Report 2

Created	@October 4, 2022 3:26 PM
Class	Graphics
Semester	Fall 2022

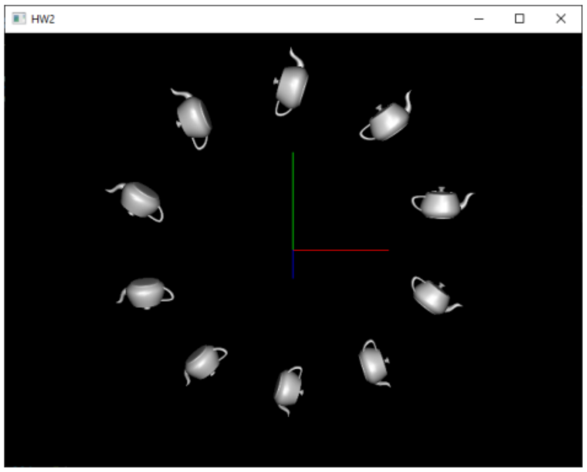
HW 2 Report

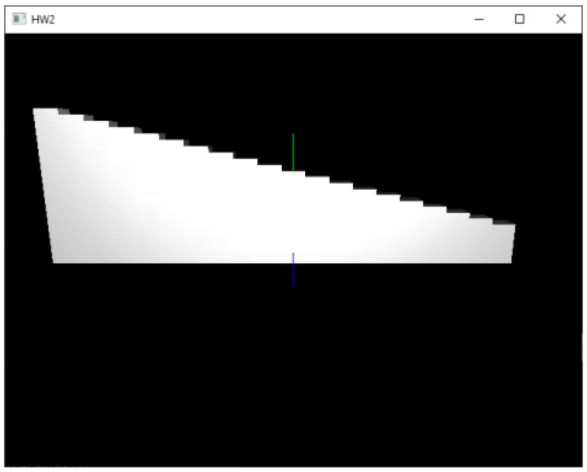
Anthony Ciocco

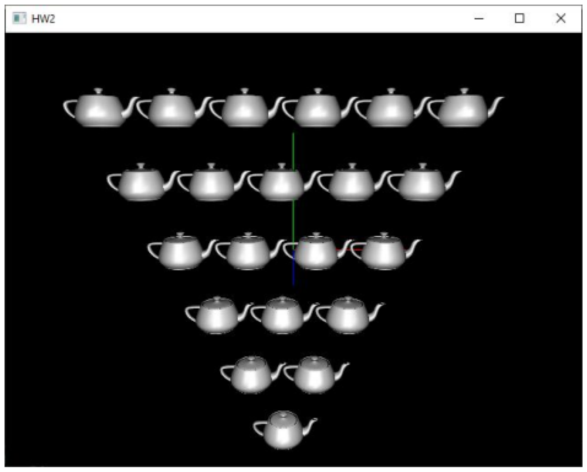
1600875

1. Problem

The assignment given was to recreate 3 images and produce a 4th of our own creation using the skills gained throughout the process. We were to use openGL glut shapes and primitives in order to recreate the images.







2. Method

We were to use openGL functions/primitives such as, `glutSolidTeapot()`, `glutSolidCube()`, `glTranslatef()`, `glScalef()`, `glRotatef()`, `glPushMatrix()`, and `glPopMatrix()` among others in order to recreate the scenes given. The way of doing so requires any individual objects, cube, teapot, etc, between `glPushMatrix()` and `glPopMatrix()`.

The solid objects are placed in space through a coordinate system. From there, the center of the object occupies that space and is build around it. This is important as scaling a translating the objects depends on the center coordinate that object defines. When translated, we have to adjust for the scaling and the initial size of the object we set.

3. Implementation

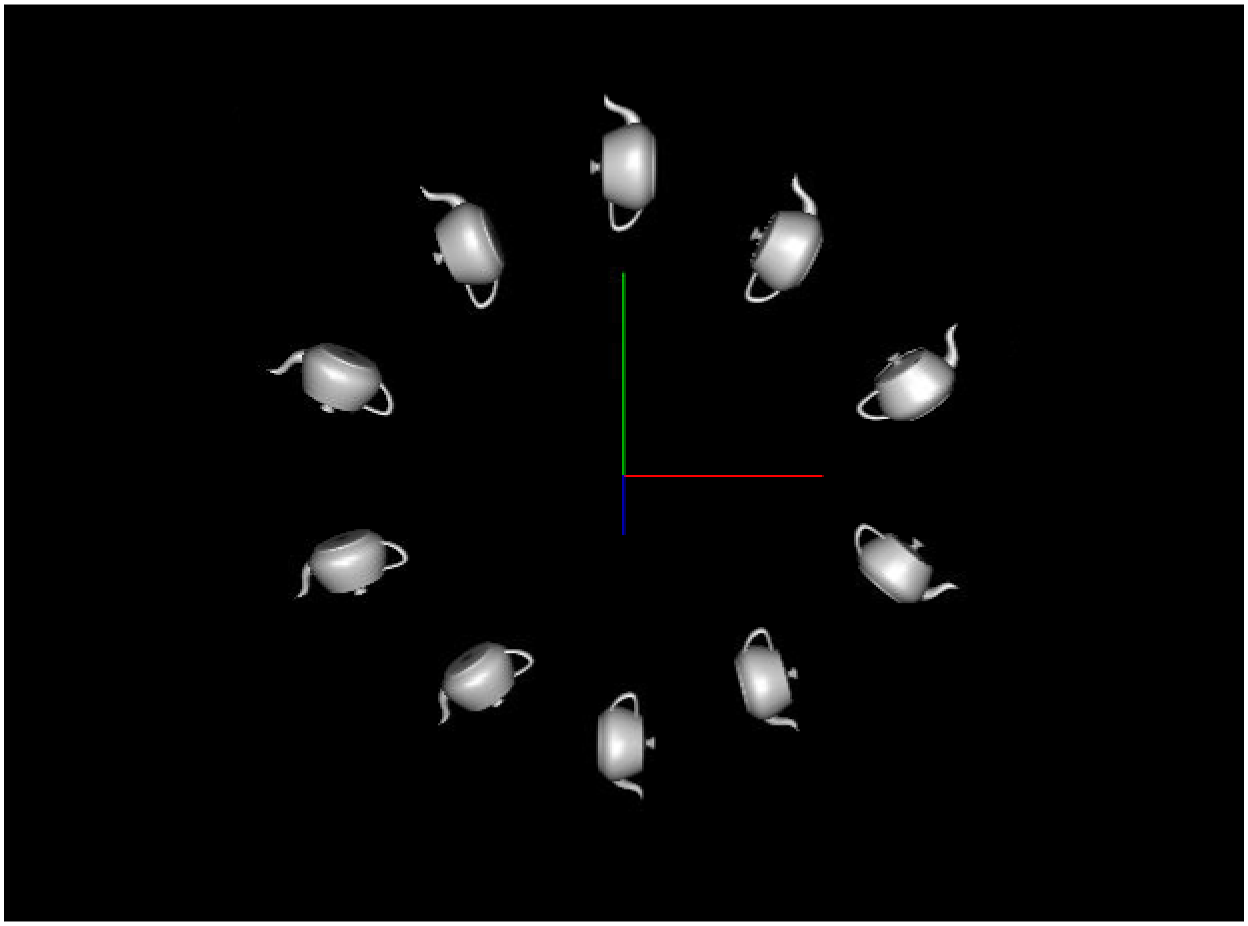
Four problems were given an each had to be handled differently, at least in my approach. Regardless all required the same structure with push → transformations → pop and repeat.

3.1

The first problem had us recreate 11 teapots that were oriented into a circle, with each teapot spout facing out perpendicular to the circle. This required code looking like this:

```
glPushMatrix();
glTranslatef(); //takes float x,y,z
glRotatef(); // takes float x, y, z, vector
glutSolidTeapot(); // takes a double size
glPopMatrix();
```

The specific values were hard coded in to replicate the pots. These values however could be placed into a for-loop in order to more easily replicate the pattern. This method requires a fine tuning of the geometry, and angles of every single pot. The hard coding was to save time.

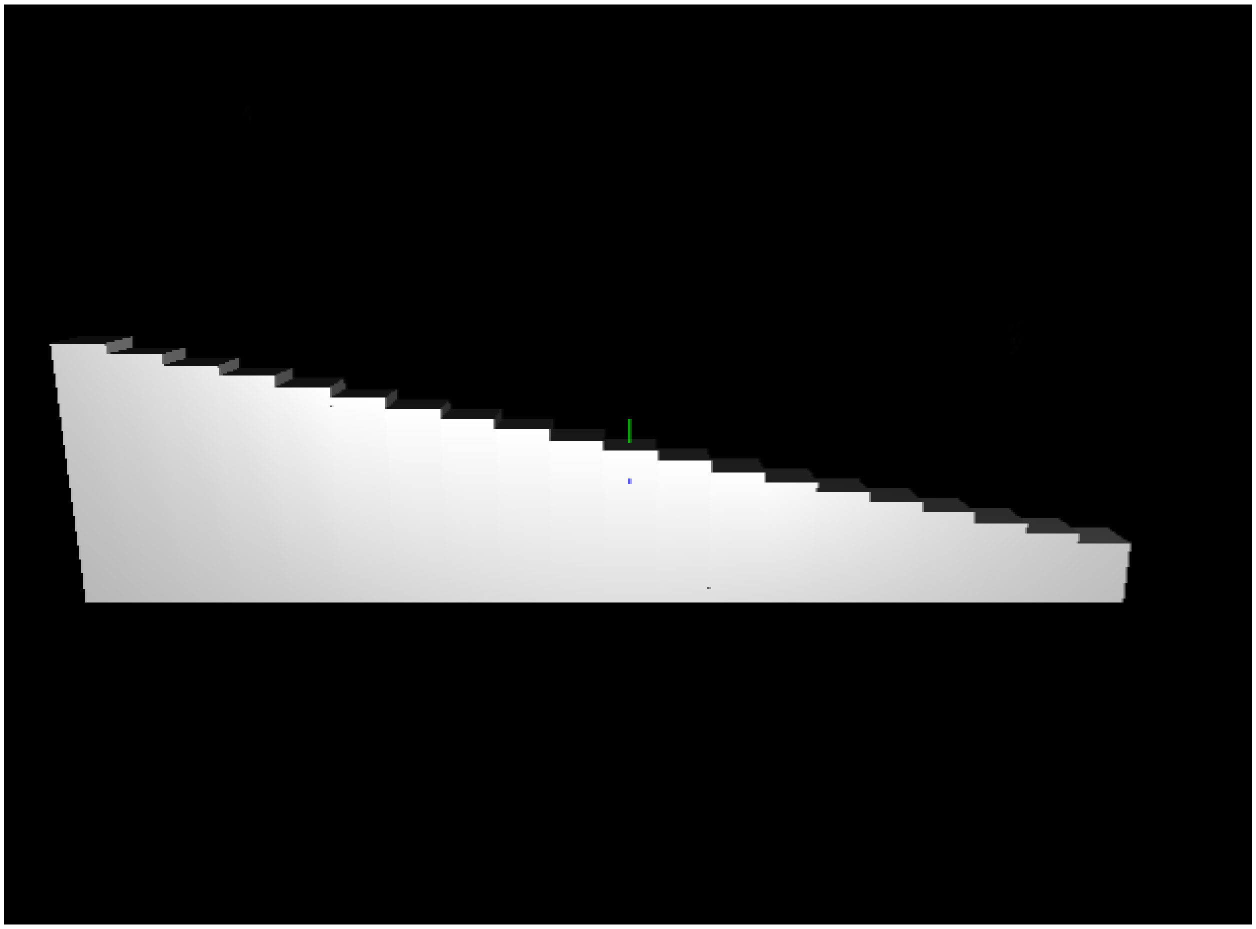


3.2

The next challenge had us replicating a stairwell. This was done using a for-loop using these functions:

```
glPushMatrix();
glTranslatef(); //takes float x,y,z
glScalef(); // takes float x, y, z
glutSolidCube(); // takes a double size
glPopMatrix();
```

The for-loop would create, translate to the right and down, then scale the cubes creating rectangular prisms. An arbitrary scaler and difference per step were chosen but the translation in the y direction had to be specific to that difference per step. Because the code reduces the scaler by 0.2 each iteration, the translation had to reflect that as well. The translation was to be moved down by half of the delta between the initial cube to the one right of it. That is, half of the scaler, 0.1, is how much the translation would push the next cube down. This would allow the scaled cubes to sit flush on one side, while creating a stair pattern on the other.



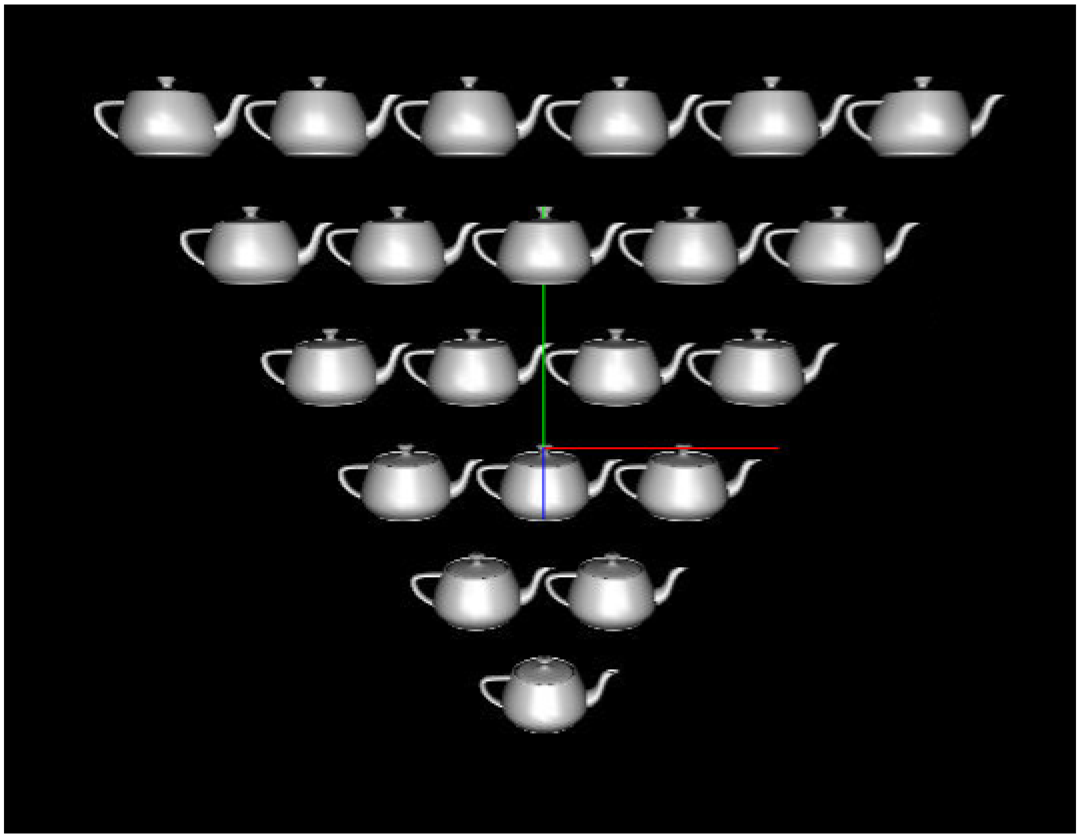
3.3

The next challenge had us replicating an upside down pyramid of teapots . This was done using a few for-loop to get a result. The functions used followed this format.

```
glPushMatrix();
glTranslatef(); //takes float x,y,z
glRotatef(); // takes float x, y, z, vector
glutSolidTeapot(); // takes a double size
glPopMatrix();
```

Here, 6 separate for loops were used. With the initial x translation pushing the the pots out to the left with x = -1.5, the for loop started to push them back with a translator pushing them over to right by increasing x each time. This was to keep in the symmetry intact for each row and column.

The translator moved each pot over by 0.6 into the right direction. Over the different for-loops, each loop was 1 less iteration than the last. This was from 6 to 1 in order to create the pyramid shape.



3.4

The last problem required us to use our creativity and make an image of our own using all the methods and any extras we wish. We were required to use a triangle and as can be seen, this triangle was used to create the face of the figure. Spheres were used as the ears and nose, along with the eyes. The cube, when scaled were used as the whiskers. The code was hardcoded to the the granularity needed for the face to look correct while keeping the implementation easy.

For the single triangle, the functions looked like this:

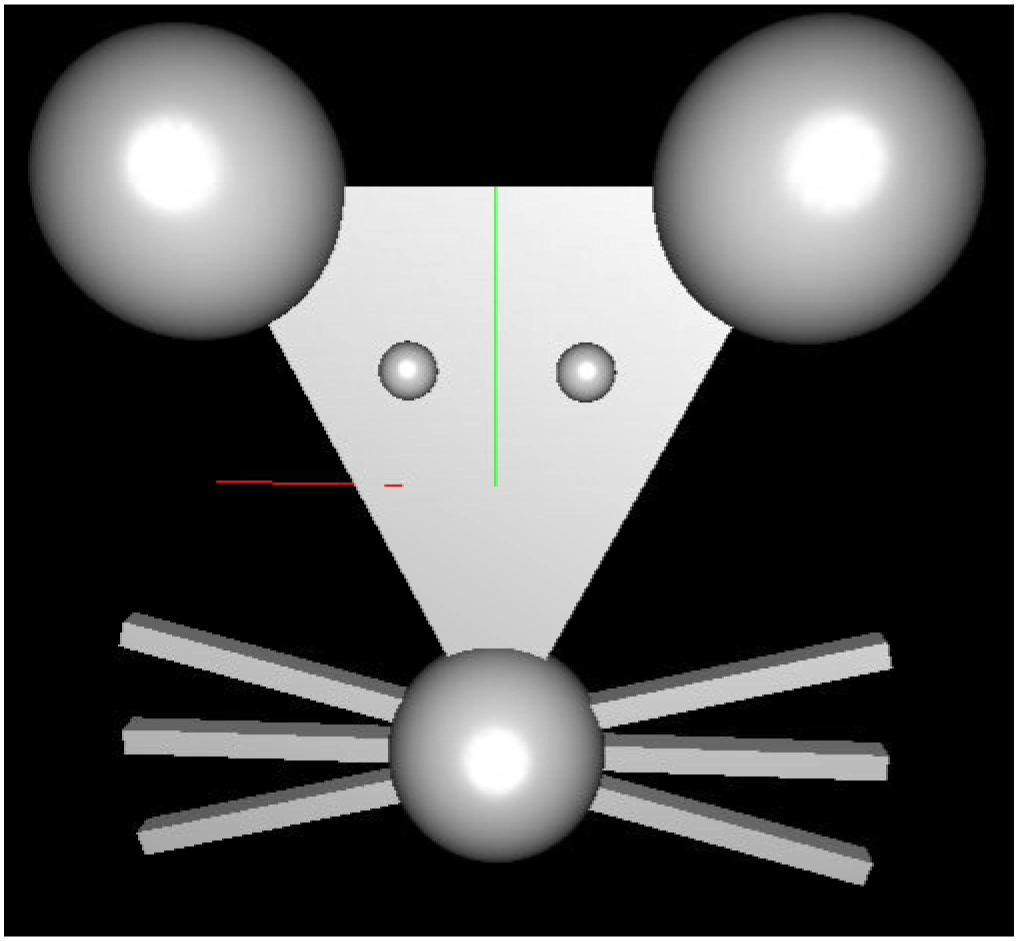
```
  glBegin(GL_TRIANGLES); // The shape being initalized
  glVertex3f(); // The vertexes for the specified shape, in this case 3
  glVertex3f();
  glVertex3f();
  glEnd(); // letting the stack know the end of the shape.
```

The spheres, 5 in total, followed this logic:

```
glPushMatrix();
  glTranslatef(); //takes float x,y,z
  glutSolidSphere(); //takes double radius, Glint slices, Glint stacks
  glPopMatrix();
```

Finally the cubes, 6 of them. looked something like this:

```
  glPushMatrix();
  glTranslatef(); //takes float x,y,z
  glRotatef(); // takes float x, y, z, vector
  glScalef();  // takes float x, y, z
  glutSolidCube(); // takes a double size
  glPopMatrix();
```



4. Result

The results are as follows and are 1 through 4. Some liberties and shortcuts were taken to recreate the initial images. However, the understanding of how and what openGL is doing is clear enough. While the specific values and images may not be exact, the idea behind the assignment, and use of the functions should seem clear.