Exercise 1

For this exercise I have replaced the cube with a figure that has a total of 6 vertices, but since I have used gl.TRIANGLES as a method of drawing the triangles, the vertices passed to the vertex shader are 24 in all.

Each vertex has an associated normal (3 coordinates) that has been calculated by averaging the normals of the triangles of which it is a vertex.

Each vertex also has an associated coordinate in the texture. The choice of coordinate was made following this algorithm for each triangle:

1) The triangle was flattened on the plane z=0 preserving the lengths of the edges and transforming it into a 2d triangle.
2) The barycenter of the triangle was calculated.
3) The distance of the 3 vertices from the barycenter was calculated and the longest one was taken.
4) The triangle was resized by dividing each coordinate by this maximum distance found.
5) The triangle was moved so that the barycenter corresponds to the origin.
6) The triangle was moved so that the lowest vertex has coordinate y = 0.
7) The triangle has been moved so that the leftmost vertex has coordinate x = 0.
8) At this point, the coordinates of the vertices of the triangle obtained are the coordinates of the texture.

Exercise 2

The barycenter has been calculated by averaging all the vertices of the shape.

For the rotation along the 3 axes around the barycenter, the code provided in the homework has been modified by inserting a translation from the origin to the barycenter before the rotation and after one from the barycenter to the origin, therefore, reading it backwards, I bring the shape with the barycenter to the origin, make the rotation and then bring it back to its original position.

Rotation along the 3 axes, change of rotation and start/stop are managed by buttons.

Exercise 3

The viewer position is set in perspective mode by looking towards the barycentre and positioning according to theta and phi viewing angles. All parameters can be changed with sliders and the following combination allows a double cut:

- zNear = 2.91
- zFar = 3
- radius = 6.35
- theta = 0
- phi = 0
- fov = 10
- aspect = 1

The code contained in "perspective2.js/perspective2.html" was used and modified to create this part of the code.
One interesting thing I have added is the invariation of light with the viewer's movement. To do this I simply multiplied the modelview matrix with each of the four lights.

Exercise 4

The cylinder was constructed using this idea: if we have two circles, one on top and one on bottom, with their coordinates saved in the same order, it would be possible to construct a square with the coordinates i and i+1 of the circle on top and the circle on the bottom.
To make these circles, I created a function which, given a centre and a radius, runs with an angle step along all the coordinates of a circle calculated with cosine and sine of the angle. Based on the angle step, I can decide how much more accurate the cylinder should be. I then inserted one light near the bottom, one near the top and one in the centre of the cylinder.
To make it look like neon, the effect that these 3 lights have on the cylinder has been eliminated but an emissive property has been added (by adding a simple red (1,0,0)) which creates a lighting effect. When this emissive property is active, my shape is affected by these 3 lights by projecting a red colour. This neon can be activated and deactivated by a button.
The three lights inserted in the neon, being three lights almost coming from the same point, have a very heavy effect on my shape, which is why an intensity of one third was set for each light and to better see the effects of this light on my shape the ambient light value was set to 0.

Exercise 5

My shape has been given an approximately blue colour with light blue reflections. This colour was chosen to contrast with the red light of the neon light to be able to see a colour tending towards magenta when the material illuminated by the white environmental light is also illuminated by the neon. The cylindrical neon, on the other hand, has a light grey, almost white, material with white reflections. This choice was made to have the effect of a real neon.
The general formula for the final colour of the shape was calculated in the vertex shader or in the fragment shader using this formula: $emissive + \sum_{i=0}^{Lights}(ambient + diffuse + specular)$

Exercise 6

The code contained in shadedSphere3.html (per-vertex) and shadedSphere4.html (per-fragment) was used for per-vertex and per-fragment shading. A button is used to switch the value of a variable passed as a uniform, allowing a simple if to change from per-vertex to per-fragment shading and vice versa.

Exercise 7

The code contained in bumpmap.js/bumpmap.html was used and modified to create a bumpmap.
A texture was created by generating a random number for each texel. Then with the procedure to make a normal map these random numbers caused ripples in the texture to make the object look very rough. This normal map was passed as a texture. In the vertex shader we switched to tangent coordinates using, for each triangle, its normal and tangent. The tangent was calculated by taking the edge of the triangle that allows for a bitangent, calculated as the vector product between normal and tangent, towards the inside of the triangle. In this new space the lights were calculated and then passed to the fragment shader where the final colour was calculated. A more uniform effect of this texture was obtained thanks to the texture mapping done in exercise 1.