1. **Modeling a chair, using 40 vertices**
   In order to model the chair , I used 40 vertices starting from 0 to 39-th vertex. The main idea was to use the concept of a cube to compose a chair made of a backrest,seat and four legs.
   To realize this, I used different cubes one for each pieces of the chair, but at the beginning there was a problem due to the fact that each cube has six vertices and using eight cubes lead to 48 vertices. Even if we simply re-used the vertex that were matching we could only decrease to 42 vertices. So the idea was to build some of the cubes in a way such that we use some vertex many times.
   In example the vertices that were matching between the seat and the backrest were used many times.
   The main saving was to declare the front-face of the behind legs, with a shape which is not perfectly a square , using the function quad () and declaring inside the vertex used also for the backseat and the seat.
   As last thing , i declared the vertices in homogenous coordinates. For the normals of the faces of the chair I simply followed the right-hand rule so the normals of every face was pointing in the outside direction.

2. **Rotation about a point different from the origin**
   In this point we were asked to perform a rotation about a point that was different from the origin of the chair. My point of rotation was the top right hand corner of the chair , and i started the rotation about the y-axis. Indeed i used a var axis= 1 to start the rotation around y, and var theta [0,0,0] to denote by which axis I will rotate my chair. I used 3 buttons in the html file to take into account the choice of the user and then the corresponding event listener. To perform the rotation i used 3 matrices: a rotation matrix called rot, a translation matrix tr and the inverse of this one called tr1. I did this because this is the way to perform this type of rotation , so first of all I initialized my rot matrix by multiplying it to the vector of rotation by x,y and z.
   Then I initialized the 2 translations matrices using the function translate in the position of the vertex above and the inverse in the position of the vertex with negative coordinates. Then i used a matrix called ctm (current trasformation matrix) and inside of this i first put the translation matrix then multiplied the result by rotation matrix and then the result again with inverse of the translation matrix. After doing that , i simply sent ctm to the shader and then multiply it with each vertex, of course all of this was done in render function; another solution could be to perform the translation and rotation inside the vertex shader that could be better because in that way we are not sending

the 16 values of the matrix but only the point of translation and the angles of rotation. But i preferred the first one in order to maintain a simplier vertex shader.

3. **Adding a viewer position and a perspective projection**
Here we have to add the position of the viewer and use a perspective projection in order to realize the view volume. Regarding the viewer position i declared 3 variables eye,at,up , the first computed based on a dependence of 2 angles theta1 and phi and a radius which represent how much my camera is far away from the chair. Then I used the lookAt function of webgl, to set the modelViewMatrix in the render. For the projection, we were forced to use a perspective projection , so my projectionMatrix was set using the function perspective with four parameters fovy,aspect ,near,far.
The last two ones are the most important, specially near because we are defining the dimension of our view volume and depending on "near" the chair can be outside of the view volume. After this i simply sent the 2 matrices above to the shader to perform the multiplication : projectionMatrix * modelViewMatrix * ctm * aPosition.
To control the viewer position i declared three sliders called theta ,phi and radius and to modify the view volume i declared four sliders to control near, far, aspect and fov.

4. **Adding a spotlight with attenuation**
To perform this task, I started from the idea of a simply point light source and then i added some parameters which are typical of a spotlight. Starting from the basic code of a point light source, i added a direction a cutOff angle and an attenuation factor. The second thing is that, since now we are dealing with a light source we have to send to the shader the normals of each vertex. So inside the function quad(a,b,c,d), which is called many times by the function colorCube() to build the faces of the chair and receives the vertices for that face, I always subtract b and a and then between c and b and then using the cross function to perform the normals. Then when we are pushing the vertices we also push the normals inside the normalsArray used by the buffer. The light direction was a simple vec3, then the cutoffangle represent the opening angle of the light and the attenuation is a factor that represent how much the spotlight is intense on the boundary. To realize the spotlight i sent the parameters of the light like the light position, ambient product , diffuse product and specular product and then the light direction , cutoffangle and attenuation factor to the shader.

I noticed that to obtain a more realistic result of the spotlight, i used the Phong model and then apply the Phong model modified in the fragment shader. In order to rotate the light with the chair, I computed the L vector in the vertex shader as " L = lightPosition - aPosition" , but I could also obtain the rotation of the chair without the light that is stationary. This could be done by computing L = (modelViewMatrix*lightPosition - modelViewMatrix * ctm * aPosition). Then in the fragment shader, I normalized the vector L,E,N and i checked if the cosine of the angle between the lightDirection normalized and L is greater than the cutoffAngle. If this is true than the fragment is lit, and then i have to compute the color based also on the attenuation factor. I simply declared a float variable intensity computed as the power between the cosine above and the attenuation factor. So the final color will be the sum of ambient, diffuse, specular times the intensity. To turn on/off the light i did a check on lightflag, controlled by a button. To control the spotlight i added six slider for the position and direction of the light along the three axis, and a slider called spotlimit to increase and decrease the angle. For this one I build a function inside the javascript which computes the angle in rad which is used by the shader.

5. **Assigning a material to the object**
   In order to add the material of my chair i used materialAmbient, materialDiffuse, materialSpecular and materialShininess. Of course the material of the chair should interact with the light so i did a moltiplication between the parameters of the light and the material parameters of the chair. I sent them to the shader.

6. **Per-vertex and per-fragment model and combine them with a slider**
   As requested I wrote the model both in vertex and in fragment shader. In particular in per-vertex I computed the vector N,L,E, checked if the dot product is greater than the cutoff angle and computed a color and this color is sent to the fragment shader. The fragment shader will receive the color computed in the vertex shader and also N,L,E not normalized. Then we do the same check in the fragment. Then i used a mix function to mix the color from the vertex shader with the one from fragment, based on the value computed by the slider.

7. **Quantization mechanism**
   To implement the quantization mechanism, in the fragment shader i used a function called distance which computes the difference between two vectors, the color computed at the and of the mixing of the shading models, with the first

choose one from the user. Using a for cycle i computed the difference between the color computed by the mixing and the following one, if this difference is less then the previous one then this is the nearest color that should be given as output from the fragment shader.