1. Replace the cube with a more complex and irregular geometry of 20 to 30 (maximum) vertices. Each vertex should have associated a normal (3 or 4 coordinates) and a texture coordinate (2 coordinates). Explain in the document how you chose the normal and texture coordinates.

-The geometry proposed is a polygon with a total of 28 vertices;
-The vertex normals are associated creating each face with the quad() function passing its vertices in an anticlockwise order, which following the right-hand rule generates normals that point outside the geometry. This allows also a correct illumination of the faces of the polygon;
-The texture normals are constant quantities (bidimensional values vec2) are assigned as well in the quad() function, but organized in triangles: each quad is divided in two triangles and the coordinates are assigned anticlockwise to the vertices of the two obtained triangles.

2. Compute the barycenter of your geometry and include the rotation of the object around the barycenter and along all three axes. Control with buttons/menus the axis and rotation, the direction and the start/stop.

-The barycenter in the code has been calculated as averaging of the vertices' values along the three axes x, y, z;
-The rotation around the polygon's barycenter is included and computed directly in the render function;

```
modelViewMatrix =translate(barycenter[0], barycenter[1], barycenter[2]);
```

-The direction of the rotation is controllable through the dedicated button which inverts the rotation at each click. Similarly the start, stop, rotation around X, Y, Z are controlled through separate buttons.

3. Add the viewer position (your choice), a perspective projection (your choice of parameters) and compute the ModelView and Projection matrices in the Javascript application.
The viewer position and viewing volume should be controllable with buttons, sliders or menus. Please choose the initial parameters so that the object is clearly visible and the object is completely contained in the viewing volume. By changing the parameters you should be able to obtain situations where the object is partly or completely outside of the view volume.

-The method chosen is the implementation of the lookAt(eye, at, up) matrix,

```
modelViewMatrix = mult(modelViewMatrix, lookAt(eye, at, up));
```

which allows the viewer to inspect the surrounding area through the buttons on the screen. In the initial position is clearly visible the polygon and the neon but they could disappear if the viewer looks left or right, or even if the viewer approaches the polygon enough;
-The viewing volume contains the whole scene in the initial parameters but it can be changed using the buttons of the Near and Far actions. These actions can also show a slice of the polygon allowing to look in the inside of it.
-The projection chosen is the perspective, because of its feature of controllability of the fovy parameter. This parameter regulates the field of view of the observer.

```
projectionMatrix = perspective(fovy, aspect, near, far);
```

4. Add a cylindrical neon light, model it with 3 light sources inside of it and emissive properties of the cylinder. The cylinder is approximated by triangles. Assign to each light source all the necessary parameters (your choice). The neon light should also be inside the viewing volume with the initial parameters. Add a button that turns the light on and off.

-The cylindrical neon light is modeled through the rendering of a classical cylinder with few key changes that gives it some particular features, allowing it to resemble a light.
First of all the cylinder is moved to the top of the canvas and then it's rotated in the correct position.
The 3 light sources are then attached to the axis of the cylinder (in the inside).
-The buttons can turn off and on both the initial light and the neon light independently, since the three new light sources are modeled in order to have uniform behavior.

5. Assign to the object a material with the relevant properties (your choice).

-The material has been chosen modifying the parameters

```
var materialAmbient ;
var materialDiffuse ;
var materialSpecular ;
var materialShininess ;
```

and in the chosen material the polygon appears polished and green.

6. Implement both per-vertex and per-fragment shading models. Use a button to switch between them.

-The switch between per-vertex and per-fragment has been implemented introducing a conditional instruction into the two shaders: in this way they adapt their behavior based on the value of the flag, which is changed by the two buttons controllable by the user.
-The differences that can be notices while switching between the two shading models are that the light is more realistic and precise while using the per-fragment shading technique.

7. Create a procedural normal map that gives the appearance of a very rough surface. Attach the bump texture to the geometry you defined in point 1. Add a button that activates/deactivates the texture.

The code implemented for the bump map generates a random data array to be attached to the polygon through the function

```
 data[i][j] = Math.random();
```

This method allows to give a rough appearance to the polygon's texture since the values generated are, as said, random.
Also two buttons have been added to the interface in order to activate and deactivate the bump map.