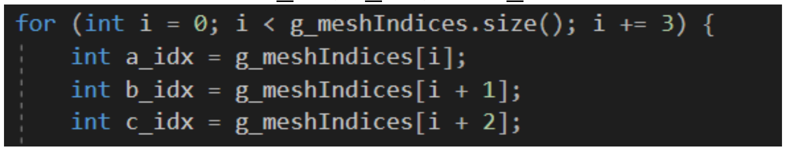
Andrew Thompson u0879848

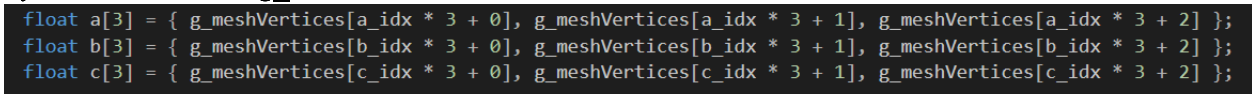
Part 1

- For starters, to find the normals for each vertex, we need to iterate through all triangles in the mesh. I do this by getting three values at a time from g_meshIndices, since all three points on every triangle are next to each other in the g_meshIndices vector. I call the three vertex indices a_idx, b_idx, and c_idx.



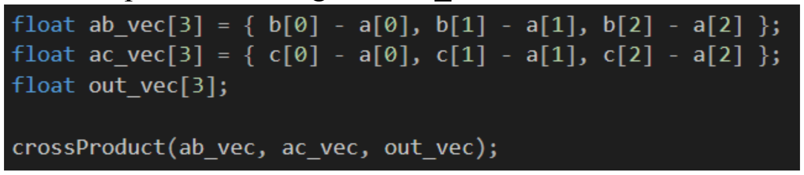
```
for (int i = 0; i < g_meshIndices.size(); i += 3) {
    int a_idx = g_meshIndices[i];
    int b_idx = g_meshIndices[i + 1];
    int c_idx = g_meshIndices[i + 2];
```

- I then find the x, y, and z components of these vertices using the index and multiplying by three to traverse g_meshVertices.



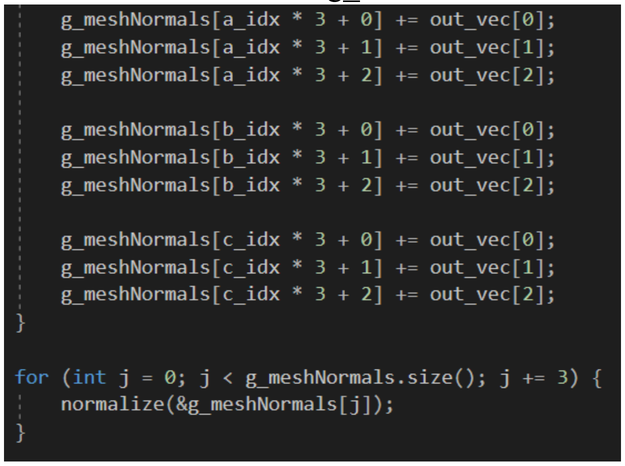
```
float a[3] = { g_meshVertices[a_idx * 3 + 0], g_meshVertices[a_idx * 3 + 1], g_meshVertices[a_idx * 3 + 2] };
float b[3] = { g_meshVertices[b_idx * 3 + 0], g_meshVertices[b_idx * 3 + 1], g_meshVertices[b_idx * 3 + 2] };
float c[3] = { g_meshVertices[c_idx * 3 + 0], g_meshVertices[c_idx * 3 + 1], g_meshVertices[c_idx * 3 + 2] };
```

- Next, I find the normal value for the triangle by finding vector AB, vector AC, and taking the cross product, storing it in out_vec.



```
float ab_vec[3] = { b[0] - a[0], b[1] - a[1], b[2] - a[2] };
float ac_vec[3] = { c[0] - a[0], c[1] - a[1], c[2] - a[2] };
float out_vec[3];

crossProduct(ab_vec, ac_vec, out_vec);
```

- Finally, I go to each vertex and add the normal vector to it. To average them, I then normalize each vertex in g_meshNormals in a separate for loop.



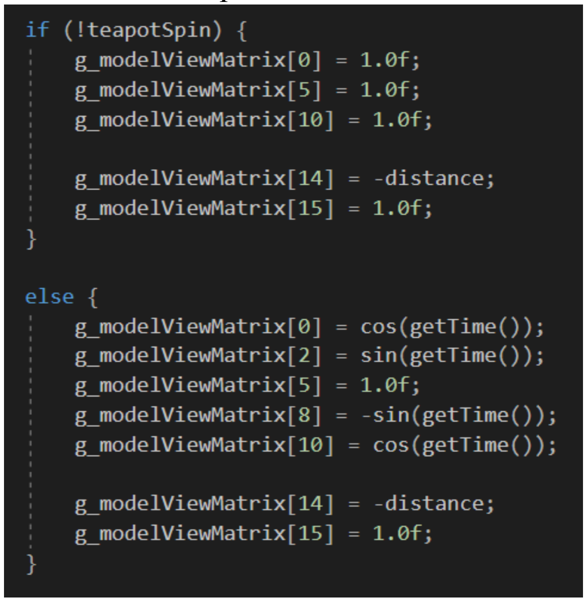
```
    g_meshNormals[a_idx * 3 + 0] += out_vec[0];
    g_meshNormals[a_idx * 3 + 1] += out_vec[1];
    g_meshNormals[a_idx * 3 + 2] += out_vec[2];

    g_meshNormals[b_idx * 3 + 0] += out_vec[0];
    g_meshNormals[b_idx * 3 + 1] += out_vec[1];
    g_meshNormals[b_idx * 3 + 2] += out_vec[2];

    g_meshNormals[c_idx * 3 + 0] += out_vec[0];
    g_meshNormals[c_idx * 3 + 1] += out_vec[1];
    g_meshNormals[c_idx * 3 + 2] += out_vec[2];
}

for (int j = 0; j < g_meshNormals.size(); j += 3) {
    normalize(&g_meshNormals[j]);
}
```

Part 2

- In order to rotate the teapot about the y axis, I just had to use the rotation matrix for three dimensions (no adjustments needed to be made since the object is at the origin.) I found a reference to this on https://en.wikipedia.org/wiki/Rotation_matrix.

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

- Given the default settings, it seemed that g_modelViewMatrix was a 4x4 matrix, and I assumed that the transformation matrix for 3d was found in the upper left. After testing, I found that this was the case. So using the matrix above (where the top row is has indices 0, 1, 2, the middle row has 4, 5, 6, and the third row has 8, 9, and 10), I just changed the values in the corresponding indices, using getTime() for theta to have it change rotation over time. I also put in a conditional to have it only spin after pressing the "s" key.

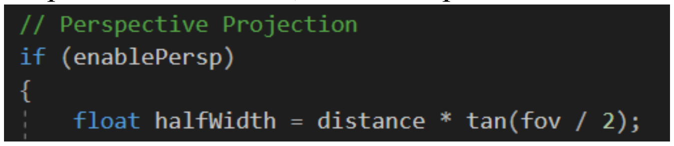


```
if (!teapotSpin) {
    g_modelViewMatrix[0] = 1.0f;
    g_modelViewMatrix[5] = 1.0f;
    g_modelViewMatrix[10] = 1.0f;

    g_modelViewMatrix[14] = -distance;
    g_modelViewMatrix[15] = 1.0f;
}

else {
    g_modelViewMatrix[0] = cos(getTime());
    g_modelViewMatrix[2] = sin(getTime());
    g_modelViewMatrix[5] = 1.0f;
    g_modelViewMatrix[8] = -sin(getTime());
    g_modelViewMatrix[10] = cos(getTime());

    g_modelViewMatrix[14] = -distance;
    g_modelViewMatrix[15] = 1.0f;
}
```

Andrew Thompson u0879848

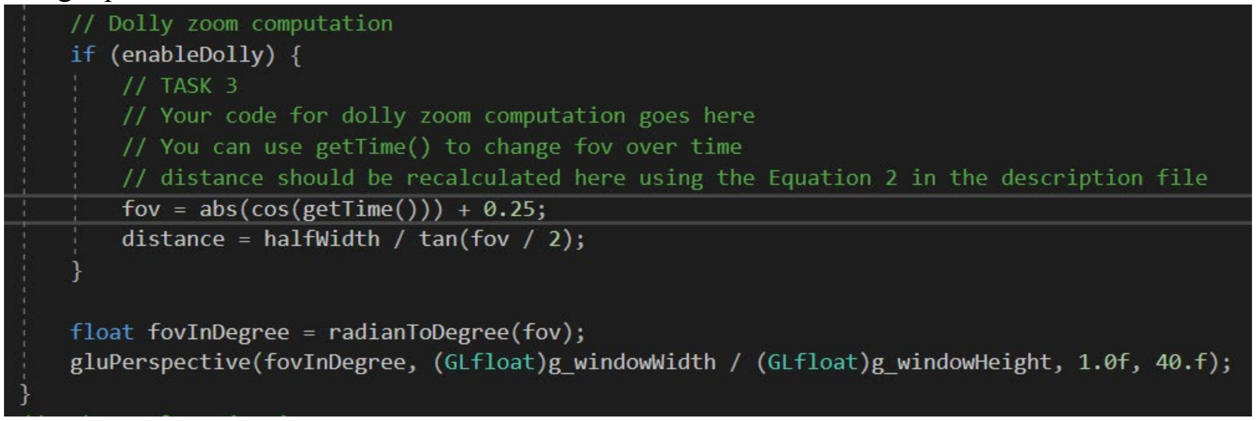
<u>Part 3</u>

- As per the instructions, I first computed halfWidth using equation 1.



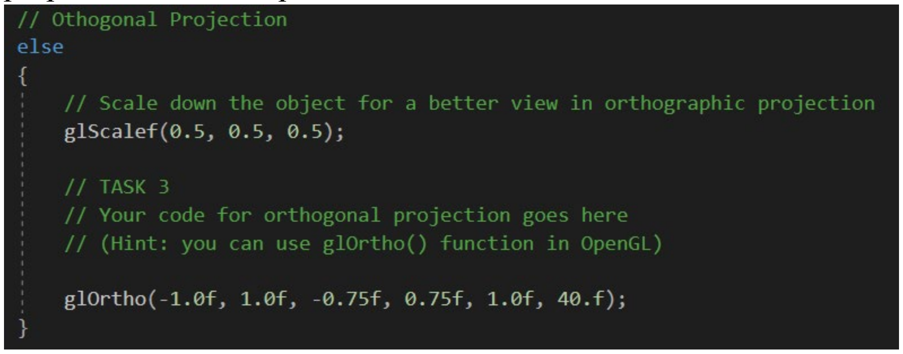
```
// Perspective Projection
if (enablePersp)
{
    float halfWidth = distance * tan(fov / 2);
```

- I then change fov over time using cos(getTime()), similar to part 2. I take the absolute value of that to ensure I only get positive values and assign it to fov. Since this can occasionally give me values close to zero (causing distance to become greater than zFar), I add 0.25 to ensure distance will stay within the bounds. Finally, I compute distance using equation 2.



```
    // Dolly zoom computation
    if (enableDolly) {
        // TASK 3
        // Your code for dolly zoom computation goes here
        // You can use getTime() to change fov over time
        // distance should be recalculated here using the Equation 2 in the description file
        fov = abs(cos(getTime())) + 0.25;
        distance = halfWidth / tan(fov / 2);
    }

    float fovInDegree = radianToDegree(fov);
    gluPerspective(fovInDegree, (GLfloat)g_windowWidth / (GLfloat)g_windowHeight, 1.0f, 40.f);
}
```

- For the orthographic projection, I used the hints provided and just used glOrtho. After testing with left = -1.0f, right = 1.0f, bottom = -1.0f, and top = 1.0f, I found that gave a squashed result. Seeing that the window width was 800 and the height was 600, I figured I need to make the bottom and top 0.75 times the left and right values. After doing so, the proportions of the teapot were corrected.



```
// Othogonal Projection
else
{
    // Scale down the object for a better view in orthographic projection
    glScalef(0.5, 0.5, 0.5);

    // TASK 3
    // Your code for orthogonal projection goes here
    // (Hint: you can use glOrtho() function in OpenGL)

    glOrtho(-1.0f, 1.0f, -0.75f, 0.75f, 1.0f, 40.f);
}
```

Youtube link: https://youtu.be/h4Znrfhc5ms