Assignment-2, 3D Rendering

Ellanti Bharath Sai
IMT2018022
bharath.sai@iiitb.org

***Abstract*—Rendering and Manipulation of 3D objects.**

I. Introduction

In this assignment we need to render 3D models using webgl and perform actions like translating, rotating, scaling on the objects.

II. Approach

A. Rendering and Coloring the models

To render an object we need vertices to be passed to vertex shader. As this is 3D rendering it is not easy to find the vertices of a model. So I first create a model n the blender and then import it as obj file. This obj file contains vertices, vertex normals, vertex faces etc. I then import this obj file into a mesh class to decode the values of file. For decoding of obj files I used webgl-obj-loader library.

This library takes the file as input and we want only the vertices of the object and with vertex indices for easy calculation. We then create a vertex buffer and pass the vertex data into it. Also we create a index buffer and pass the indices. Then we create a object of this mesh class in index file for rendering.

Before this we need to update the vertex shader which we used for 2D rendering. In 2D rendering vertex shader we need position variable and modeltransformation matrix for rendering, For 3D rendering along with position and model transformation matrix we also need view matrix and projection matrix. View matrix defines the position of camera and projection defines whether it is perspective projection or orthographic projection.

We find the values of this view and projection matrix in transform.js file and then send to the vertex shader from draw function of mesh class. These view and projection matrix are uniform variables.

After rendering this we get a black image of our object. To render color, We create a color vertex and we add our color code of size 3 example (Red - [1,0,0]) to this color vertex repeatedly for vertices.length/3 times. Then we create a color buffer and send it to the vertex shader from there it goes to fragment shader as varying variable. By this we can render an object with desired color by setting some camera angle.

B. Translate, Rotate,Scale,Camera

For translating we pass a translate vector which contains the position to which it translates and then update the matrix. For rotating we need to specify the axis of rotation also as this is 3D rotation. For rotaion we pass rotation axis and rotation angle. For scaling similar to 2D we pass the amount to be scaled in three directions. For changing the camera we need to change the eye vector in lookat() function.

C. Picking

To pick a 3D object in world space. I first find the coordinates of mouse while clicking on page. And use this values to find the pixel at that point, Now we got the pixel value at mouse click. Now we check if this pixel value matches with pixel value of any object in the space, If it matches with some object then we pick that object by changing the color of object to black.

Similarly for picking face of an object we follow above procedure but we import the object as faces rather than as whole object. Consider an example of cube. We import this cube as 6 squares and then form a cube. These six squares have very small difference in pixel value and hence appear to be in same color. While in face mode we use these different pixel values for different faces to find the face we click.

D. Camera Rotating

To change the camera we need to update the eye parameter in lookat() function. rotating camera by mouse drag I first calculate the mouse click position and then calculated the position to which it moved. We then calculated the slope of line formed by these two points. If tan inverse of this slope lies between 90 and -90 then mouse is dragged right wards else left wards. We now found the direction of mouse drag. We maintain a theta variable and update it by plus 1 if mouse moved right wards and minus 1 if mouse moved leftwards. We use this theta to calculate xyz values of eye vec of lookat() function. Based on the axis of rotation we keep the rotation axis parameter constant and update the remaining two.

III. Questions

A. Question - 1

Renderer and fragment code are same for both 2D and 3D. In vertex code of 2D we need to now add another two variables view matrix and projection matrix so that we can do 3D rendering also we need to update the gl position as view * projection * modelmatrix * aposition which is modelmatrix * aposition in 2D. In mesh class almost everything is same as 2D expect in 2D we can hard code the vertice but here we read vertices from another file. Also in 2D we only work with two coordinates x and y, we ignore Z coordinate, but we can't do this in 3D now Z coordinate also plays important mainly deciding the depth of the object. Transform.js has no impact on changing from 2D to 3D but in extra we calculate view matrix and projection matrix.

B. Question - 2

In 2D we totally ignore the Z value of every vertex but in 3D we also need to take care of Z value or depth. In 2D we never care about camera, But 3D rendering differs by changing the position of camera. Also by changing the projection matrix between orthographic and persepective views we get different rendering of same object. Whereas 2D has no such projection problems.

C. Question - 3

We do translation rotation and scaling with respect to origin . Hence scaling and rotation independent of orientation and position of object.

IV. CONCLUSION

By moving from 2D to 3D the code change is less but understanding the concepts is more and a little bit confusing as we now need perform operations by keeping in the mind the position of camera.