INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY, BANGALORE

Assignment 2

Sushranth P Hebbar

February 21, 2021

1 PROBLEM STATEMENT

1.1 Rendering and manipulation of 3D models

Objectives:- Creating 3D models (using a modeling tool), Importing 3D mesh models, Transformations of 3D objects, View transformations, Picking model objects and their constituent parts in 3D

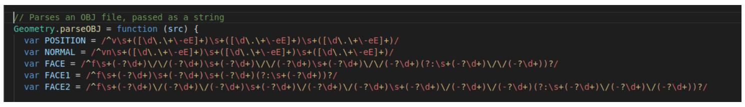
2 APPROACH

2.1 Drawing the Axes for the scene

An axis primitive was created using blender. It has two parts. The first is a cuboid and the second is a pyramid. Both of these parts were rendered and merged in blender. Then it was exported as an obj file. Check source/obj/axis1.obj to get a glimpse. Axis.js contains relevant data and methods related to the axis. Three instances of this class were created. They were appropriately transformed, colored and rendered at the origin.

2.2 Importing the 3D models

Geometry.js contains methods for parsing obj files. Parsing of the files was done using pattern matching with regular expressions. The file is read line by line, extracted based on the pattern, typecasted and finally returned as a dictionary of positions and indices. For more details refer, Loading obj model section in this link

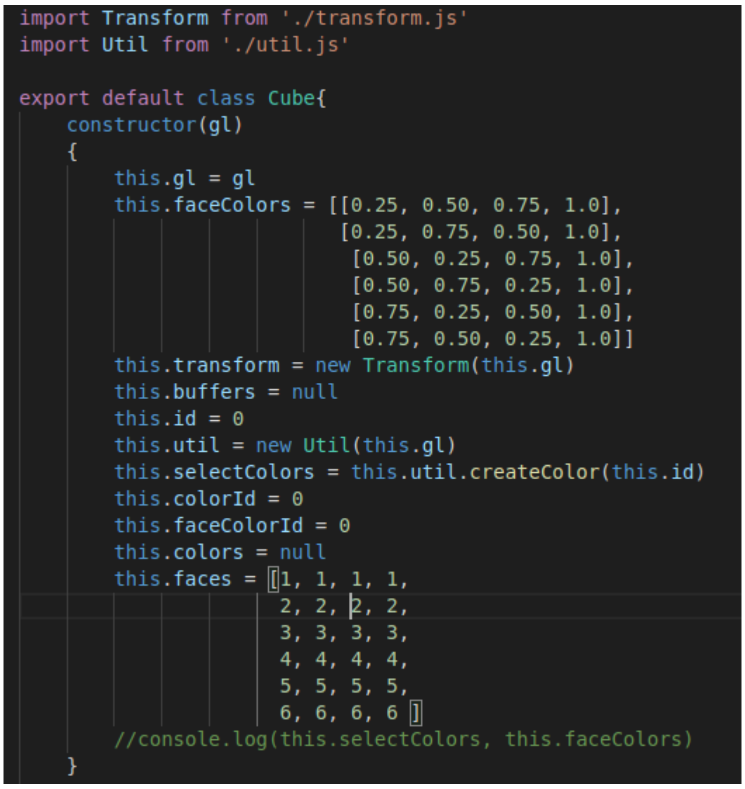


```
// Parses an OBJ file, passed as a string
Geometry.parseOBJ = function (src) {
  var POSITION = /^v\s+([\d\.\+\-eE]+)\s+([\d\.\+\-eE]+)\s+([\d\.\+\-eE]+)/
  var NORMAL = /^vn\s+([\d\.\+\-eE]+)\s+([\d\.\+\-eE]+)\s+([\d\.\+\-eE]+)/
  var FACE = /^f\s+(-?\d+)\/\/(-?\d+)\s+(-?\d+)\/\/(-?\d+)\s+(-?\d+)\/\/(-?\d+)(?:\s+(-?\d+)\/\/(-?\d+))?/
  var FACE1 = /^f\s+(-?\d+)\s+(-?\d+)\s+(-?\d+)(?:\s+(-?\d+))?/
  var FACE2 = /^f\s+(-?\d+)\/(-?\d+)\/(-?\d+)\s+(-?\d+)\/(-?\d+)\/(-?\d+)\s+(-?\d+)\/(-?\d+)\/(-?\d+)(?:\s+(-?\d+)\/(-?\d+)\/(-?\d+))?/
```

Figure 2.1: Pattern matching

2.3 Rendering models at Origin

There are 3 models at the origin. They are a cube, a pyramid and a diamond. There obj files were obtained from here. Each of these objects are rendered as instances of classes cube.js, pyramid.js and diamond.js respectively. Each of these classes contain important attributes like facecolors, a unique id, a dictionary of data buffers and an array of face values which is used to determine the face id.



```
import Transform from './transform.js'
import Util from './util.js'

export default class Cube{
    constructor(gl)
    {
        this.gl = gl
        this.faceColors = [[0.25, 0.50, 0.75, 1.0],
                           [0.25, 0.75, 0.50, 1.0],
                            [0.50, 0.25, 0.75, 1.0],
                            [0.50, 0.75, 0.25, 1.0],
                            [0.75, 0.25, 0.50, 1.0],
                            [0.75, 0.50, 0.25, 1.0]]
        this.transform = new Transform(this.gl)
        this.buffers = null
        this.id = 0
        this.util = new Util(this.gl)
        this.selectColors = this.util.createColor(this.id)
        this.colorId = 0
        this.faceColorId = 0
        this.colors = null
        this.faces = [1, 1, 1, 1,
                      2, 2, 2, 2,
                      3, 3, 3, 3,
                      4, 4, 4, 4,
                      5, 5, 5, 5,
                      6, 6, 6, 6 ]
        //console.log(this.selectColors, this.faceColors)
    }
```

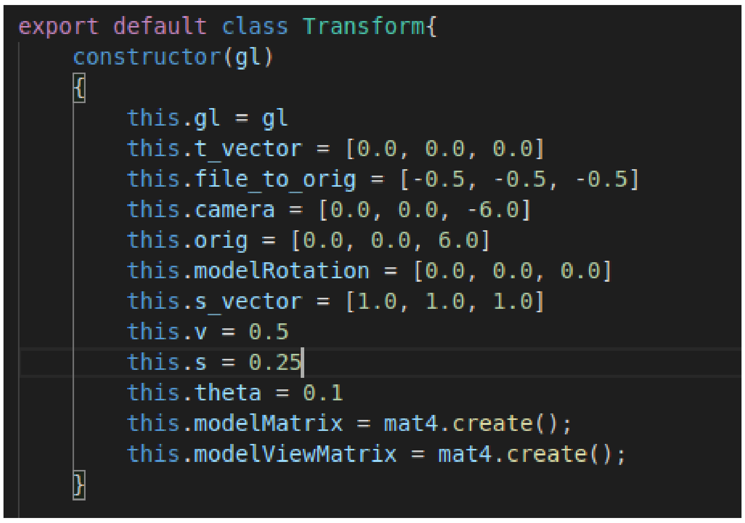
Figure 2.2: Primitive attributes

2.4 Instance Transformations

Every primitive class is composed of an instance of a transformation class. This class has been defined in transform.js . It contains attributes like a translation vector, rotation vector, scaling vector, a Model matrix and a Model-View matrix. During a keyboard event, the selected primitive's class attributes are modified. For example, if the arrow keys are pressed, then the appropriate component of the translation vector is modified. Similarly, based on the pressed keys, the scaling and rotation vectors are modified. These vectors are then passed as inputs for matrix operations which compute the final position and orientation of the instance.

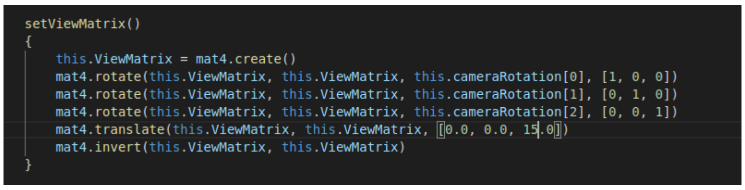
2.5 Camera Mode

This feature is implemented in camera.js. It contains the view matrix, perspective matrix and a camera rotation vector as its attributes. Depending on the mouse click events, the camera rotation vector is appropriately modified. It is then used as an input to modify the view matrix. The perspective matrix is computed just once inside the constructor as its inputs which are znear, zfar, field of view and aspect ratio are constant.



```
export default class Transform{
    constructor(gl)
    {
        this.gl = gl
        this.t_vector = [0.0, 0.0, 0.0]
        this.file_to_orig = [-0.5, -0.5, -0.5]
        this.camera = [0.0, 0.0, -6.0]
        this.orig = [0.0, 0.0, 6.0]
        this.modelRotation = [0.0, 0.0, 0.0]
        this.s_vector = [1.0, 1.0, 1.0]
        this.v = 0.5
        this.s = 0.25
        this.theta = 0.1
        this.modelMatrix = mat4.create();
        this.modelViewMatrix = mat4.create();
    }
```

Figure 2.3: Transform attributes



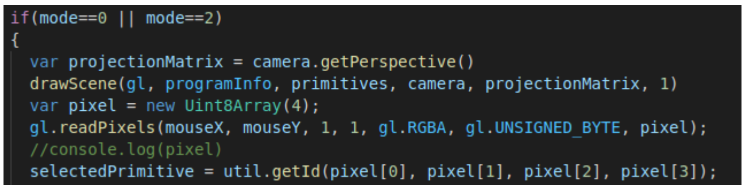
```
setViewMatrix()
{
    this.ViewMatrix = mat4.create()
    mat4.rotate(this.ViewMatrix, this.ViewMatrix, this.cameraRotation[0], [1, 0, 0])
    mat4.rotate(this.ViewMatrix, this.ViewMatrix, this.cameraRotation[1], [0, 1, 0])
    mat4.rotate(this.ViewMatrix, this.ViewMatrix, this.cameraRotation[2], [0, 0, 1])
    mat4.translate(this.ViewMatrix, this.ViewMatrix, [0.0, 0.0, 15.0])
    mat4.invert(this.ViewMatrix, this.ViewMatrix)
}
```

Figure 2.4: View Matrix Computation

2.6 Picking

2.6.1 Object Mode

Each primitive has a unique id. This id is converted to RGBA values and then scaled to fit the range [0.0, 1.0]. In object mode, the primitives are first rendered using the id as a color. Then the pixel value associated with the mouse click is obtained. Now, these color components are converted into an integer. If this integer matches the id of any primitive, then that primitive has been picked. The primitives are finally redrawn using their original colors. Converting the primitive id to RGBA and vice-versa are implemented using utils.js. In case of a color change, all the colors inside the color buffer of the appropriate primitive are replaced in a cyclic fashion with each mouse click.

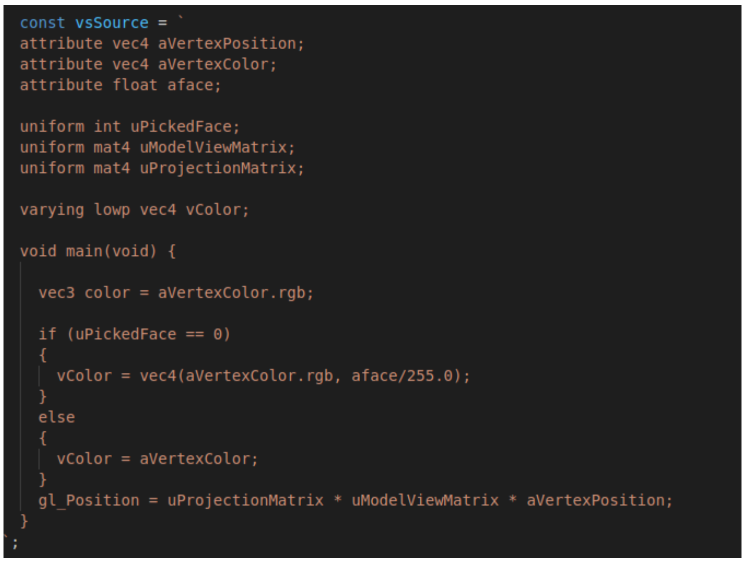


```
if(mode==0 || mode==2)
{
  var projectionMatrix = camera.getPerspective()
  drawScene(gl, programInfo, primitives, camera, projectionMatrix, 1)
  var pixel = new Uint8Array(4);
  gl.readPixels(mouseX, mouseY, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, pixel);
  //console.log(pixel)
  selectedPrimitive = util.getId(pixel[0], pixel[1], pixel[2], pixel[3]);
```

Figure 2.5: Object Picking

2.6.2 Face Mode

Every vertex that belongs to the same face must have the same face id. Additionally, there exists a uniform qualifier called picked face. When this is set to 0, the alpha component of the vertex will be set to that face id after scaling it. So, if any point is picked inside the face, it's alpha component will return the face id. This is because all points inside the same face will have the same alpha value as their endpoint vertices due to interpolation.



```
const vsSource = `
attribute vec4 aVertexPosition;
attribute vec4 aVertexColor;
attribute float aface;

uniform int uPickedFace;
uniform mat4 uModelViewMatrix;
uniform mat4 uProjectionMatrix;

varying lowp vec4 vColor;

void main(void) {

  vec3 color = aVertexColor.rgb;

  if (uPickedFace == 0)
  {
    vColor = vec4(aVertexColor.rgb, aface/255.0);
  }
  else
  {
    vColor = aVertexColor;
  }
  gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
}
`;
```

Figure 2.6: Vertex Shader

In face mode, the id of the selected primitive is obtained using the steps mentioned in Object mode. Then the qualifier upickedface is set to 0. The alpha value associated with the mouse click is obtained. This serves as the face id of the selected primitive. Then the color of just that face is updated in a cyclic fashion in a manner similar to what was described in object mode.

3 QUESTIONS

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

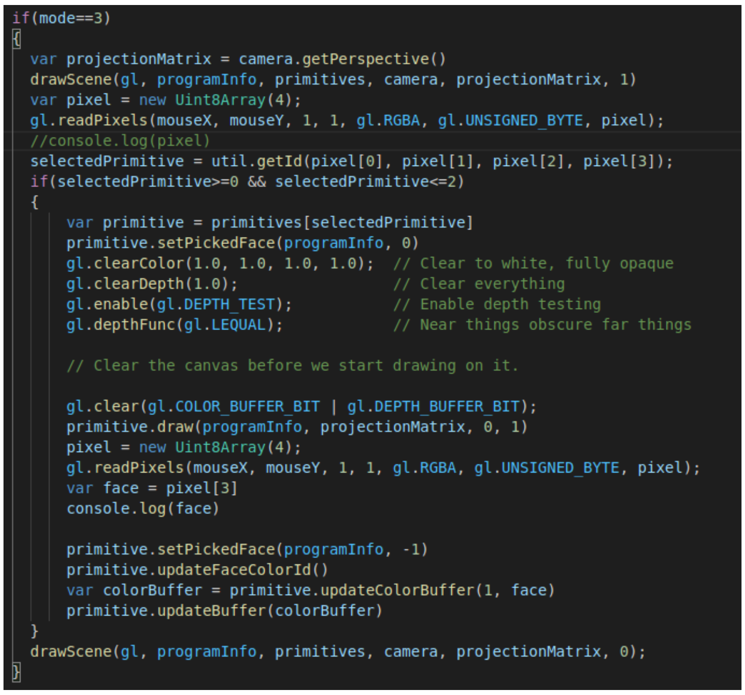
There was a lot of scope for reuse. The different primitives were implemented using classes just like in the previous assignment were reused. The vertex and fragment shader code were reused. The shader class which was used for compiling and linking the program was reused. And lastly code for creating and enabling the position vertex buffers were reused.

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

The changes were the creation of an index buffer. When there is a transition from 2D to 3D, the number of vertices increase. So, the number of faces increase which means the number of triangles increases. This can be organized easily using an index values which are loaded into an index buffer. A color buffer was added to store the colors for each vertex as opposed to hard coding the color value inside the fragment shader. Lastly, the depth test flag was enabled and the depth buffer was cleared during rendering.

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

Each primitive's center was first translated to the origin. The scaling and rotation were then performed. This will ensure that these operations are independent of the position and orientation of the primitive. It is then finally translated based on the translation vector attribute defined in it's constructor method.



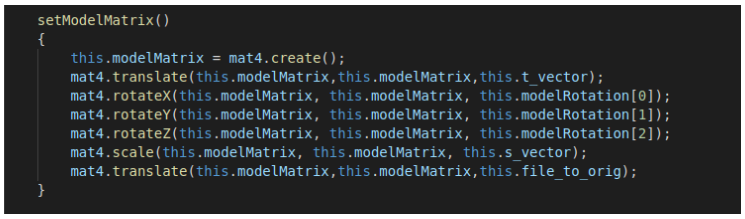
```
if(mode==3)
{
  var projectionMatrix = camera.getPerspective()
  drawScene(gl, programInfo, primitives, camera, projectionMatrix, 1)
  var pixel = new Uint8Array(4);
  gl.readPixels(mouseX, mouseY, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, pixel);
  //console.log(pixel)
  selectedPrimitive = util.getId(pixel[0], pixel[1], pixel[2], pixel[3]);
  if(selectedPrimitive>=0 && selectedPrimitive<=2)
  {
      var primitive = primitives[selectedPrimitive]
      primitive.setPickedFace(programInfo, 0)
      gl.clearColor(1.0, 1.0, 1.0, 1.0);  // Clear to white, fully opaque
      gl.clearDepth(1.0);                 // Clear everything
      gl.enable(gl.DEPTH_TEST);           // Enable depth testing
      gl.depthFunc(gl.LEQUAL);            // Near things obscure far things

      // Clear the canvas before we start drawing on it.

      gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
      primitive.draw(programInfo, projectionMatrix, 0, 1)
      pixel = new Uint8Array(4);
      gl.readPixels(mouseX, mouseY, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, pixel);
      var face = pixel[3]
      console.log(face)

      primitive.setPickedFace(programInfo, -1)
      primitive.updateFaceColorId()
      var colorBuffer = primitive.updateColorBuffer(1, face)
      primitive.updateBuffer(colorBuffer)
  }
  drawScene(gl, programInfo, primitives, camera, projectionMatrix, 0);
}
```

Figure 2.7: Face Picking



```
setModelMatrix()
{
    this.modelMatrix = mat4.create();
    mat4.translate(this.modelMatrix,this.modelMatrix,this.t_vector);
    mat4.rotateX(this.modelMatrix, this.modelMatrix, this.modelRotation[0]);
    mat4.rotateY(this.modelMatrix, this.modelMatrix, this.modelRotation[1]);
    mat4.rotateZ(this.modelMatrix, this.modelMatrix, this.modelRotation[2]);
    mat4.scale(this.modelMatrix, this.modelMatrix, this.s_vector);
    mat4.translate(this.modelMatrix,this.modelMatrix,this.file_to_orig);
}
```

Figure 3.1: Transformation Matrix.

4 MISCELLANEOUS

4.1 Keyboard and Mouse Events

The M key is used to cycle across different modes. There are 4 modes. Mode 0 is instance transformation mode. The arrow keys are used to move the selected object in the x-y plane. The I and O keys move the primitive along the z axis. The numpad + and numpad - keys scale the selected primitive. The numpad 8 and numpad 2 rotate the primitive by a positive and negative angle respectively. The A key is used to cycle across the 3 different axes. Mode 1 is Camera transformation mode. Click the left mouse button and drag left to right or the reverse to move the camera along the selected axis. Mode 2 is object mode. Clicking on the primitive changes the color of the whole primitive in a cyclic manner. Mode 3 is face mode. Clicking on the face of the primitive changes the color of the face in a cyclic fashion. Clicking anywhere else unselects the clicked face or object.

4.2 Code

The source folder contains the code and obj sub folder. The code folder contains all the JS files. The classes related to the primitives can be found inside cube.js, pyramid.js, diamond.js and axis.js. Camera.js contains the methods associated with the camera mode functionality.

Transform.js contains methods related to instance transformation. Util.js contains methods for converting RGBA values to an id and vice-versa. Geometry.js contains methods for parsing obj files.

5 References

Link to Video
https://webglfundamentals.org/webgl/lessons/webgl-3d-camera.html
https://webglfundamentals.org/webgl/lessons/webgl-picking.html
https://www.youtube.com/watch?v=rNyKWppAO38