CS 606 - Computer Graphics / Assignment 2

1st Shashank Reddy
IMT2018069
International Institute of Information Technology, Bangalore
shashank.reddy@iiitb.org

***Abstract*—In this assignment we render and manipulate 3D objects. New concepts learnt in this assignment are - Creating 3D models (using a modeling tool), Importing 3D mesh models, Transformations of 3D objects and Picking model objects and their constituent parts in 3D.**

I. Problem Statement

For the exact key mappings, please refer to the README file.

A. Initial Rendering

Initially, we render a set of axis designed on blender. We also render the 3 meshes at the centre. After this, the user can select from a set of specified modes in any order.

B. Place the meshes at the vertices of a triangle (Mode d)

When we enter this mode, the meshes are rendered at the vertices of a triangle whose centroid is at the origin.

C. Place the meshes at the mid points of the sides of the triangle (Mode e)

When we enter this mode, the meshes are rendered at the mid points of the sides of the triangle described in the above mode.

D. Rotate the meshes about the x-axis, y-axis and the z-axis respectively. (Mode f)

When we enter this mode, every time we press f, the first mesh is rotated about the x-axis, the second about the y-axis and the third about the z-axis respectively.

E. Scale the meshes 0.5x, 2x, 3x respectively (Mode g)

When you enter this mode, you can scale the objects either 0.5x, 2x or 3x based on the mode you choose within this mode.

F. Picking objects and faces (Mode h)

In this mode, you can pick objects by entering object picking mode and clicking on them, or you can enter face picking mode and pick faces by clicking on them. To unpick objects/faces click anywhere else on the canvas.

G. Rotate the camera (Mode i)

In this mode, we can rotate the camera about the x-axis, y-axis and z-axis, by picking the respective mode and then rotating the camera by dragging the mouse horizontally.

II. Implementation of the modes-

For each of these meshes I assume the centroid to be at the origin. I translated the meshes on blender so that they are approximately centred at the origin.

A. Transformation-

I used transformation matrices for the required vertex transformations. I used the glMatrix library to generate these matrices. I made the transformations such that the history of operations is stored in the mesh's Model Transformation Matrix.

```
Rotate(){
    let rotationMatrix = mat4.create();
    mat4.fromRotation(rotationMatrix,this.rotationAngle,this.rotationAxis);
    mat4.multiply(this.modelTransformMatrix, rotationMatrix, this.modelTransformMatrix);
}
```

Fig. 1. Implementation of Rotation.

```
Translate(){
    let translationMatrix = mat4.create();
    mat4.fromTranslation(translationMatrix, this.translate);
    mat4.multiply(this.modelTransformMatrix, translationMatrix, this.modelTransformMatrix);
}
```

Fig. 2. Implementation of Translation.

B. Model View Matrix

I used the lookAt function in glMatrix to generate the model view matrix. I made this a part of the renderer as it is the same for every object.

C. Projection Matrix

I used the perspective function in glMatrix to generate the projection matrix for perspective viewing. I made this a part of the renderer as it is the same for every object.

D. Axis Rendering-

I only designed one axis (y-axis) on blender, using a cone and a cylinder. To generate the other 2, I rotated the y-axis by 90 degrees about the z-axis to get the x-axis and vice versa. I then colored the x, y and z axes Red, Green and Blue respectively

E. Updating the position of the Centroid

Since the entire history of transformations of the mesh is stored in it's Model Transformation Matrix, I multiply this matrix with the vector pointing to the origin (initially the centroid is at the origin) to get the position of the updated centroid.

```
getCentre(){
    const currentVertex = vec4.fromValues(0,0,0,1);
    const updatedVertex = vec4.create();
    vec4.transformMat4( updatedVertex, currentVertex , this.transform.getModelTransformationMatrix());

    this.centre = [updatedVertex[0],updatedVertex[1],updatedVertex[2]];
    return this.centre;
}
```

Fig. 3. Updating the centroid of the mesh.

F. Mode d-

The co-ordinates of the vertices of the triangle are hard coded. I translate each of the meshes such that their centroid is at their respective vertex.

G. Mode e-

Given the co-ordinates of the vertices of the triangle, it is easy to calculate the mid points for each of it's sides. I then translated each of these objects such that their centroids are at their respective mid points.

H. Mode f-

w.l.o.g, we will look at the first mesh. I rotate the mesh about the axis with respect to it's centroid. To do this, first, I translate the mesh such that it's centroid is at the centre, I then apply the rotation about the x-axis and translate the mesh back to it's original location.

I. Mode g-

Based on the scaling factor (0.5x, 2x, 3x) I scale the objects about their centroid. To do this, first I translate them such that their centre is at the origin. After this I apply the scaling about the origin and then translate them back to their original position.

J. Mode h-

The basic idea I used behind picking is that I color each component differently, and when the mouse is clicked on the canvas, I use the color of the pixel to identify this component.

1) Face Picking-: I divided each mesh into components, I did this by sequentially picking vertices to form a components [Ex: Component 1 = {v1,v2,v3,..,vN}]. I did this because of the notion that closely related vertices will exist in close proximity in the vertex buffer. I found that the ideal number of components was 3 for all the meshes and it captured the notion of sub-meshes well. For example, in case of the helicopter, the first component represented the chassis, the second the blades and the last the payload and in the car, the first component represented the chassis, the second the tyres, and the third the windows. I then matched the color of the pixel with the color of each component of the meshes, and if there was a match, I colored that component differently.

2) Object Picking-: To pick the objects, I check if the color of the pixel matches any of the colors of the sub components of a mesh, and if it is true, I color every sub component differently to indicate picking.

K. Mode i-

The eye parameter in the lookAt function in glMatrix decides the location of the camera. Thus, I store a vector pointing to this location in the renderer class. To rotate the camera about a certain axis, I rotate this vector about the axis w.r.t the origin and then reconstruct the Model View matrix. I also added a zoom in/out feature where I translate the eye vector towards and away from the origin.

III. QAs

A. To what extent were you able to reuse code from Assignment 1?

I was able to reuse most of the code from Assignment 1, mainly the class structure. Most of the code to render the animations remained the same, that is, the renderer class, I reused some of the code in the transform class from the previous assignment(used to transform meshes). In the previous assignment I defined a different class for each type of mesh, however in this case I just made one mesh class to handle all the meshes.

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

The main change is the introduction of Model View and Model Projection Matrices. The Model View matrix maps the co-ordinates from the world space to the camera space, and the projection matrix maps from the camera co-ordinates to the clip space. Thus, these 2 matrices are essential in rendering 3D models. Other than this, every thing has remained pretty much the same, the transformations (matrices) are identical in both the cases.

C. How were the translate, scale and rotate matrices arranged such that rotations and scaling are independent of the position or orientation of the object?

I initialized the model view matrix to be identity. Whenever I encountered a transformation I multiplied the transformation matrix to the left of the model transformation matrix so that the transformations occur in the correct order. Thus, the entire history of transformations is stored in the model transformation matrix.
Also in the case of scaling and rotation, these operations are done about the centroid of the mesh. (which is assumed to be at the origin initially).

IV. Acknowledgment

I would like to thank Prof Jaya and Prof Srikanth for coming up with such an interesting assignment and the TAs for their tutorials and quick responses to doubts we had when attempting this assignment.

REFERENCES

[1] https://glmatrix.net/
[2] https://webglfundamentals.org/
[3] https://developer.mozilla.org/en-US/docs/Web