Computer Graphics Assignment

Assignment Specification

In order to make almost all of the objects in my scene, I transform, scale and rotate the model matrix and apply these transformations to the unit cube that I created using the makeCube function. (The makeCube function acts as a constructor and returns a cube object, with the relevant buffers as properties). The model matrix is defined in g_modelMatrix, and view matrix and projection matrices are combined into the viewProjMatrix (since only one transformation is performed on each per draw).

I defined a function to make textured cubes, which is essentially a constructor that acts as a subclass of makeCube, with the added properties that the shapes can also be textured by adding texture coordinate buffers to the image and loading an image from disk into the image.

I also defined a generalised function that could create any sided shape using polar coordinates and a series of triangles. However, I realised that I actually had little design use for them, so I attached a range of shapes to the right hand side wall. It can, in fact, create a shape of any number of sides.

I defined a range of keys that can be used to interact with objects in the scene, such as turning lights on and off, transitioning between powerpoint slides, and moving the camera around the room. Additionally, the camera can be rotated around by holding down the mouse and moving the cursor around. A full description of the keys can be found beneath the scene.

I also wrote the shaders such that the intensity of the light source is proportional to the inverse of the square of the distance from the light source (as with real light). This ensured that objects far from the sources are lit less than those close by.

Instead of performing key actions immediately when a key is pressed, the keydown function actually sets that key to be true in a map, and performs the corresponding actions for the keys that are set to true on each draw. The corresponding values are set to false when the key is released. This allows the user to press multiple keys at once.

The drawshapes function represents the scene graph, since the all of the model transformations occur in this method.

The vertex and fragment shaders are defined outside of the main javascript file, in vshader.gl and fshader.gl, and loaded in using XMLHttpRequests. This makes it easier to write and modify the shader programs.

Since the lookat method defined in the Matrix4 library defined a 3d point for the camera, and a 3d point to look at, I defined two arrays, ‘eye’ and ‘look’, such that ‘look’ acts as a vector by always setting the camera to look at as the the ‘eye’ array plus the ‘look’ array, essentially making ‘look’ a vector to point at relative to the current position. This allowed camera rotations easier to manage, by using polar coordinates to define the vector and varying these angles to change direction.

I use four diffuse light sources in my lecture hall (which can be switched on and off during the ‘day’ time) and also have ambient lighting which varies according to the ‘time of day’.

Limitations of the implementation

Due to the large number of objects included in my Scene, the performance is somewhat low when using browsers that do not allow for use of the dedicated graphics card (e.g. Chrome sometimes can only perform software rendering, or in the case of dual graphics card notebook, cannot access the dedicated graphics card). Most of my testing was performed in Firefox, which I recommend is used to run my implementation.

Another limitation I could have designed the models in an external modelling tool, such as blender, to create more realistic models, which would improve the appearance of the scene.

Also, no shadowing currently exists, though the library does allow for this, and given more time this could be easily implemented.

Descriptions of the attached screen shots

Screenshot 1: This screenshot shows the previously mentioned lighting diffuse lighting effect with just one light active (note the area around the light is lit more due to the inverse square effect). Additionally, the three whiteboards and computer screen can be seen, and the chairs are visible to the left. The floor tiling is texture mapped with a single texture on repeat.

Screenshot 2: This screenshot is outside of the lecture hall whilst all of the lights are on, and shows the grass repeating texture, as well as the ‘sun’ almost at it’s highest point, which is when the ambient light is at its greatest value. The different perspective also demonstrates the ability to move the camera around.

Screenshot 3: Of course, when faced with government funding cuts, the university must raise additional money by converting the lecture hall into a disco hall during the night time. In order to make my lecture hall more realistic, I included this as a feature.

Additional Features

I enabled mipmapping in order to reduce memory use and perform anti-aliasing on the textures, but I quickly discovered that many of the textures were poorly sampled and appeared blurry when mipmapping was implemented. In order to combat this, I enabled anisotropic filtering, which significantly improved the appearance of the textures in the hall, as well as the grass texture applied to the floor.

Public Domain Source Code

Many parts of my code are derivations of code from the WebGL Programming Guide textbook, which I referred to extensively whilst developing my own scene: https://sites.google.com/site/webglbook/