**CS606 Computer Graphics**

**Programming Assignment 2**

**Rendering & Manipulation of 3-Dimentional Models**

***~Adrij Sharma  
IMT2019004***

**ANSWERS**

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

* Transform class whose object is instantiated in the constructor of the four model classes is re-usable. The main purpose of this class is to store the transformation matrix and make changes to this matrix when an appropriate function for translation, rotation and scaling is called.
* The primitives class in ass-1 is now changed to model class with some tweaks to the constructor where we have to parse the obj file and extract out the vertex data information for the blender model.
* Fragment shader has also remained the same.
* Several sections of code which have been repeated at several places have remained as they were.

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

* In the 2-Dimentional case, we assumed that our coordinate system is from -1 to 1. It was transformed using the *gl.viewport()* transform into a square canvas.
* View and projection matrices were not required in the case of 2-Dimentional images. In the case of 3-Dimensions, 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. There is a need to introduce the concept of View Matrix so that we get the world view of the objects.
* View matrix is able simulate the moving camera through the *lookAt()* function. The *lookAt()* function has 3 parameters – eye (position of camera), centre (the point where the camera aims), up (determines the direction of the up for the camera).
* Projection matrix is needed to give us the clip space and to assist in how we view the objects (Orthographic or Perspective).
* The object that we intend to draw must be inside the cube of [-1, +1]. Any part of the scene which is outside the unit cube will be clipped. Through a projection matrix, we can virtually enlarge or shrink the clipping volume. These matrices have to be passed to the vertex shader and perform matrix multiplication.
* The z-buffer had to be enabled. This was needed to test the depth of the objects and get expected outputs.

1. How were the translate, scale and rotate matrices arranged? Can your implementation allow rotations and scaling during the animation?

* The updateModelTransformMatrix() function in the Transform class handles the arrangement of translate, scale, and rotate. The important component that makes the animation accurate is the order of translate, scale, and rotate.
* If the model is not at origin when scaling is applied, the model will be scaled. It will, however, move the model's centroid towards or away from the origin.
* Because scaling works by multiplying vertex position by a factor, it will relocate the centroid while also scaling the model. However, we just wanted to scale the model down to its original size. If the model's centroid is at the origin and scaling is applied, the model will just be scaled and the centroid will remain at its current location.
* Because all models in our application start at the origin in the beginning scene, we can scale the model first and then translate it to the desired location. However, because rotation has the same problem as scaling, we must perform rotation before final object translation and then perform the final translation.
* So, the order of matrices is: Translate \* Rotate \* Scale.
* When multiplied with model vertices, the last matrix is applied first to the model. Because scaling and rotation are applied while the model is at the origin, they are unaffected by the object's position or orientation.

1. For computing the coefficients a, b, and c, the choise of t1 is 0.5. After choosing t1 we had to solve the three quadratic equations. Instead of a quadratic polynomial, we'll require n-1 degree polynomials for interpolation for n locations.

**CODE SPECIFICS**

* Camera and CameraTransform are 2 different classes so as to add modularity to code and correspondence with the Shape and Transform classes.
* Cube class was not of any use and hence has been removed from the code.
* The most important thing to mention here is that animation and ObjectAnimation are 2 different functions having different roles. The animation over path in mode 1 has been handelled in mode 1 where as the animation function handles the regular redering of scene.
* getModelObject() function does the task of making the objects in .obj available for use in the scene.
* 3 different files in the “axes” folder contains the 3 axes. 3 files in the objects folder correspond to each object respectively.

**REFERENCES**

1. <https://github.com/Amit-Tomar/T2-21-CS-606>
2. <https://github.com/invent-box/Learn-WebGL>
3. <https://www.youtube.com/watch?v=-nqIzV9OuQM&list=RDCMUCRQCzMftIUElX03kHjV4rmQ&start_radio=1>
4. <https://www.youtube.com/watch?v=-nqIzV9OuQM&t=810s>
5. <https://glmatrix.net/docs/index.html>
6. <https://www.youtube.com/watch?v=Y5eHAfuO9aY&t=491s>
7. <https://webglreport.com/>
8. <https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer>
9. <https://dens.website/tutorials/webgl/gl-matrix>
10. <https://glmatrix.net/docs/module-mat4.html#.perspective>
11. <https://glmatrix.net/docs/module-mat4.html#.lookAt>
12. <https://math.hws.edu/graphicsbook/c7/s1.html>
13. <https://webglfundamentals.org/webgl/frustum-diagram.html>
14. <https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Creating_3D_objects_using_WebGL>
15. <http://www.songho.ca/opengl/gl_transform.html>
16. <http://www.songho.ca/opengl/gl_projectionmatrix.html>
17. <https://www.scratchapixel.com/lessons/mathematics-physicsfor-computer-graphics/lookat-function>
18. <https://www.scratchapixel.com/lessons/3d-basic-rendering/3d-viewing-pinhole-camera>
19. <https://www.youtube.com/watch?v=MF1qEhBSfq4&list=PLa1F2ddGya_-UvuAqHAksYnB0qL9yWDO6&index=1>
20. <https://www.youtube.com/watch?v=nIoXOplUvAw&list=PLjEaoINr3zgFX8ZsChQVQsuDSjEqdWMAD>