Computer Graphics Assignment 2
(Rendering and manipulation of 3D models)

—Archit Sangal

Problems

- **Creating 3D models (using a modeling tool like the Blender)**
  I followed the video links given to us during the tutorial. I also read the documentation for the same. This covered topics like how to export the mesh object as '.obj' or '.ply' file. It was easy to do once we spend a little time, trying to do it. The procedure followed for making a mesh of one of the objects is included in the video.
- **Importing 3D mesh models**
  For this, we use the 'objLoader' package. This included the use of the inbuilt function fetch() which returns a Future or Promise type of object, not the required value. Hence, we need to make an async function and had to use await. This was a new concept. The object which we get after using the library was the data about vertices, indices, etc. which needs to be processed further for our final goals.
- **Transformations of 3D objects**
  I had already implemented the transformations for the objects for 3D in the previous assignment. It was more or less the same. But I got introduced to Euler angles for the rotation, involving more than one axis. But it was easy, too.
- **Computing animation paths and transforms**
  This was a bit mathematical. First, we convert clip coordinates of points - centroid of the selected shape, p1 and p2 to world coordinates. We will keep t1 = 0.5, and we calculate values of a, b, and c for x, y and z coordinates respectively (hence we get 3 values of a, 3 values of b, and 3 values of c). And we vary t =0 to 1 uniformly in every iteration and get a set of world coordinates. This will lead to interpolation of x, y and z.
- **View transformations**
  We define eye, up and centre (here we take this as origin). This can be used to find the 2 matrices - projection and view matrices. For finding them we use in-built functions like lookAt or perspective.
- **Picking model objects**
  We get the color of the pixel on which the mouse click happened. We get it by readPixels() function. Here as we do a composite action, screen gets cleared, so we set `preserveDrawingBuffer: true` and render the screen again before readPixels(). For matching the colors, as mathematical terms are involved, hence we cannot compare this with equality operator. We can treat the color (r,g,b) as a vector. We find distance

between the vectors and the color which is nearest to the color given by mouse click, shape corresponding to that color gets selected.

Answers to the Questions

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

Solution 1.

We were given the code template in the GitHub repo under example 5. We were able to use it to get example 6. Just a few lines were different. And we implemented those lines in assignment 1. Hence, assignment 1 was sufficient in this aspect. The file 'transform.js' was also nearly the same. To use it for 3D model instead of 2D models, some minute changes were made. But this part of the code was useable. Code related to VertexShader and FragmentShader were nearly the same. For all the key binding and mouse click, code structure was same, but the function body for the key binding were changed were doing were performed changed or got rearranged. Most of the code was reused.

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

Solution 2.

The code was nearly the same with very few changes. We added projection and view matrix for better visualization as in case of 3D objects. So we need to add these in VertexShader and update their values when a change in camera position is done(for view matrix). For this assignment, we also migrated from only vertex (duplication) to vertex-indices approach. In 2D, we could have used 2 float values to represent the vertex. But in 3D, we have used 3 float values to represent the vertex. So we need to increase the size of STRIDE or OFFSET. This was not required for our case as in previous assignment which was based on 2D models as we were keeping our z fixed (z=0).

Question 3. How were the translate, scale and rotate matrices arranged? Can your implementation allow rotations and scaling during the animation?

Solution 3.

We first get our object at the origin. In that case we can follow the steps in the order-

- Scaling
- Rotation around z-axis
- Rotation around y-axis
- Rotation around x-axis
- Translate along x, y, z axes

Yes, during animation we are changing translation values along x, y, z axes. We can change the rotation or scaling as it is not interfering with animation translation values.

Question 4: How did you choose a value for t1 in computing the coefficients of the quadratic curve? How would you extend this to interpolating through n points (n > 3) and still obtaining a smooth curve?

Solution 4.

As we assumed that p0, p1, p2 are not collinear and not too close to each other. So I took t1 as 0.5; If the t1 is varied, the quadratic equation will change. This will have no impact on interpolation, but it will affect the speed or the path of the animation as the coefficient will change.

For more than 3 points, we need to use (n-1) degree polynomial equation. Polynomial Equations are continuous and is a smooth curve. We can use n points given to find the n coefficient of n-1 degree polynomial.