Interactive Graphics 1st homework

Name : Afrah Salwa Ali Hersi
Matricola number : 1837991

1- the viewer position and projection(my choice) , viewer position and volume controlled by buttons :

For this task I used the lookAt function to add the camera or the eye position (the viewer)
The lookAt function takes parameters :
1-eye which is vector of three element (X,Y,Z)each specified as an operation to rotate the object to get the desired view with the help of the Math library in javaScript
2- at: which is the desired point to look at
3- up: which is the direction for the camera or the eye
As for projection I decided to use orthographic projection which also controls the viewing volume it need six parameters which are: bottom, ytop, left, right, near and far , I sit the first four parameters to be between -1 , 1 as for how gar or near its from the plane I tried several numbers which made the cube half shown or not appearing at all ,eventually I sit it to 0.3 and 3.0 (respectively).
After specifying the viewer position I was able to compute the ModelView by making it equal to lookAt function.
Finally both matrices are send to vertex shader using
Gl.uniformMatrix4fv(modelView, false, flatten (modelViewMatrix))
Gl.uniformMatrix4fv(projection, false, flatten (projectionMatrix)) , all written in the render function .
As for both already defined in the .html file as uniform matrix of 4 and in the init function to get their location
The viewer position controlled through buttons (increase theta , decrease theta , increase phi , decrease phi) by setting the radius to 2.0 , initializing theta and phi to 0.0 and using the “dr” equation (from the examples provided by the book)
When the clicked the cube rotate to position of the desired view
As for viewing volume I controlled the viewing volume by clipping using Interactive orthographic viewing of cube , same as the orthographic projection, I sit near to 0.3 and far 3.0 choosing the values for near and far was difficult I tried multiple values which made the cube in weird shape the viewing volume is controlled by buttons Z and R when z increases the near and far both multiplied by 1.1 while decreasing Z are multiplied by 0.9 which result in cutting the volume in contrast with increasing which increase the volume size , As for R it controls the radius.
Both values and parameters are from the book examples .

2- scaling and translation matrix :
Scaling is easier after defining the modelViewMatrix, only By multiplying the modelViewMatrix diagonal elements with scale factor define and sit to 0.5 “scale” and I was able to execute uniform scaling in all direction cube bigger or smaller with the same value controlled by slider with min value at 2 and max value at 3 . As for translation I added three sliders to control the translation around each axis , using the function translate define in MV.Js file which multiply the modelViewMatrix with translate function which takes three parameters I defined to have the same value translate_x , translate_y and translate_z which is 0.0 .

3- orthographic projection with plane near and far controlled by slider:
orthographic maintains parallel lines but provide no sense of depth In real life objects that are further from the camera appear smaller.
I already used the orthographic projection in the first question here I only added slider to control far or near instead of buttons and Z,R parameters , min to near is 0.001 otherwise the image can't be seen
These near and far values can be changed using a slider. The problem again is decideing the values for near and far ,at the end I used the values specified by the book examples and other online resources The advantage of using this method is that it is simple and clear how to use it and requires only one line of code. The disadvantage is that it can be very difficult to find the correct values for the parameters

4- splitting the window vertically to two parts , one shows the orthographic and the second shows perspective projection both controlled by the same slider

At the beginning I was confused on how to split the window eventually I learned about

```
  gl.enable(gl.SCISSOR_TEST);
gl.scissor
  gl.viewport
```

Both in webgl which allow us to display the same object or multiple one the same window by limiting the drawing to a specific rectangle defined by us and I implemented the same orthographic projection as mentioned above as for perspective projection parameters which are fovy, aspect, near and far. Fovy is the Field of View which is the vertical angle of the camera lens and aspect is the aspect ratio of the width by the height of the canvas, and near and far are the same as chosen for orthographic ,I used the same values pacifies in the slides and the book for those parameters , Bothe method still have the disadvantage of deciding which values to pick but still the pros which is all Donne in couple line of codes.

5- introduces light source , replace the color by properties of the the material , assign to each vertex normal : 6- implement with phon and Gouraud method with a button to switch between them:I decided to combine those two steps since they are related to each other

At first I introduced light source and properties of a material same as the one suggested in the book examples As for normalizing I used the known method which is subtracted the vertices that join at the edges and take the cross product of these.

```
var t1 = subtract(vertices[b], vertices[a]);
  var t2 = subtract(vertices[c], vertices[b]);
  var normal = cross(t1, t2);
  var normal = vec3(normal);
```

And I created a buffer for normals array and specified location to be able to send it to the vertex and fragment shader I also added a button to switch between the phong method or Gouraud method by using if,else statement and compute the desired method ,initiated with true which is the phon method , to implement the two method we have to write few changes with both vertex shader and fragment shader in .html file . As for phong method the parameters in vertex shader are , the light source L as the distance between this and the object, the viewer E, the half-way vector as the sum of these two, Kd and Ks, coefficient of diffusing and specular, respectively and the final color is the combination of ambient, specular, and diffuse and is sent to the fragment shader . Both of the changes and code I wrote with the help of the book examples .as for Gouraud method I used same code from the book to calculate the average of normals in the vertex shader and then sends it to the fragment shader to apply the Phong model to each vertex .

7-add procedural texture on each face, with the pixel color a combination of the color computed using the lighting model and the texture ?

For the texture I used the same chessboard mentioned in the book's examples I had a lot of time figuring out how to compute the combination since I kept getting error with varying names being similar with different type eventually I was able to introduced new parameter to store the combination in the fragment shader in addition to that, the last line which computes the combination

```
 vec4 fColorname;
 gl_FragColor = fColorname*texture2D( texture,
fTexCoord );
```