Assignment 2a

CS4610

Michael Rallo – msr5zb – 12358133

2/27/2017

Assignment 2a

The *purpose* of this assignment was to familiarize ourselves with loading in files, specifically OBJ files, and manipulating its data to render and display geometry in OpenGL.

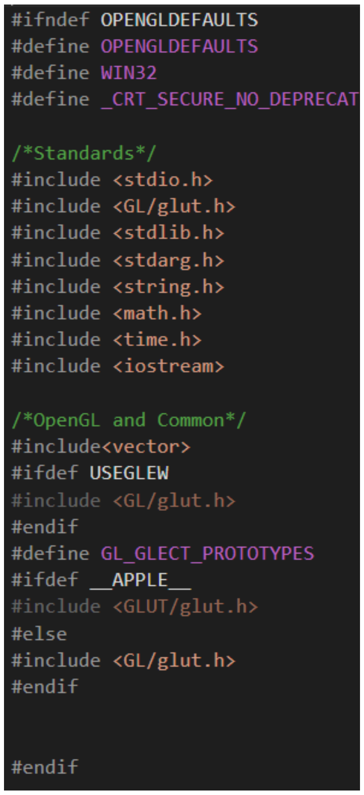
Objectives:

1. Read in OBJ mesh files containing a geometric model represented as triangle meshes and display it centered in the display window.
2. Render an object using points, wireframe and surface representations.

Approach: I created a function that would load in the OBJ file data into global arrays to be rendered later on. In order to ensure that the object was displayed correctly, I kept track of the min/max positions of the object and transitions/scaled it to fit in the view. I also added in keyboard functions in order to swap between different objects and display modes.

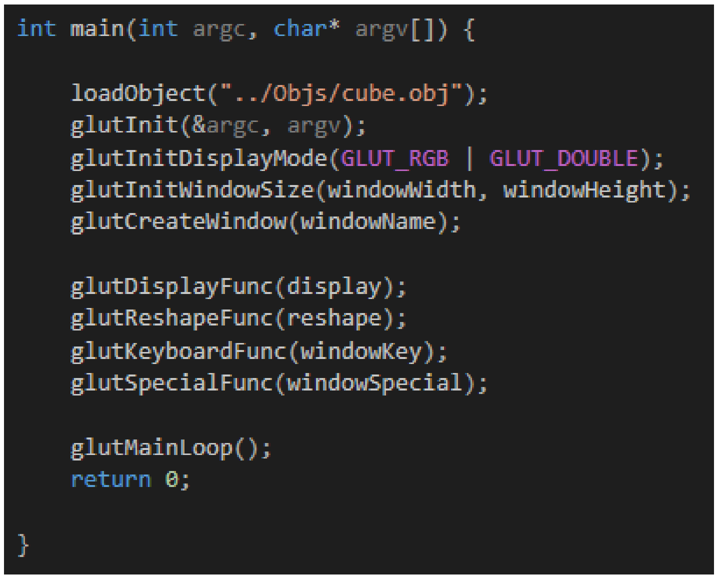
The Header File (OpenGLDefaults.h)

First and foremost, I have decided to include a header file to be used for this assignment's, as well as future assignments', libraries. Note this file include OpenGl basic libraries, as well as printing for debugging and math for easy/complex calculations.



```
#ifndef OPENGLDEFAULTS
#define OPENGLDEFAULTS
#define WIN32
#define _CRT_SECURE_NO_DEPRECAT

/*Standards*/
#include <stdio.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <iostream>

/*OpenGL and Common*/
#include<vector>
#ifdef USEGLEW
#include <GL/glut.h>
#endif
#define GL_GLECT_PROTOTYPES
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif


#endif
```

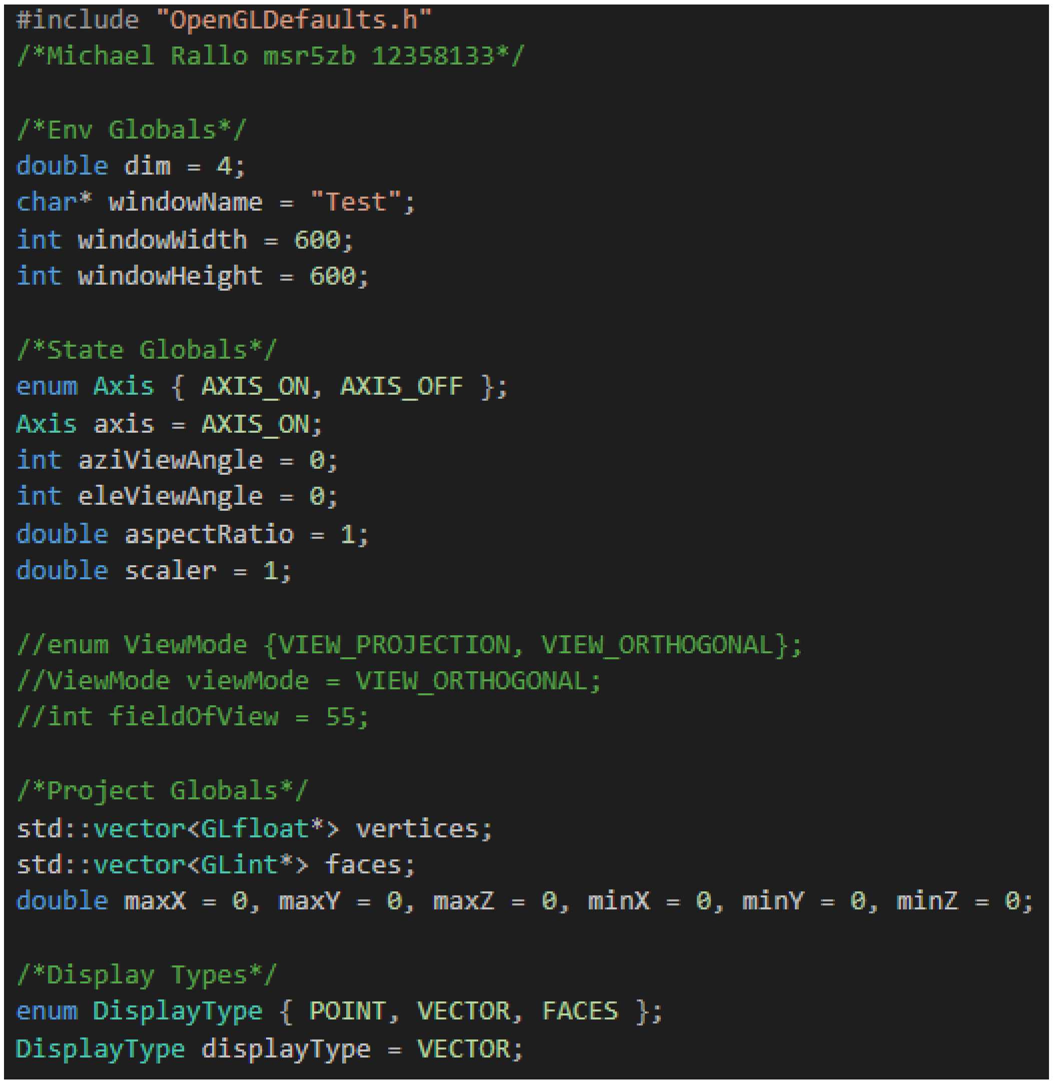
Main



```
int main(int argc, char* argv[]) {

    loadObject("../Objs/cube.obj");
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutInitWindowSize(windowWidth, windowHeight);
    glutCreateWindow(windowName);

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(windowKey);
    glutSpecialFunc(windowSpecial);

    glutMainLoop();
    return 0;

}
```

Our main function is similar to that from Assignment one. Note by default we load in the cube object file.

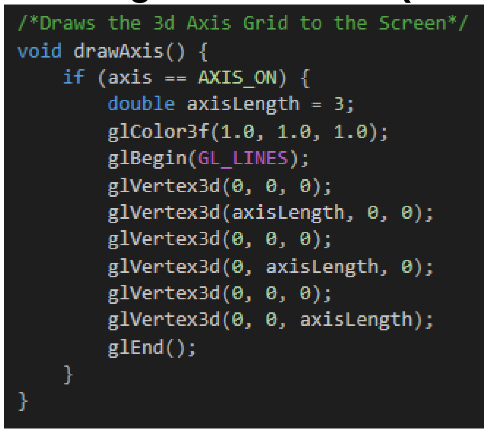
Global Variables



```
#include "OpenGLDefaults.h"
/*Michael Rallo msr5zb 12358133*/

/*Env Globals*/
double dim = 4;
char* windowName = "Test";
int windowWidth = 600;
int windowHeight = 600;

/*State Globals*/
enum Axis { AXIS_ON, AXIS_OFF };
Axis axis = AXIS_ON;
int aziViewAngle = 0;
int eleViewAngle = 0;
double aspectRatio = 1;
double scaler = 1;

//enum ViewMode {VIEW_PROJECTION, VIEW_ORTHOGONAL};
//ViewMode viewMode = VIEW_ORTHOGONAL;
//int fieldOfView = 55;

/*Project Globals*/
std::vector<GLfloat*> vertices;
std::vector<GLint*> faces;
double maxX = 0, maxY = 0, maxZ = 0, minX = 0, minY = 0, minZ = 0;

/*Display Types*/
enum DisplayType { POINT, VECTOR, FACES };
DisplayType displayType = VECTOR;
```

For this assignment, I will be using global variables for the window and view values. I will also be adding in an XYZ Grid in order to “see” the object more clearly in 3d space. Vertices and Faces from the file will be loaded into the vertices and faces vectors. The DisplayType will keep track of how we will be displaying our object.

Drawing the XYZ Grid (Extras)

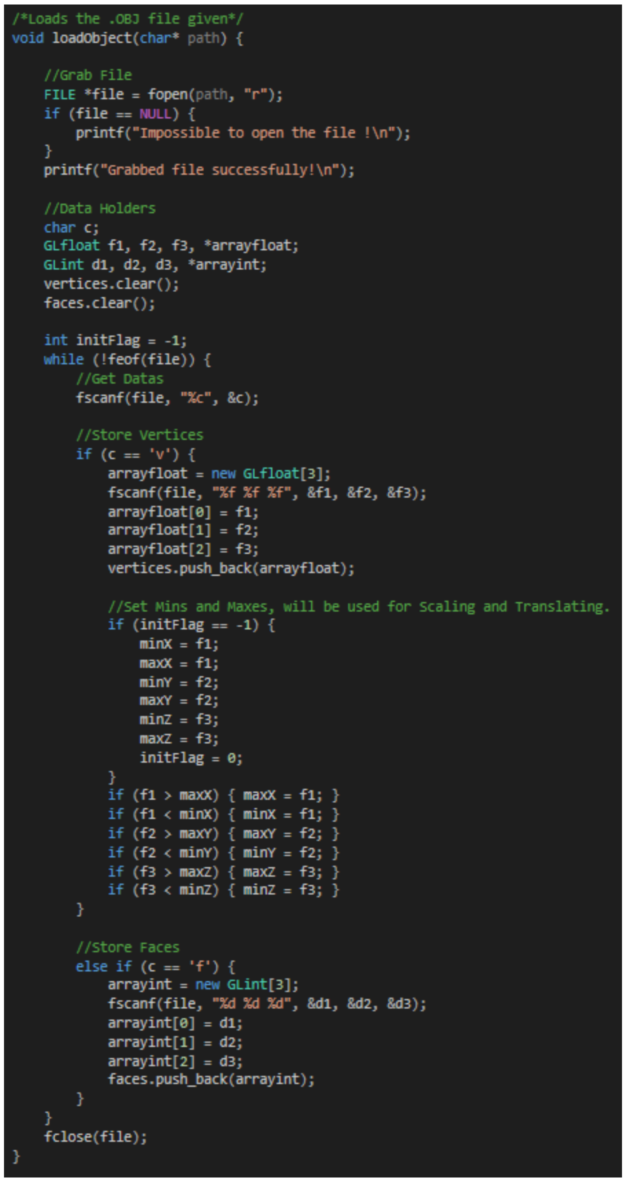


```
/*Draws the 3d Axis Grid to the Screen*/
void drawAxis() {
    if (axis == AXIS_ON) {
        double axisLength = 3;
        glColor3f(1.0, 1.0, 1.0);
        glBegin(GL_LINES);
        glVertex3d(0, 0, 0);
        glVertex3d(axisLength, 0, 0);
        glVertex3d(0, 0, 0);
        glVertex3d(0, axisLength, 0);
        glVertex3d(0, 0, 0);
        glVertex3d(0, 0, axisLength);
        glEnd();
    }
}
```

This is a simple function to draw a grid at the origin of our view in order for us to see the object more clearly. This can be toggled on and off with the "i" key. By default, it is on.

Loading the File (loadObject)

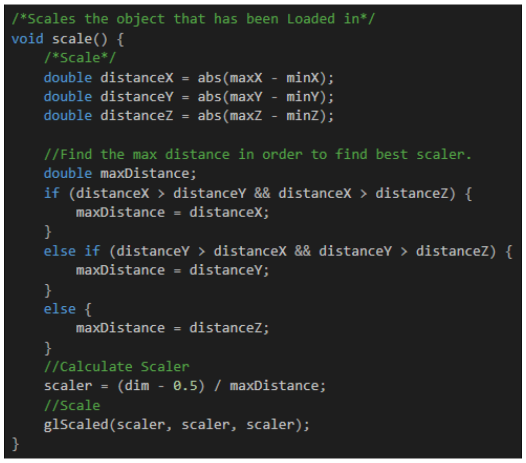
This function take the object the as a parameter and sets our global vertices and faces variables with the data the OBJ file contains. This function also sets the min/max values that will be later used for scaling/transitioning our object.



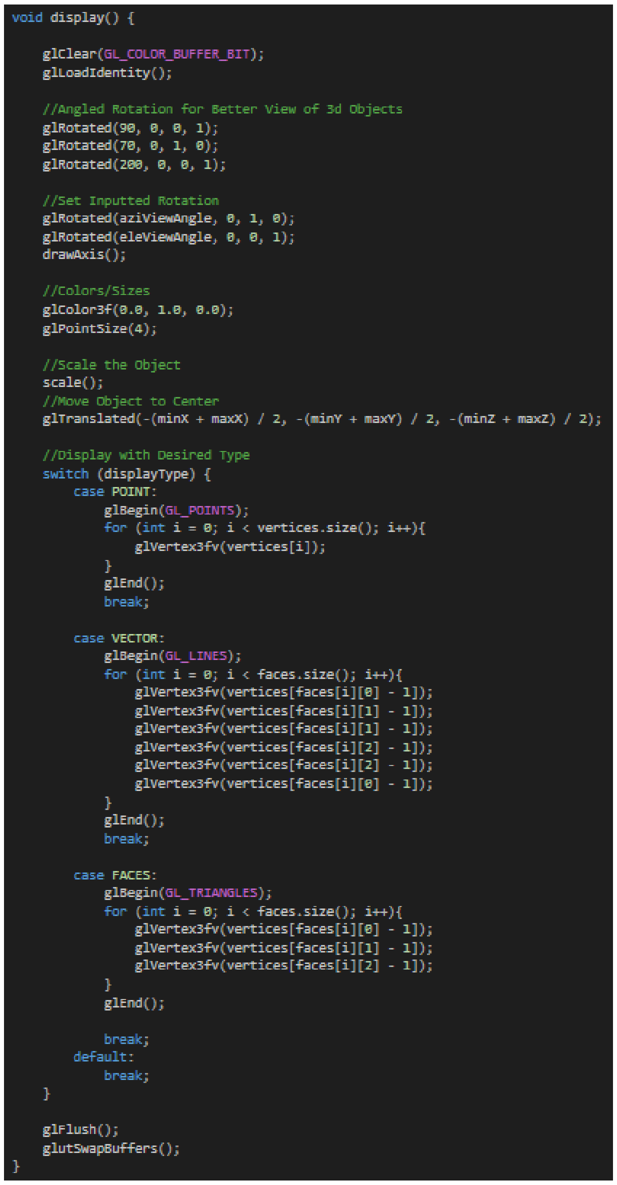
```
/*Loads the .OBJ file given*/
void loadObject(char* path) {

    //Grab File
    FILE *file = fopen(path, "r");
    if (file == NULL) {
        printf("Impossible to open the file !\n");
    }
    printf("Grabbed file successfully!\n");

    //Data Holders
    char c;
    GLfloat f1, f2, f3, *arrayfloat;
    GLint d1, d2, d3, *arrayint;
    vertices.clear();
    faces.clear();

    int initFlag = -1;
    while (!feof(file)) {
        //Get Datas
        fscanf(file, "%c", &c);

        //Store Vertices
        if (c == 'v') {
            arrayfloat = new GLfloat[3];
            fscanf(file, "%f %f %f", &f1, &f2, &f3);
            arrayfloat[0] = f1;
            arrayfloat[1] = f2;
            arrayfloat[2] = f3;
            vertices.push_back(arrayfloat);

            //Set Mins and Maxes, will be used for Scaling and Translating.
            if (initFlag == -1) {
                minX = f1;
                maxX = f1;
                minY = f2;
                maxY = f2;
                minZ = f3;
                maxZ = f3;
                initFlag = 0;
            }
            if (f1 > maxX) { maxX = f1; }
            if (f1 < minX) { minX = f1; }
            if (f2 > maxY) { maxY = f2; }
            if (f2 < minY) { minY = f2; }
            if (f3 > maxZ) { maxZ = f3; }
            if (f3 < minZ) { minZ = f3; }
        }

        //Store Faces
        else if (c == 'f') {
            arrayint = new GLint[3];
            fscanf(file, "%d %d %d", &d1, &d2, &d3);
            arrayint[0] = d1;
            arrayint[1] = d2;
            arrayint[2] = d3;
            faces.push_back(arrayint);
        }
    }
    fclose(file);
}
```

Scaling (Scale)

This scale method finds the greatest distance between the X, Y, and Z axis and uses that as a scaler for this Object. The reason we use the longest distance is so that we can scale everything equally whilst still being in our view.



```
/*Scales the object that has been Loaded in*/
void scale() {
    /*Scale*/
    double distanceX = abs(maxX - minX);
    double distanceY = abs(maxY - minY);
    double distanceZ = abs(maxZ - minZ);

    //Find the max distance in order to find best scaler.
    double maxDistance;
    if (distanceX > distanceY && distanceX > distanceZ) {
        maxDistance = distanceX;
    }
    else if (distanceY > distanceX && distanceY > distanceZ) {
        maxDistance = distanceY;
    }
    else {
        maxDistance = distanceZ;
    }
    //Calculate Scaler
    scaler = (dim - 0.5) / maxDistance;
    //Scale
    glScaled(scaler, scaler, scaler);
}
```



```
void display() {

    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();

    //Angled Rotation for Better View of 3d Objects
    glRotated(90, 0, 0, 1);
    glRotated(70, 0, 1, 0);
    glRotated(200, 0, 0, 1);

    //Set Inputted Rotation
    glRotated(aziViewAngle, 0, 1, 0);
    glRotated(eleViewAngle, 0, 0, 1);
    drawAxis();

    //Colors/Sizes
    glColor3f(0.0, 1.0, 0.0);
    glPointSize(4);

    //Scale the Object
    scale();
    //Move Object to Center
    glTranslated(-(minX + maxX) / 2, -(minY + maxY) / 2, -(minZ + maxZ) / 2);

    //Display with Desired Type
    switch (displayType) {
        case POINT:
            glBegin(GL_POINTS);
            for (int i = 0; i < vertices.size(); i++){
                glVertex3fv(vertices[i]);
            }
            glEnd();
            break;

        case VECTOR:
            glBegin(GL_LINES);
            for (int i = 0; i < faces.size(); i++){
                glVertex3fv(vertices[faces[i][0] - 1]);
                glVertex3fv(vertices[faces[i][1] - 1]);
                glVertex3fv(vertices[faces[i][1] - 1]);
                glVertex3fv(vertices[faces[i][2] - 1]);
                glVertex3fv(vertices[faces[i][2] - 1]);
                glVertex3fv(vertices[faces[i][0] - 1]);
            }
            glEnd();
            break;

        case FACES:
            glBegin(GL_TRIANGLES);
            for (int i = 0; i < faces.size(); i++){
                glVertex3fv(vertices[faces[i][0] - 1]);
                glVertex3fv(vertices[faces[i][1] - 1]);
                glVertex3fv(vertices[faces[i][2] - 1]);
            }
            glEnd();

            break;
        default:
            break;
    }

    glFlush();
    glutSwapBuffers();
}
```

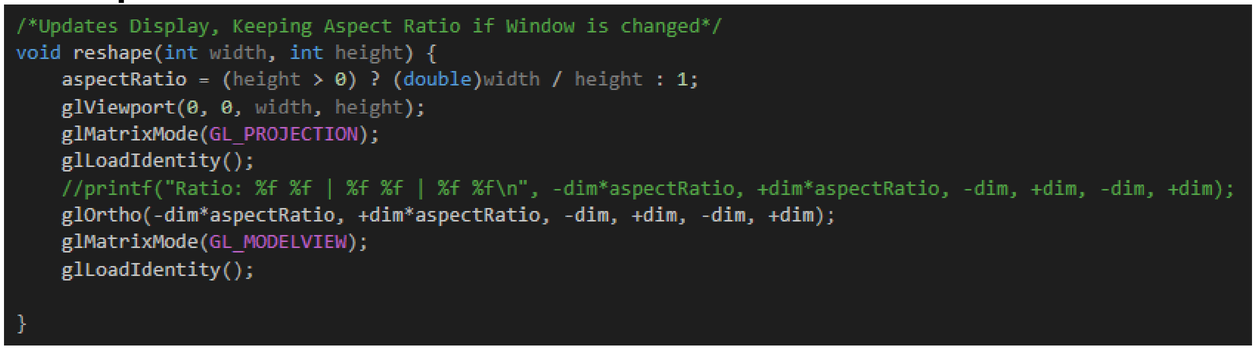
Display

Our display function is pretty simple. We start by rotating the object slightly in order to achieve the best starting position for our object in 3d space. Note we also add in adjusted rotation angles (azi and ele angles).

Our glTranslated function takes the midpoint of our object's X, Y, and Z points in order to translate our object to the origin of our view.

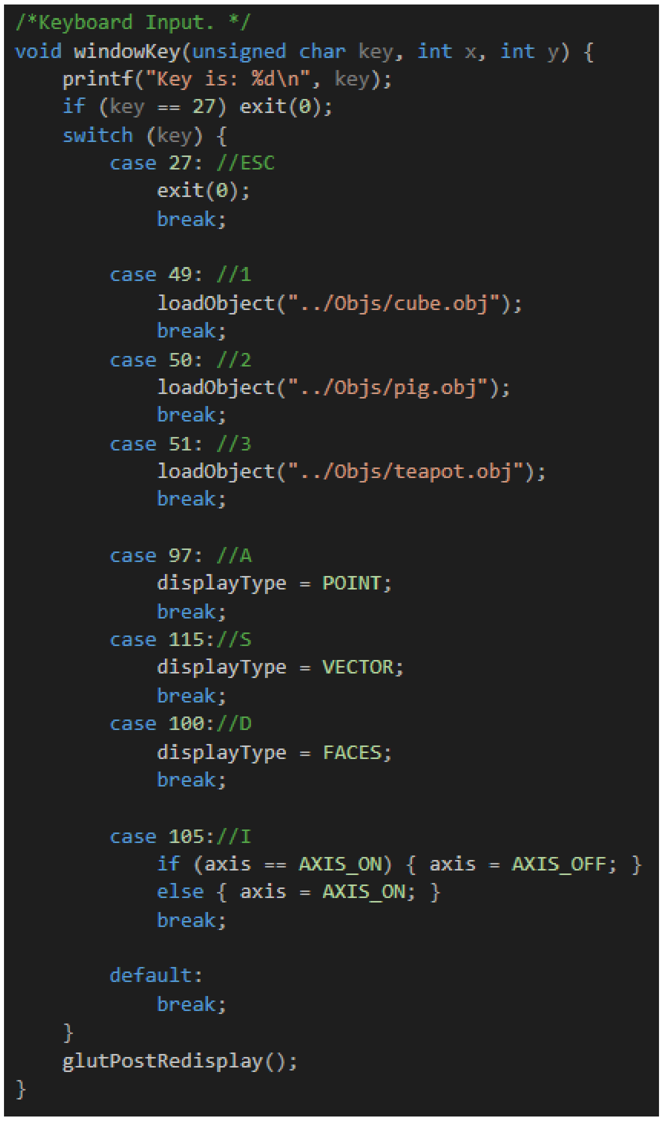
From there, depending on the set displayType, the object will be displayed accordingly (default is set to vectors because it looks really cool).

Reshape



```
/*Updates Display, Keeping Aspect Ratio if Window is changed*/
void reshape(int width, int height) {
    aspectRatio = (height > 0) ? (double)width / height : 1;
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    //printf("Ratio: %f %f | %f %f | %f %f\n", -dim*aspectRatio, +dim*aspectRatio, -dim, +dim, -dim, +dim);
    glOrtho(-dim*aspectRatio, +dim*aspectRatio, -dim, +dim, -dim, +dim);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

}
```

Our reshape function is very basic. It sets our Ortho view while also maintaining our aspect ratio.



```
/*Keyboard Input. */
void windowKey(unsigned char key, int x, int y) {
    printf("Key is: %d\n", key);
    if (key == 27) exit(0);
    switch (key) {
        case 27: //ESC
            exit(0);
            break;

        case 49: //1
            loadObject("../Objs/cube.obj");
            break;
        case 50: //2
            loadObject("../Objs/pig.obj");
            break;
        case 51: //3
            loadObject("../Objs/teapot.obj");
            break;

        case 97: //A
            displayType = POINT;
            break;
        case 115://S
            displayType = VECTOR;
            break;
        case 100://D
            displayType = FACES;
            break;

        case 105://I
            if (axis == AXIS_ON) { axis = AXIS_OFF; }
            else { axis = AXIS_ON; }
            break;

        default:
            break;
    }
    glutPostRedisplay();
}
```

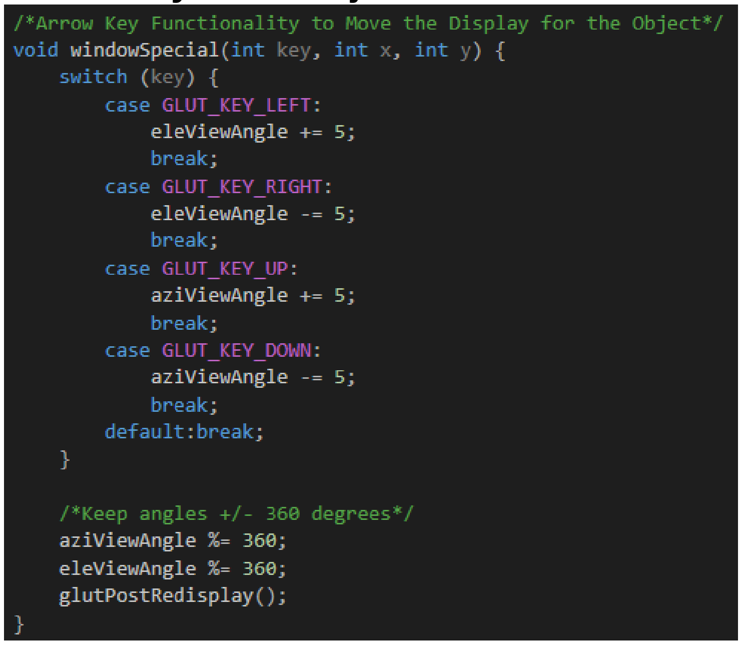
Keyboard Input
The Esc key simply exits our program.

Keys 1, 2, 3 swaps the object we will be displaying (Cube, Pig, Teapot respectively).

Keys A, S, D changes the display type of the object (Point, Vector, and Faces respectively).

Our I key toggle the XYZ grid on/off.

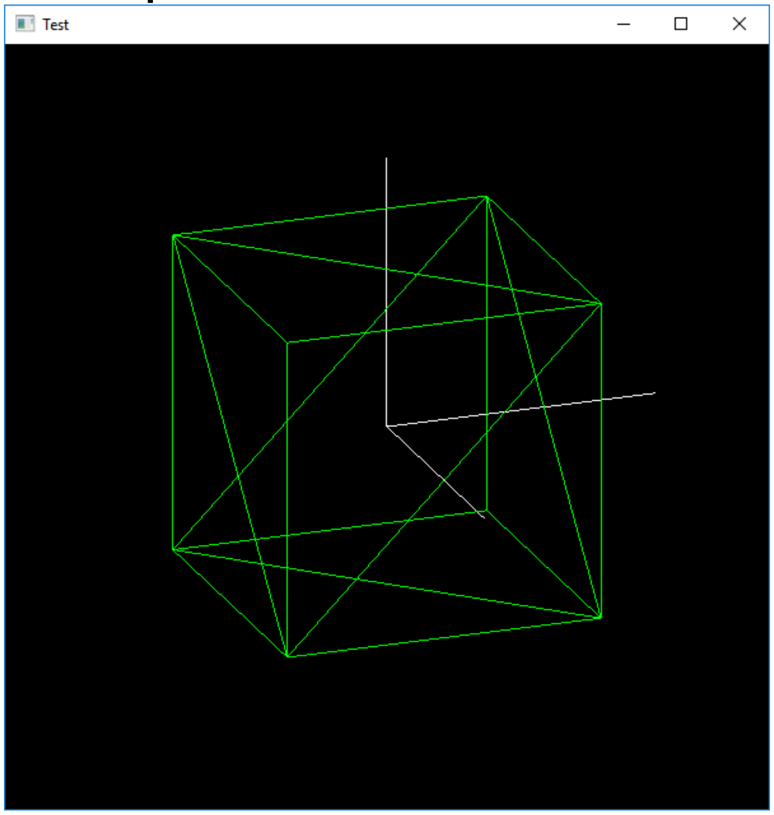
Arrow Keys and Object Rotation

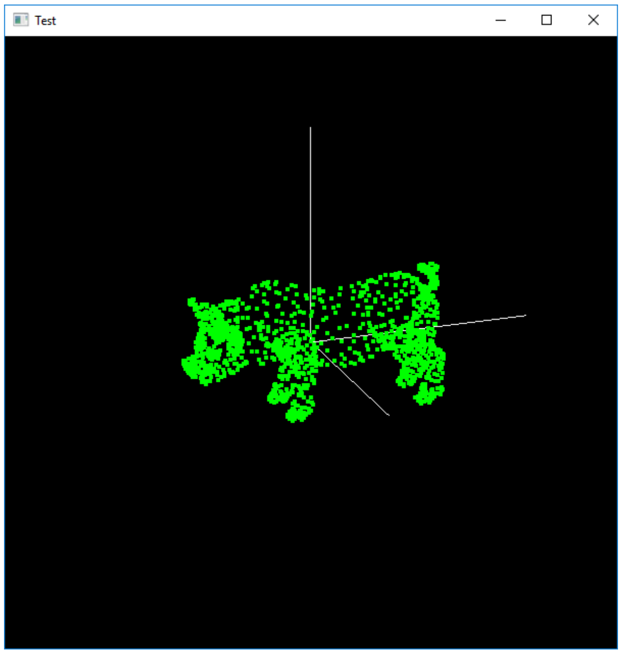


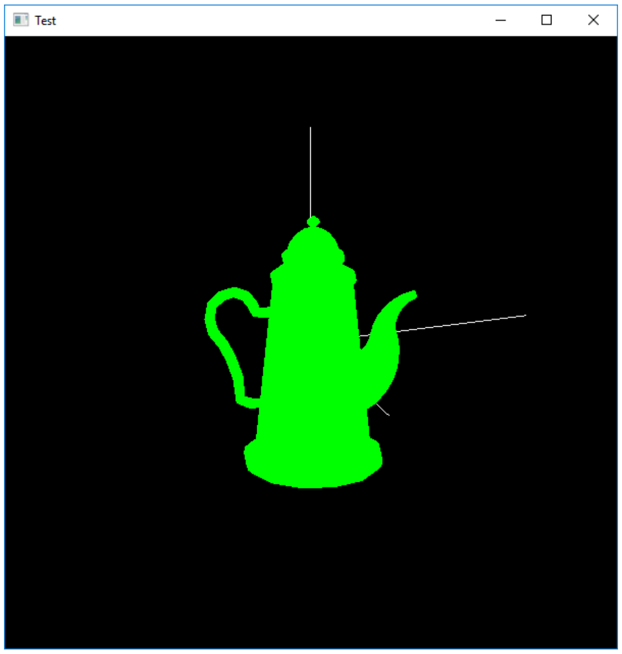
```
/*Arrow Key Functionality to Move the Display for the Object*/
void windowSpecial(int key, int x, int y) {
    switch (key) {
        case GLUT_KEY_LEFT:
            eleViewAngle += 5;
            break;
        case GLUT_KEY_RIGHT:
            eleViewAngle -= 5;
            break;
        case GLUT_KEY_UP:
            aziViewAngle += 5;
            break;
        case GLUT_KEY_DOWN:
            aziViewAngle -= 5;
            break;
        default:break;
    }

    /*Keep angles +/- 360 degrees*/
    aziViewAngle %= 360;
    eleViewAngle %= 360;
    glutPostRedisplay();
}
```

Pushing the arrows keys will rotate our objects around in space. Note both the grid and our object moves simultaneously because we did not pop/push our matrices.

The Output







The following are samples of outputs. Note how the grid helps us see how the objects was centered and how the points, vectors, and faces are displayed. Also notice how we can move our objects in space to get a better perspective of how the object is built.

I did not bump into any remaining issues. Possible future additions our be to instead of using the arrow keys to rotate our object, use the mouse dragging abilities.