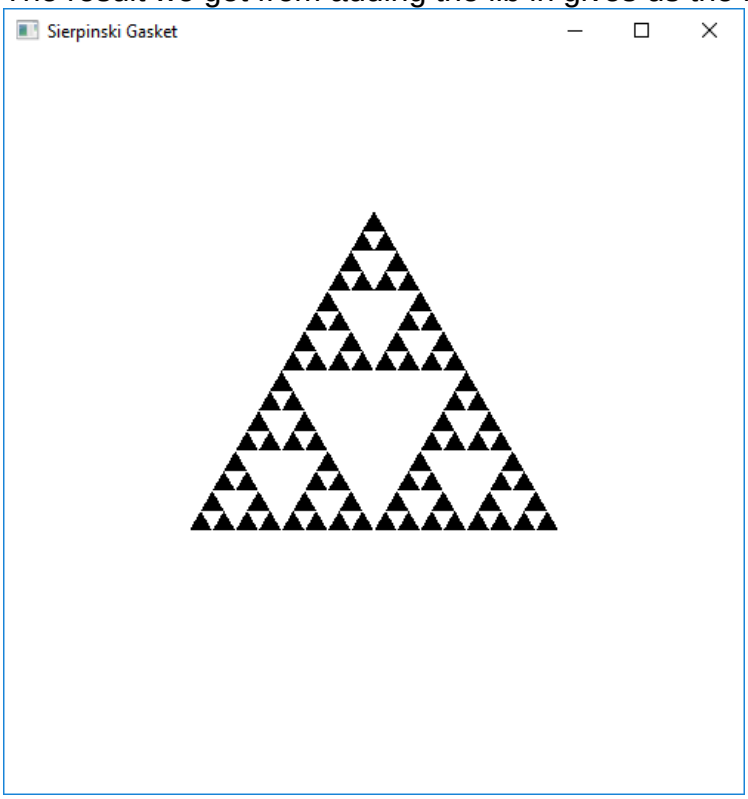
```
void triangle( GLfloat *a, GLfloat *b, GLfloat *c)

/* specify one triangle */
{
       glVertex2fv(a);
       glVertex2fv(b);
       glVertex2fv(c);
}
void triangle( GLfloat *a, GLfloat *b, GLfloat *c)

/* specify one triangle */
{
       glVertex2fv(a);
       glVertex2fv(b);
       glVertex2fv(c);
}
```

b) In the beginning of A.1, add the following line:
#include <stdlib.h>

The result we get from adding the lib in gives us the following shape:

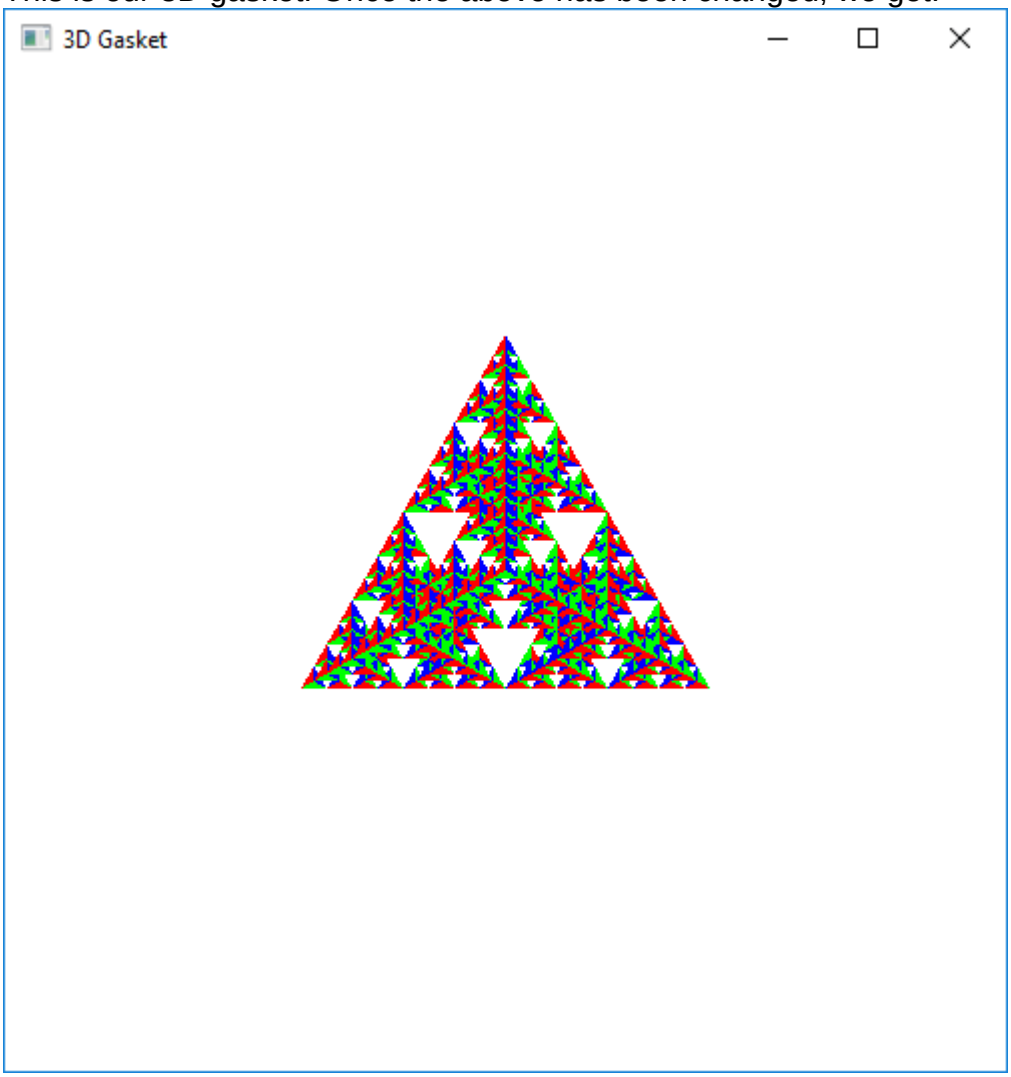


This uses the same algorithm as the previous shape in (Part B a), however instead of having 5,000 iterations, and now have only 4.

```
int main(int argc, char **argv)
{
    //n=atoi(argv[1]); /* or set number of subdivision steps here */
    n = 4; /* or set number of subdivision steps here */
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Sierpinski Gasket");
    glutDisplayFunc(display);
    glutMouseFunc(mouse);
    myinit();
    glutMainLoop();
}
```

c) In the main function of A.2 and A.3,
replace "atoi(argv[1])" by a fixed integer number such as 4.

This is our 3D gasket. Once the above has been changed, we get:



This is a 3D shape in OpenGL. As such, we need to have used the 3D vertex functions.

```
void triangle(GLfloat *va, GLfloat *vb, GLfloat *vc)
{
	glVertex3fv(va);
	glVertex3fv(vb);
	glVertex3fv(vc);
}
```

The algorithm is the same as the previous 2 shapes, the difference is simply we are working with 3D as opposed to 2D.

Part C:

Enhance the three programs in Assignment Part B such that the user can interact with it in a similar fashion as Assignment 1a:

* Pressing the left mouse button rotates the Gasket.
* Pressing the right mouse button stops the rotation.

In order to enhance all three of these functions, there we particular methods that needed to be added in. The most important methods being glutMouseFunc(mouse), the spinDisplay(), and the mouse() methods. With these, I was able to add mouse events that would manipulate how the shape would work.

Gasket1

Gasket 1 was probably the trickiest one of them all – For the origin of rotation was note centered within the shape. I am assuming this was due to how the shape was structured and divided. To combat this, I added translate functions before and after a rotation.

```
glTranslatef(25, 21, 0);
glRotatef(spin, 0.0, 0.0, 1.0);
glTranslatef(-25, -21, 0);
```

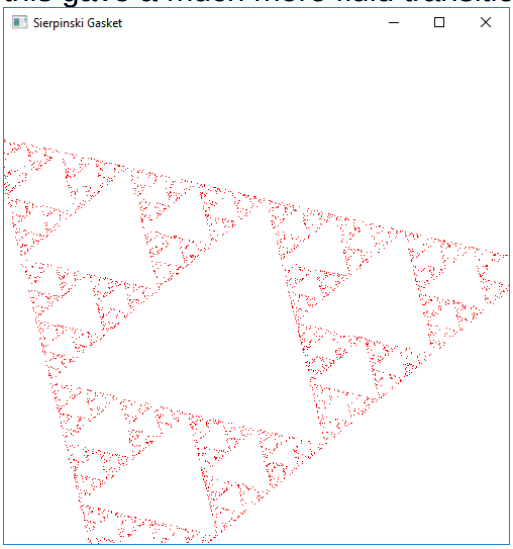
On top of this, I slowed down the rotation speed by lowering the angle of degree it rotates on.

```
void spinDisplay(void)
{
    spin = spin + 0.5;
    if (spin > 360.0) {
        spin = spin - 360.0;
    }

    glutPostRedisplay();

}
```

The reason I chose to lower the angle as opposed to a timeout function was because this gave a much more fluid transitioning.



Gasket2

Gasket 2 was the easiest of them all to implement. Like Gasket1, I had to implement the mouse eventListener and function to manipulate the shape.

I also adjusted the angle of rotation to be lower so that the transition is smoother and slower.

Gasket3

Once again, for gasket 3 I need to add the mouse eventListener in order to rotate the shape. Because this is a 3D shape, I decided to add a slightly different functionality to this shape. Instead of only rotating on the x/y coordinate plane, I decided to rotate 3 dimensionally, using the same spin variable so that the pace of equally distributed.

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    //glRotatef(spin, 0.0, 0.0, 1.0);
    glRotatef(spin, spin, spin, spin);
    glBegin(GL_TRIANGLES);
    divide_tetra(v[0], v[1], v[2], v[3], n);
    glEnd();
    glPopMatrix();
    glFlush();
}
```

This makes for a much better view on how the 3D shape looks within OpenGL.