CS606 Computer Graphics
Spring 2020
Agam Kashyap
IMT2018004

Rendering and Manipulation of 3D models

Problem Statement

The given task is to render 3D models built using software such as Blender, to perform manipulations on them such as translation, rotation, scaling. Furthermore, we also need to implement a mouse drag based camera rotate system. The final task is to select the objects as a whole and select their faces individually.

Approach

I will first explain the tasks, and the setup involved with respect to the Model matrix, and then go on to explain the Projection and View matrix, since they are common to every object in the scene in this case.

Axes of the Scene

For rendering the three axes, I created a single arrow mesh in Blender positioned along the X axis. So, now my task was to first import the OBJ file to create the Arrow object in webgl and rotate the Arrow into the Y and Z axes.

I used the `fetch()` function to import the `.obj` files. `fetch` is and **asynchronous** function, which means that the program flow doesn't wait for its completion. So, to prevent facing errors, I setup booleans inside each of the fetch functions for all objects. Before calling the `draw` function on each of these objects, I use these booleans to verify that indeed the files have been read and processed. I used the `webgl-obj-loader`[1] to extract a Mesh object from the `OBJ` file.

So now I had one arrow aligned along the positive x axis. To create the Y Axis, I rotated the arrow about the z axis by $+\pi/2$ and similarly, to create the Z Axis I rotated the arrow about y axis by $-\pi/2$. At beginning of the scene or in the *Axes* mode, the scene Axes will be positioned at the origin.

Three 3D objects and Transformations

The `.obj` file importing was the same method as mentioned before, but certain modification that I made at the final stage, was splitting the mesh into several sub-meshes. This was done to facilitate the face selection feature for the object. I will explain the my approach to that in detail in upcoming sections. So, to make the code modular and easier to perform actions on the objects, I created a class called `SuperMesh` which consisted of an array of all the different face objects which combined

[1]https://www.skypack.dev/view/webgl-obj-loader

to form the final complete object. All my actions -draw, translate, rotate, etc. were performed by using the `SuperMesh` objects which consists of helper functions which call the corresponding methods on the sub-meshes.

The three objects that I created for this assignment were- Cube, Torus, Random, wherein the Random object is an abstract 3D figure.

The first two tasks here were to translate these objects to various points on a triangle(either the corner or the mid-point of the sides). In these two modes - `Triangle Corner` and `Triangle Side`, and the remaining modes as well, I translated the Axes made in `Axes` mode to the right corner of the canvas, to provide the user with an intuition about the coordinates. I created a Triangle at the center of the canvas, which is the origin in this case. After that, all that was required to be done, was to calculate the coordinate of the points - either the corners or the page centers, and pass them as translation values to the objects, since the objects by default are getting drawn at the center of the canvas.

The next task was to scale. I scaled the Cube by 0.5, Random by 2 and Torus by 3. I also added an extra option to reset the scaled objects to their original scale.

Question 1: To what extent were you able to reuse code from Assignment 1?
As I mentioned until now, the transformation tasks involved did not require any special mechanisms to be handled. The same code that we used before, that was implemented for 3D systems itself, worked perfectly fine for the Model matrices. Furthermore, the class that we created for generating the Meshes remained almost unchanged, with the exception of vertex coordinates and the indices, which we are now getting from the `.obj` files. The `draw()` function remained the same with the exception of the Projection and View matrices. That brings me to the shaders. The fragment shader was the same, but the vertex shader was modified a bit to include the projection and view matrices as well. The last change that was required was in the `Renderer`, wherein we changed the coordinates which earlier used to go from -1 to 1, now corresponds to the maximum resolution of the window.

The next task was to rotate the objects about different axes by 90°. This utilised the same transformation matrix as the previous assignment.

View Matrix and Projection Matrix

This was one of the places where I faced some issues setting it up. I decided to keep perspective projection rather than orthographic projection, since it provided a clearer sense of the objects. The first issue that I faced was that, as soon as I was moving the camera away along X axis, irrespective of the value, the objects would all come along the Y axis, flipped about their position and the axes that I was drawing on the upper right corner of the scene got flipped onto the other side as well. The reason for this issue was that I had placed the camera or the eye at the origin itself. So when I move the camera by any value on either side, I would be getting the side view of the whole scene. Hence it looked the same irrespective of the changes I did. Upon discussion with the TA, I realised that we had to rotate the camera around the scene. So, I set a circular path in the XZ plane for the camera to move around. To achieve this, I considered the positive Z axis as $0°$and rotation by θ, I calculate the X coordinate as $Rsin(\theta)$ and the Z coordinate as $Rcos(\theta)$.

To include the Camera and Projection in our program, I changed the vertex shader to include the them as well. The final value of `gl_Position` is equal to Project*View*Model matrix in that order. After doing this, one issue that I faced was that certain objects didn't seem to be 3D, in that, when they were supposed to be at the back of certain objects, they would be rendered in front of them. Since webgl renders the objects in which they were called, the objects don't appear to follow the rules. To deal with this issue, we utilise the `gl.DEPT_TEST` which verifies, if the final output that is being rendered follows the depth based property that it has, and if not, it will be discarded. So after enabling this, I was able to solve this issue. One tiny specification regarding this feature, is that it requires that the near value of the projection matrix, not be equal to 0.

I also implemented camera rotation using mouse drag, which was a simple enough task. For achieving this functionality, I found the distance moved by the cursor while it was being pressed. This distance was then divided by a Radius pre-set and the output gave me an angle θ which I then used to move the camera using the method mentioned above.

Picking Object and its faces

For picking the objects, I used the method of comparing the pixel value with the object's colour. Since each object had a distinct colour, I found the value of the colour under the pixel where the mouse is clicked and compared it with the stored values of the colours of the drawn objects. Upon selection the object turns black.

For the face picking, the only method that I knew was a similar approach as that of object picking. So I created different meshes for each of the faces of the objects, in complex cases, I broke the object into 4 parts and considered them as a separate face. Now for each of these sub-meshes I gave colours that varied from the other faces by a very small number, which made it look visually as the same colour but the pixel colour matching method worked for the same.

Initially most of the tasks were done using key combinations, but then I introduced a UI based mode changing feature, which makes it easier to change modes and perform the different actions.

Question 2: What were the primary changes in the use of WebGL in moving from 2D to 3D?

In our previous assignment, or the 2D case, we assumed our coordinate system to be from -1 to 1, which was then transformed using the `gl.viewport()` transform into a square canvas. There was no need of a view matrix or a projection matrix since it was a 2D image. But in the case of 3D, the point from where a user views the scene changes the output and the type of projection we use changes how a user looks at the scene. So, we needed to introduce the View Matrix which gave us the world view of the objects. We then needed a Projection matrix to give us the clip space and how we view the objects (Orthographic or Perspective). We also needed to enable the z-buffer, as I mentioned before, which would then test the depth of the objects and we will get outputs as expected.These were primarily the main changes that needed to be done.

Question 3: How were the translate, scale and rotate matrices arranged such that rotations and scaling are independent of the position or orientation of the object?
The concept of this remains the same as we had in the case of 2D. Firstly, the order in which these operations are performed is - [Scale, Rotate, Translate]. So regarding the point of scaling and rotating irrespective of the position of the objects- we perform those two operations at the origin itself, and then we translate it to another location. Now, regarding the orientation and position in terms of the Camera Position, it won't affect the model transformations since it is a separate matrix, that is multiplied after the model transform matrix has been applied.

Conclusion

At the end of the challenge I was able to get a clear understanding of concepts such as that of a z-buffer, whose task I saw in action. I was also able to implement and learn about the View and Projection matrix. I also was able to resolve of of my doubts from previous assignment, as to how does one make the canvas have the same aspect ratio as the browser window and not mess up the objects drawn. Finally this was also helpful in learning about models in Blender and dealing with the contents of the OBJ file and utilising them in our programs.