Interactive Cube

1. **Add a button that changes the direction of the current rotation.**

In the .html:
In order to switch the rotation, first of all it is important to add a button in the HTML.
```
<button id = "ButtonS">Switch Rotation</button>
```

In the .js:
Then, in the JavaScript, I have to declare an auxiliary `var switch_direction = 1`**.**

In the init, to change the direction from 1 to -1 I used
```
document.getElementById("ButtonS").onclick = function(){switch_direction
= - switch_direction;};
```

Finally, in render I can switch the direction.
```
if(flag) theta[axis] += 2.0 * switch_direction;
```

2. **Move the transformations matrices from the shader to the JavaScript application, so that the ModelView and Projection matrix are computed in the application and then transferred to the shader.**

In the .html:
From the HTML main I removed matrices rx, ry, rz, I added variables and modelViewMatrix, projectionMatrix and multiplies them in gl_position.
```
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
```
gl_Position = modelViewMatrix * vPosition;

In the .js:
I have to compute model-view matrix in JavaScript render instead of HTML main by declaring a ctm matrix. Projection matrix will be declared later in the task 4.
```
var = modelViewMatrix;
var = modelViewMatrixLoc;

ctm = mult(ctm,rotateZ(-theta[zAxis]));
ctm = mult(ctm,rotateY(-theta[yAxis]));
ctm = mult(ctm,rotateX(-theta[xAxis]));

gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(ctm));
```

3. **Include a scaling (uniform, all parameters have the same value) and a translation Matrix and control them with sliders.**

In the .html:

I have to add scaling and translation matrices, with a slider to control the scaling and three sliders to control translation axes. This four sliders will be controlled with a JavaScript function.

In the .js:
I have to add two vectors, `scaleVector = [1.0,1.0,1.0]` **and** `translateVector = [0.0, 0.0, 0.0]`**, for scaling and translation, controlled by the sliders previously added in the HTML.**
Finally, everything will be multiplied with CTM in the render, first with translation, then with scaling, finally with rotation.

4. Define an orthographic projection with the planes near and far controlled by sliders.

In the .html:
Finally, I can initialize projection matrix in vertex-shader and multiply it with modelViewMatrix and vPosition in gl_position.
In order to control near and far planes with sliders, we can add them in the HTML and control them in the JavaScript init function.

In the .js:
I can initialize projection matrix in the render with ortho and lookAt functions: in order to do this I have to initialize rispectively near, far, bottom, ytop, left and right variables for ortho function, and I have to initialize eye, at, up variables for lookAt function.
In order to control near and far planes with sliders, in the JavaScript init function I can modify values of near and far variables.

At last,
```
gl.uniformMatrix4fv(projectionMatrixLoc, false,
flatten(projectionMatrix));
```

5. Define a perspective projection, introduce a button that switches between orthographic and perspective projection. The slider for near and far should work for both projections.

In the .html:
It is sufficient to declare a button to change projection.

In the .js:
With change projection button declared in the html it is possible to change a boolean variable that control ortho projection.

To use perspective projection it is used perspective function with fovy and aspect variables, inserted in an if-else structure in order to switch from the two projection types (ortho and perspective).

6. Introduce a light source, replace the colors by the properties of the material (your choice) and assign to each vertex a normal.

In the .html:
We have to change vColor with vNormal and declare the variables ambient, diffuse and specular, needed for lightning. Then, computing calculations inside main function.

In the .js:
Subsistute colors array with normals array and replace vertex colors with new variables for lightning. Old variables are replaced with new ones in quad function. I change cBuffer with nBuffer and vColor with vNormal.

Now:
```
ambientProduct = mult(lightAmbient, materialAmbient);
diffuseProduct = mult(lightDiffuse, materialDiffuse);
specularProduct = mult(lightSpecular, materialSpecular);
```

Finally, it is possible to use uniform4v and getuniformlocation functions for ambientProduct, diffuseProduct, specularProduct, lightPosition and materialShininess.

7. Implement both the Gouraud and the Phong shading models, with a button switching between them.

In the .html:
It has been added a button to let change lightning method using a boolean variable initialized in JavaScript. In order to use a second shader, I have to use two different programs (declared in both vertex and fragment and respectively called program_gouraud and program_phong) that will let me to use Phong model.

In the .js:
To use this Phong model, it is needed to use two program called program_gouraud e program_phong and it is needed to move program initialization in render, otherwise the switch between them will not work. Finally, in render I insert all variables passed to vertex shader insider an if-else structure that will let us to choose the shading type.