Computer Graphics Tutorial 2

In this tutorial, we will introduce some basic 3D graphics using javascript and WebGL. We will load in a simple 3D model, display it using shaders, and transform it via rotations and perspective projections.

Running the Code

- Extract the given zip file from the course webpage (`http://www.inf.ed.ac.uk/teaching/courses/cg/index2017.html#Tutorials`).
- Open "tutorial_2.html" in a web browser.
- If using Google Chrome, you must open it via the command line with `google-chrome -allow-file-access-from-files`. This extra flag allow the 3D model file to be loaded from the local filesystem.
- If using Google Chrome, helpful debugging tools can be opened by pressing `F12`.
- If using Firefox, debugging tools can be opened by pressing `Ctrl + Shift + K`, or by installing the Firebug extension.

Questions

1: Loading the 3D Model

We start by loading in a 3D model of a cube from a simple .OBJ file. If you open the included "cube.obj" file in any text editor, you will see that there are two main types of lines. Those starting with "v" define the positions in 3D coordinate space of the vertices that make up the cube:

```
v 1.000000 -1.000000 -1.000000
```

Lines starting with "f" define how the cube's vertices combine together in triangles to form faces. Each numbers listed here is an indices into the list of vertices previously specified by the "v" lines. NOTE: This indexing scheme starts at 1.

f 2 3 4

When rendering the cube, we will eventually use the `gl.drawArrays` function with the `gl.TRIANGLES` setting. This tells the GPU to render an array of arbitrary triangles using 3 sets of 3D coordinates per triangle (9 floating point values).
Your task is to complete the missing section of the `loadMeshData` function so that the `vertices` array contains a continuous list of 3D coordinates for every triangle in the mesh. Each triangle has 3 vertices, which have 3 coordinates each, there should be 9 floating point values per triangle.
If you have done this correctly, you should see a message saying 36 vertices were loaded, and the `vertices` array should contain 108 values.

2: Perspective Projection

Next, we will add perspective projection so that the cube we just loaded will appear correctly on screen. Complete the `perspective` function so that it returns the correct 4D projection matrix given the values for top t, right r, and near plane n and far f:

$$\begin{bmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

You may wish to use the `arrayToMat4` or `createMat4` helper functions for this to ensure the matrix's contents are of the correct format.
After completing this, you should be able to see a red square in the centre of the screen. Changing the FOV slider should make it appear closer or further away as the perspective changes.
NOTE: WebGL's matrices are layed out in column-major order, so the indices may not be in the order you expect. They run in this order:

$$\begin{bmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{bmatrix}$$

3: Translation

Now that the cube is visible, we can move it around to see the 3D perspective effect we just implemented. We can do this by multiplying its `modelMatrix` by a rotation transformation matrix.
Implement the function `translate` to return the correct translation matrix:

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

When this is completed you can move the cube around using the sliders.

4: Rotation

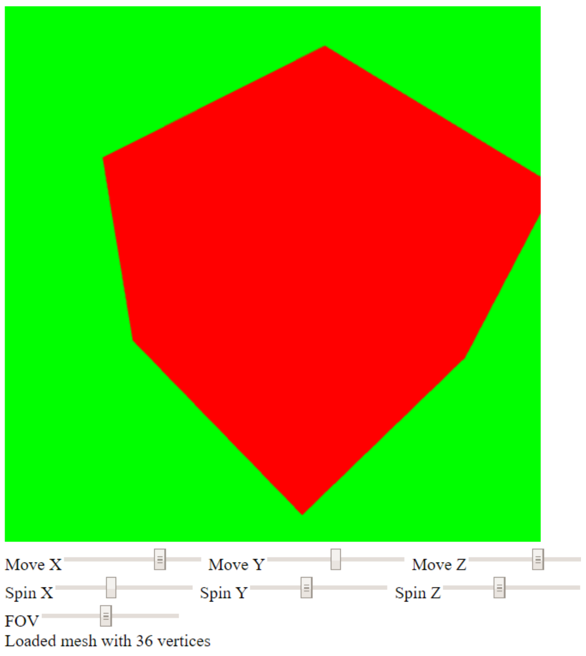
Implement the rotation functions `rotateX`, `rotateY`, `rotateZ` so that they return the correct rotation matrices for each axis (see the lecture slides).

5: Rotation about the Origin

To rotate the cube properly, we will need to rotate it around the origin point $(0, 0, 0)$ rather than simply rotating it in place. To do this, we will need to apply a translation matrix to move it to the origin, rotate it, and then apply the inverse of this translation matrix to move it back.
Implement the `applyXYZRotationTo(m, thetaX, thetaY, thetaZ)` function such that the correct rotation transformation is applied to it:

$$M = T^{-1} \cdot R \cdot T \cdot M$$

where T is the translation matrix that moves the cube from its position to the origin, and R is the rotation matrix $R_x \cdot R_y \cdot R_z$ which applies rotations about each axis in order. If you complete this, the 3D cube will spin whenever you alter the X, Y, or Z sliders.



Loaded mesh with 36 vertices

Figure 1: The completed cube model spinning.