COSC 4370 – Homework 2

Name: Austin Thibodeaux PSID: 2037865

October 2022

1 Problem

This assignment requires the completion of four problems, all of which involve various transformations of simple shapes in OpenGL. The first problem asks to arrange an array of teacup shapes into a circular formation. The second problem asks to make a staircase with the cube shape. The third problem asks to arrange an array of teacups into an upside-down triangle shape. Problem 4 asks to make any complex shape using various shapes and triangles using glPushMatrix/glPopMatrix and transformations.

2 Method

Four functions have been defined for this project:

- **problem1:** Draws 10 teapots in a circle along with a translation to the right and up. Each teapot is also rotated forward slightly on its local coordinates.
- **problem2:** Draws 20 cubes that decrease in height for each step of the staircase. The bottom of each cube is aligned to form a straight line, whereas the top of each cube decreases in height for each iteration.
- **problem3:** Draws 21 teapots in an upside-down triangle formation. Each teapot has a vertical and horizontal margin.
- **problem4:** Draws a sunflower using two cylinders representing the stem and base, and 16 petals using triangles. The sunflower sits on top of a “hill” rendered with a flattened sphere.

3 Implementation

Each implementation of these four problems utilizes variables declared at the beginning of each function that can be changed to configure the overall shape of the render. This allows each problem to be easily configured without getting into the meat of the implementation.

Each problem makes heavy use of the glPushMatrix() and glPopMatrix() functions provided by the freeglut-3.2.1 package. glPushMatrix() is used whenever a transformation of any kind is needed, otherwise it would tranform the whole scene. glPushMatrix can also be nested to, essentially, make a transformation within a transformation such as rotating a flower petal on the base of the flower, which has its own rotation transformation from the stem.

3.1 problem1

This implementation consists of four configurable variables: *numTeapots, radius, vOffset*, and *teapotTilt.* numTeapots allows you to set the number of teapots to render in the circular shape;

radius allows you to set the distance from the center of the world that the teapot revolves around; vOffset allows you to set the offset perpendicular to the line in-between the center of the world and the teapot; and teapotTilt sets the forward or backward tilt of the teapot.

For each teapot, a total of three transformation are applied within a single matrix. The first transformation rotates the teapot at the center of the world by increments of $\frac{360°}{numTeapots*teapotIdx}$. The second transformation translates the teapot to the right in the local coordinate system (radius) and translated up (vOffset). The third transformation tilts the teapot around the z-axis around the teapots center.

3.2 problem2

This implementation consists of four configurable variables: *startHeight, stepWidth, heightInc,* and *numSteps.* startHeight sets the height of the first step; stepWidth sets the width of each step; heightInc sets the change in scale each step iteration applies; and numSteps sets the number of steps to iterate.

Before the steps are rendered individually, the width of the staircase is computed which is then applied to a translation within a matrix to center it to the world. For each staircase, a nested matrix is created, and two transformations are applied. 1) The cube is translated to the right for each cube y the width of the stair. Each cube is also translated up by half of its height so the bottom lines up with the rest. 2) Each cube is scaled by the width of each step and the computed height (calculated by adding startHeight with heightInc).

3.3 problem3

This implementation consists of five configurable variables: *numRows, numObjsH, objSize, horizSpacing,* and *vertSpacing*. numRows sets the number of rows to render the triangle; numObjsH sets the number of teapots to render in the initial row; objSize sets the size of each object to render; horizSpacing sets the spacing in between each object horizontally; vertSpacing sets the spacing in between each row.

Before each teapot is rendered individually, the width and height of the triangle is computed which is then applied to a translation within a matrix to center it to the world in both x and y coordinates. For each row, the rowOffet is computed depending on the total width of the triangle minus the space the teapots in the row take up. This rowOffset it then used to translate the row to the right to make each teapot centered. Each teapot is also translated down in accordance to the spacing each row takes and the current row number it is.

3.4 problem4

This implementation consists of ten configurable variables:

1. **groundRadius:** Sets the radius of the ground that the flower sits upon.
2. **groundRoundness:** Sets how spherical the ground is. 0 is perfectly flat, 1 is a sphere.
3. **stemHeight:** Sets how high the stem of the flower is.
4. **stemRadius:** Sets how thick the stem of the flower is.

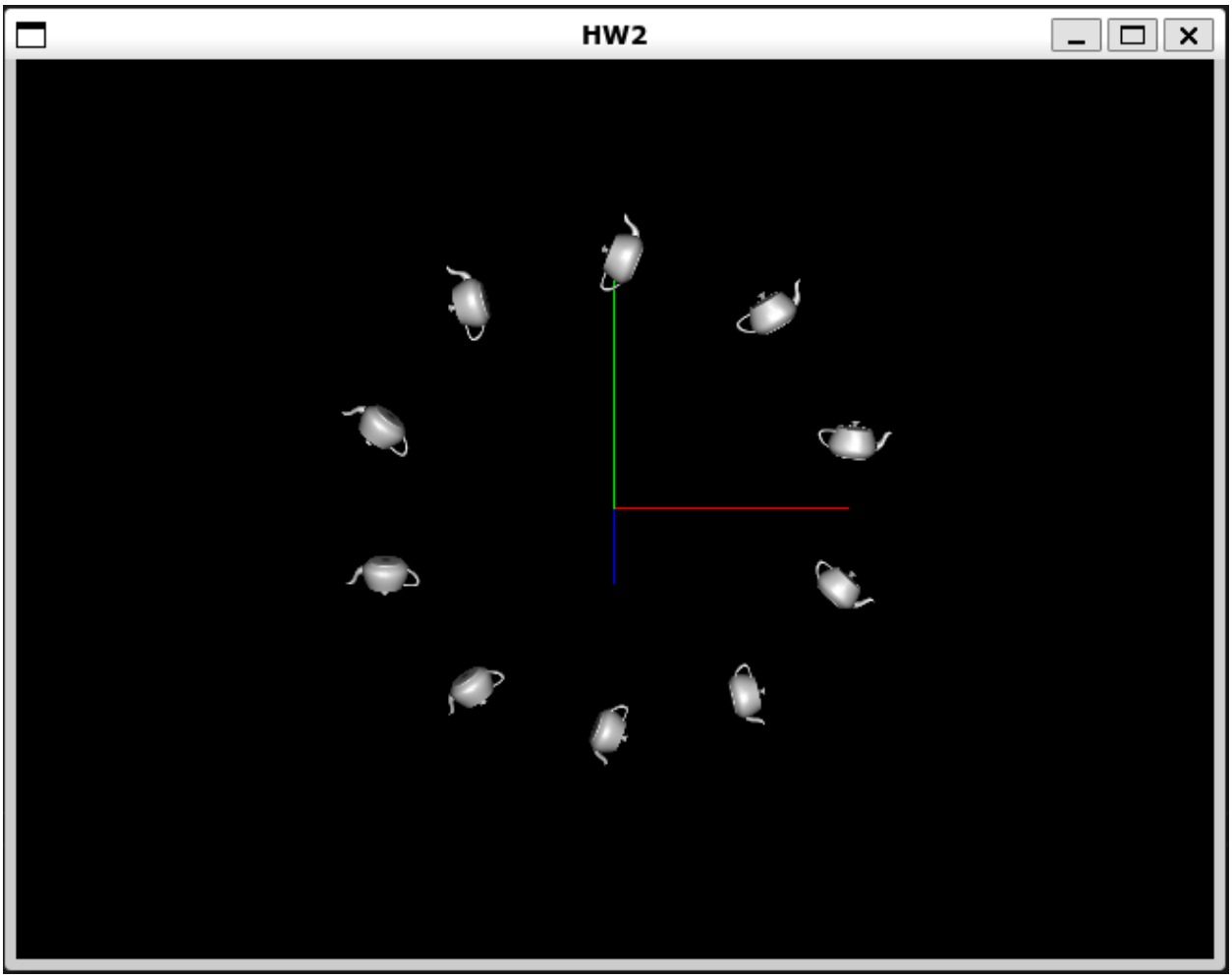
5. **flowerAngle:** Sets the angle the flower is to the stem.
6. **flowerBaseWidth:** Sets the width of the base that the flower petals are attached to.
7. **flowerBaseDepth:** Sets the depth of the base that the flower petals are attached to.
8. **petalWidth:** Sets the width of each flower petal triangle.
9. **petalHeight:** Sets the height of each flower petal triangle.
10. **numPetals:** Sets the number of flower petals on the flower.

This complex shape uses a maximum of three nested matricies. The first matrix translates the flower down by half of the stem height to ensure that it doesn't get cut off from the top. Then, the second layer of matricies draws the hill the flower sits on, then draws the stem and the base of the flower petals. Then the third matrix level draws the flower petals in a circle around the base of the flower using triangles.

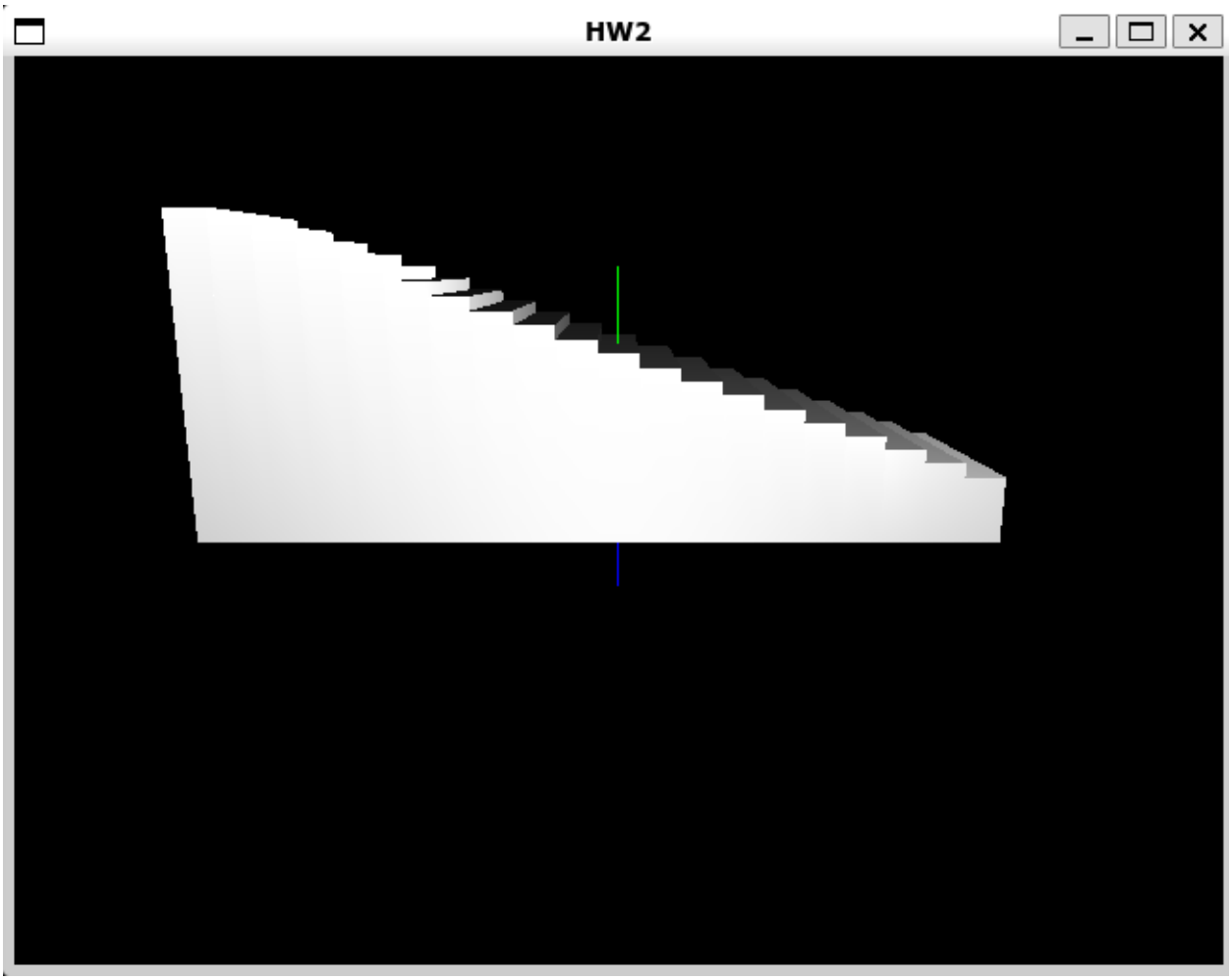
4 Results

Each of these implementations result in a render that is very close to the examples given in the assignment requirements. The fourth problem results with a render of a very rudimentary sunflower that displays how nested matrices can be used effectively.

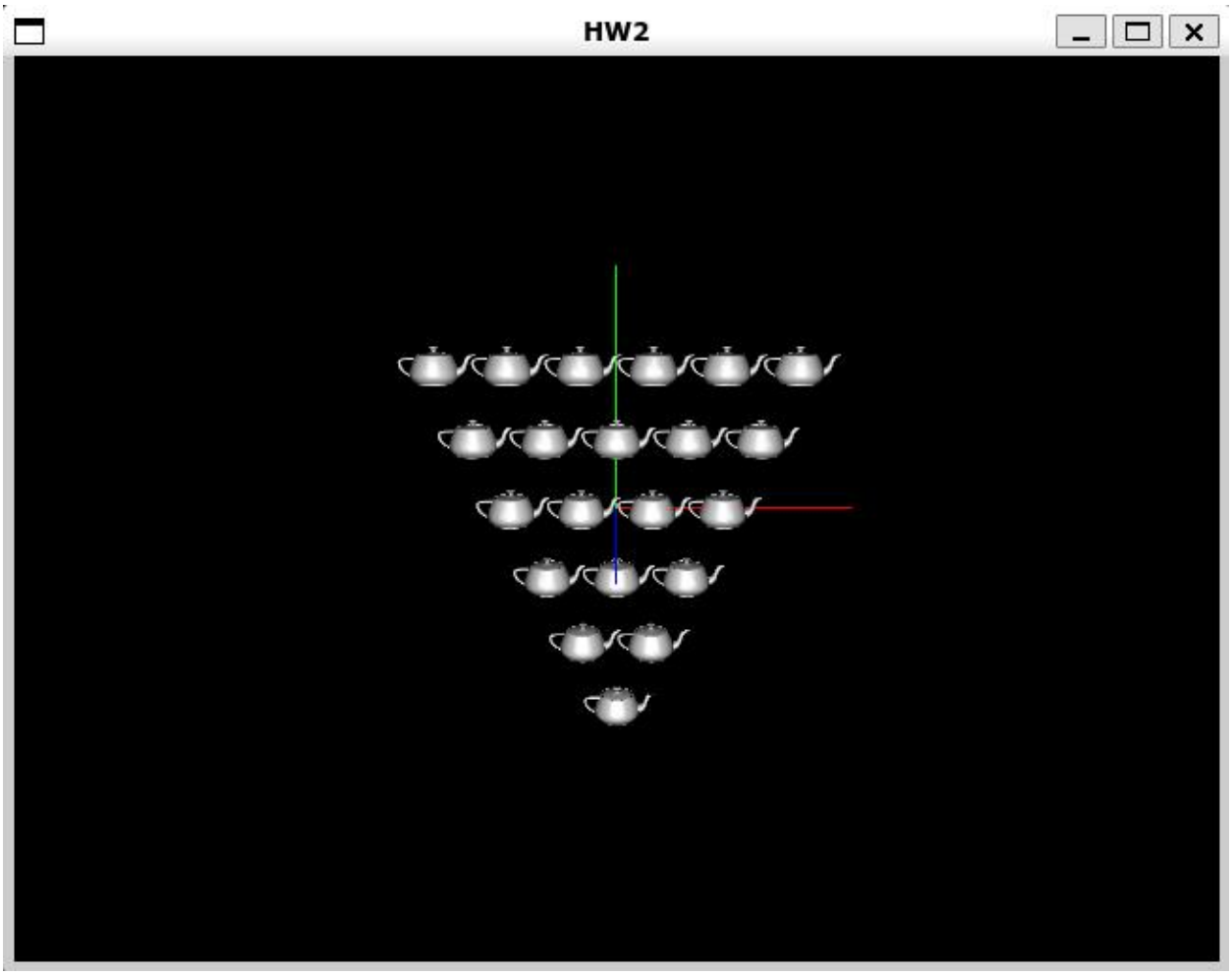
Problem 1 Result (Fig 4.1):



Problem 2 Result (Fig 4.2):



Problem 3 Result (Fig 4.3):



Problem 4 Result (Fig 4.4):