Computer Graphics

Assignment 2

Simple 3D Model Animator

G Sri Harsha
IMT2019030

Answers

1)

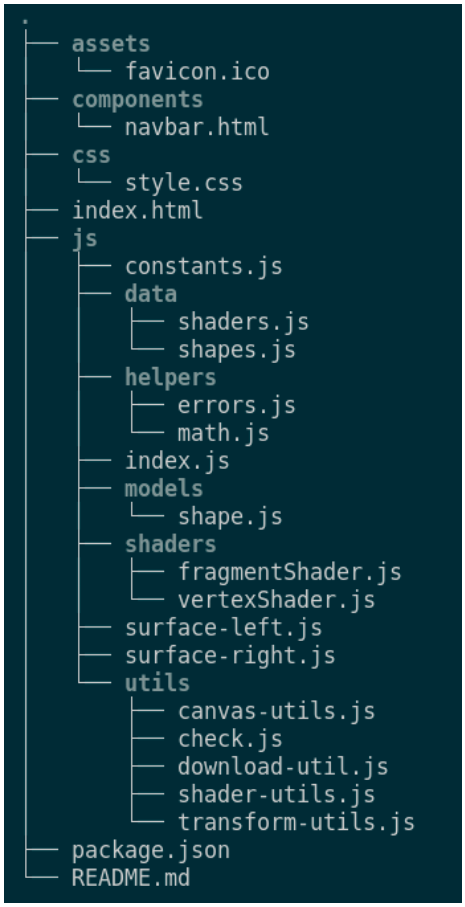
a) I have used almost all the code that I wrote in my previous assignment (Tangram Application). Though I had to refactor a few modules to support the 3d view and had to remove some unnecessary code such as generating random shapes, math.random functions etc.

b) The file structure of my project is almost the same unless this time I have made the code even more modular in nature. And have also added a few extra files to support the assignment specifications.

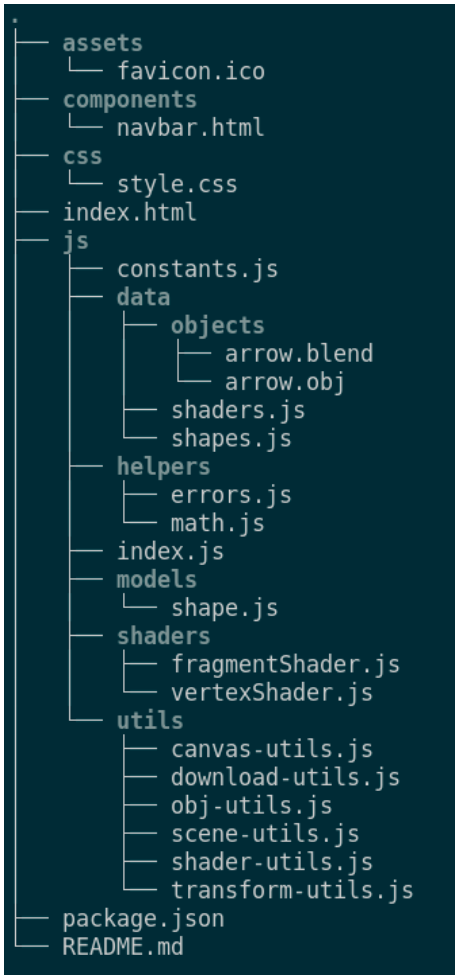
c) The shaders were also modified. Actually, they were even more generalized. Also, a few other functions such as reading/importing .obj file type code and handling these files were included this time.

d) All the matrices were also refactored made to support 3D model representations.

Previous Folder Structure:



Current Folder Structure:



2)

a) In 2D, we can only have one camera view. I.e planar view which we can directly see on our screen. But when it comes to 3D, there are multiple planar view possibilities (as we are trying to view 3D space on a 2D screen).
b) Hence we need a camera view that can help in selecting which planar view to present on the screen. Since everything in webGL is handled using matrices, this camera view data is also represented using two matrices.
c) This includes the view matrix and projection matrix.
d) A view matrix makes everything relative to the camera as though the camera was at the origin (0, 0, 0). I.e. using the view matrix, we basically select which 3D region we want to view as if our eye is present near the origin.
e) The projection matrix will change this 3D region and project all these details on a 2D planar region (w.r.t the screen). This is also referred to as ‘Perspective Projection’.
f) To be precise, perspective projection is a type of projection that graphically approximates the 3D objects on a planar surface (2D surface) so as to approximate the actual visual perception.

g) In terms of the shaders' code, the fragment shader didn't have any changes. Though the vertex shader had 2 new uniforms to support this view and projection matrix.
h) The data matrices (in JS) were also refactored (made nx3 from nx2) to support the z-axis attribute data.
i) Also since the data from the .obj files have indices instead of the direct data, I have introduced a 'drawElements' function based on 'drawArrays' function.
j) In terms of the pre-processing and actually rendering of the shapes, many other functions such as converting the 2D coordinate data into 3D coordinate data (using the view and projection matrices), etc were also implemented.

3)

a) I have used the following order: Translation matrix, Rotation matrix, and Scaling matrix (refer to 'updateModelTransformMatrix' function in '/js/utils/transform-utils.js'). Since in the shader code, we have '`proj_matrix * view_matrix * model_matrix * a_position`', we technically multiply the scaling matrix with the a_position first. Then followed by the rotation matrix and the translation matrix.
b) Hence the scaling of the vertices happens first. Then they are rotated w.r.t origin and then finally translated. Even during the animation, my implementation supports the rotation and also scaling.
c) In fact, all the operations can be applied at any stage in any order.

4)

a) In the current implementation, I took t1 as 0.5 as it seemed comparatively easy in terms of solving the calculations and getting the coefficients. Hence I have directly hardcoded.

```
let [ax, bx, cx] = abc(p0[0], p1[0], p2[0], 0.5);
let [ay, by, cy] = abc(p0[1], p1[1], p2[1], 0.5);
let [az, bz, cz] = abc(p0[2], p1[2], p2[2], 0.5);
```

b) Now to extend it to interpolating through n points (and still obtain a smooth curve), we can similarly hardcode the n t's as the $(1/n)^{th}$ of the total time.
c) In the first case, we will get a quadratic equation and can solve a, b, c (coefficients).
d) Though in the case of n points, we will be getting an n-1 degree polynomial and will be getting n coefficients.

Approach

- The first step was to generate the 3D models. And for this, I have used the blender software. I have generated the axes and two other 3D models. And then exported them as .obj files.
- Then from the code, I have imported these .obj files (data is in '/js/data/objects' and the functions to import them are in '/js/utils/obj-utils.js')
- Then I have set up the view and projection matrices and a few other importing variables to remember the state.
- Then I have used the code segments of the tangram application to apply the translation, rotation, and scaling to the modal matrix, also using the view and projection matrices.
- To find the nearest shape based on the click of the mouse, I have used the Euclidean distance with a small bias (0.002). Though there are also many other processes and functions applied to get the nearest shape.
- Then I have mapped a set of different keys to achieve different actions (key bindings).
- I have also used other event listeners such as 'keyup', 'mousedown', 'mousemove' etc.
- Then finally I have reused the download canvas function in the tangram application code, to actually implement a download option.

Key Bindings

- ArrowRight : Move right by 0.1px
- ArrowLeft : Move left by 0.1px
- ArrowUp: Move up by 0.1px
- ArrowDown : Move down by 0.1px
- m : Move along +z axis by 0.1px
- n : Move along -z axis by 0.1px
- 5 : Move camera-In
- 6 : Move camera-Out
- z : Rotate along x axis in clockwise direction (left key of 'x' on keyboard)
- c : Rotate along x axis in anti-clockwise direction (right key of 'x')
- t : Rotate along y axis in anti-clockwise direction (left key of 'y')
- u : Rotate along y axis in anti-clockwise direction (right key of 'y')
  etc.

```
if(event.key == 'ArrowRight') SHAPE.transform.moveX(0.1);
else if(event.key == 'ArrowLeft') SHAPE.transform.moveX(-0.1);
else if(event.key == 'ArrowUp') SHAPE.transform.moveY(0.1);
else if(event.key == 'ArrowDown') SHAPE.transform.moveY(-0.1);
else if(event.key == 'm') SHAPE.transform.moveZ(0.1);
else if(event.key == 'n') SHAPE.transform.moveZ(-0.1);
else if(event.key == 'z') SHAPE.transform.rotateX(0.1);
else if(event.key == "c") SHAPE.transform.rotateX(-0.1);
else if(event.key == 't') SHAPE.transform.rotateY(0.1);
else if(event.key == "u") SHAPE.transform.rotateY(-0.1);
else if(event.key == 'a') SHAPE.transform.rotateZ(0.1);
else if(event.key == "s") SHAPE.transform.rotateZ(-0.1);
else if(event.key == '+') SHAPE.transform.scalePos(0.1);
else if(event.key == "-") SHAPE.transform.scaleNeg(0.1);
else if(event.key == "x") CameraRotationAxis = 'x';
else if(event.key == "y") CameraRotationAxis = 'y';
else if(event.key == "z") CameraRotationAxis = 'z';
else if(event.key == "w"){
    if(MODE == 1) ANIMATION = 1-ANIMATION;
}
```

Project

- Google Drive Link: https://drive.google.com/drive/folders/1ReZejyS7JMjL--4ZBzvv-OU_tE6Zi72R?usp=sharing

References

- https://webglfundamentals.org/
- https://github.com/Amit-Tomar/T2-21-CS-606/tree/main/src/example6