COMPUTER GRAPHICS
Assignment-2 Report

Samaksh Dhingra (IMT2019075)

March 23, 2022

1 ANSWERS TO GIVEN QUESTIONS:-

Q1. To what extent were you able to reuse code from Assignment 1?

Ans. There were many similarities between Assignment 1 and Assignment 2, and hence a fair amount of implementations I was able to reuse in Assignment 2 from Assignment 1.

- Most part of shaders was same. Code for setting up the GL Shaders (compiling, linking, passing attributes and uniform to shaders, creating buffers for shaders) in Javascript was reused.

  I only had to create extra buffer to store indices and add 3 functions (drawElement, bindElementBuffer, readPixels) in shader, and reused rest of them.

- In Renderer also, most of the part was same - Setting up the canvas, animation function for infinite loop, rendering object (looping through primitives in scene, adding model transformation matrix to vertex shader, passing color of primitive to fragment shader, was same) , converting mouse to clip coordinates.

  I had to add parameter in getContext() Function to preserve buffer for readPixels and clear Depth Buffer manually in clear function. Also some modifications in render function was needed to draw object in different format and to pass the view and projection matrices also along with

model transformation matrix to shader. Rest all part was same and hence also reused.

- Fragment shader code was reused as it is.
- For vertex shader, just 2 more matrices, view and projection were to be added and multiplied to positions along with model transformation matrix to get 3d view. Rest all things were reused.
- Code for Scene class was also reused as it is.
- Transform class and all the basic transformation methods were reused as it is.

Q2. What were the primary changes in the use of WebGL in moving from 2D to 3D?

Ans. Moving from 2D to 3D primary changes in use of WebGL were:-

- z coordinates were no longer always 0. Hence 4×4 transformation matrices were passed instead of 3×3 transformation matrices as I did for 2D objects.
- Also added View Transformation Matrix and Projection Transformation matrix and multiplied them along with model transformation matrix to the initial position of object in vertex shader to find glPositions.
- Enabling Depth.test for better visualization.
- Since, in 3D the mesh data has potential to become quite large, so it is not advised to use traditional method of storing model as only triangle vertex information (redundant use of vertices). Instead if only unique vertices are stored and indices corresponding to it are mapped to identify faces in model, that would save lot of data with no loss of required information. Hence, for this, instead of array buffers and draw arrays, we used element buffers and draw elements to render.

Q3. How were the translate, scale and rotate matrices arranged? Can your implementation allow rotations and scaling during the animation?

Ans. For camera, since the rotation was required about the world axis at origin, translate, scale, rotation matrices were arranged as:-

model = scale × rotate × translate

For object, since rotation was required about the world axis at its centroid, translate, rotate, scale matrices were arranged as:-

model = scale × translate × rotate

Yes, I have implemented my code the way that it would allow rotations and scaling during the animation. As object is just translating during animation, there would not be any problem if we also, change rotation and scale matrices to update model transformation matrix.

However I have not done that as Assignment statement did not ask for it, but I tried for my own testing and it was working completely fine. You can remove the condition of not being in animation mode where I am rotating and scaling by keybindings to test it.

Q4. 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?

Ans. Given the points, p0,p1,p2, for this assignment I have calculated t1 by following method:-

$$t1 = \frac{d(p0, p1)}{d(p0, p1) + d(p1, p2)}$$

For extending this to interpolate through (n > 3) points, we can fit multidimensional bezial curves rather than just using quadratic curve for getting a smooth curve.

2 ADDITIONAL DESCRIPTION

2.1 Creating 3D models using Blender Modelling tool

Modelling operations used in the models I created using blender are:-

- **Aeroplane basic model:** Boolean-Intersection, Extrude, Screw, Boolean-Union
- **Satellite basic model:** Boolean-Difference, Extrude, Bevel, Screw, Solidify, Boolean-Union

Both objects were saved in '.obj' format with triangular meshes.

2.2 Importing 3d models in Javascript

'Fetch' API was used to read the text in obj file. 'WebObjLoader' library available online was used for parsing the object string and calculate vertices and indices arrays.

2.3 Passing these values to shaders and rendering the objects

Instead of 'gl.drawArrays()', 'gl.drawElements()' function was used to render the objects as their format consists of unique vertices and index mappings.

2.4 Picking The Object

The color value of pixel clicked by mouse is calculated and matched with the color of objects. If it matches the color of any objects we have, it selects the object. It is implemented on the assumption that all the objects are of different colour.

Initially, I was facing problem that readPixel() function to read pixels was giving (0,0,0,0). Later I figured out that when each time scene is rendered to another scene in animation function, there is a very minute time interval when the screen is empty. Due to that the function is not able to read the pixel values as it is evoked with mouse click, and there is no guarantee between synchronization of mouse click and rendering.

To solve this problem, when we get context of canvas we keep parameter of preserveBuffer as true, so that each time it renders new scene it keeps copy of previous scene using which we can read pixel.

2.5 Rotations

For Camera:

Since the camera is rotating about the world axes placed at origin of scene, when we update Euler Angles along X,Y, Z seperately, we could only get the rotations about it's local axis as it always goes XYZ Euler. This problem may give arise to Gimbal Locks. Although, we should be using quaternions for 3D Rotations to avoid these situations. But there is a trick we can still perform the rotation by minimizing the chances of Gimbal Lock and still rotate about world axes at origin rather than Local axis of object. Instead of keeping track of Euler Angles separately, I kept track of orientation by maintaining a Rotation Matrix in Transform class. When finding Model Transformation matrix each time we render, we left multiply this rotation matrix to get desired rotations.

For Objects:

Same thing has been done for objects also, just instead of left multiplying rotation matrix, we just right multiply rotation matrix with Model Transformation matrix. This is because, we want rotation about world axes situated at centroid of the object.

2.6 Animation

For animation, simply the curve is calculated based on values of p0, p1, p2. Mouse coordinates are calculated and converted to clip and further world coordinates to get p1, p2. We then find the updated centroid value of selected object to get p0.

Curve paramters are calculated by solving different quadratic equations we got by putting p0, p1, p2 points on curve.

t1 calculation has been explained in question asked above.

3 REFERENCES USED

a. Basic Template of fragment shader, vertex shader and different classes like Rendering, Shading, Shapes, Transformation are inspired from the TA's 'Example Code' uploaded on Github for reference of this course.

`https://github.com/Amit-Tomar/T2-21-CS-606`

b. Basic concepts of WebGL, concepts, formulas used for Linear Algaebra operations for transformations.

`https://webglfundamentals.org/`

c. Syntax and concepts of javascript.

`https://developer.mozilla.org/en-US/docs/Web/JavaScript`

d. gl-Matrix.

`https://cdn.skypack.dev/gl-matrix`

e. '.obj' model string parser

`https://cdn.skypack.dev/webgl-obj-loader`

f. References for more specific things implemented can be found in Tutorial Lecture 5 slide shared by TA.