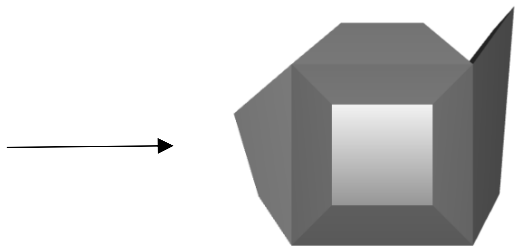
Homework 1 - Interactive Graphics

Leandro Maglianella - 1792507

I have tested my code using both Google Chrome and Firefox browsers.

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 object I defined is composed of twenty-eight vertices expressed in four coordinates *(x, y, z,* 1*)* and twenty-six faces: every face is built as a quadrilateral, formed by two triangle meshes, defined as a sequence of vertices following the right-hand rule (counter-clockwise encirclement of outward-pointing normal) to be outwardly facing polygons. When creating my solid, I was inspired by the famous model of the Utah teapot and I tried to emulate it: my result may resemble its shape, however due to the few vertices used they are not actually very similar.



A normal, formed by 3 coordinates, is assigned to each vertex and calculated following the formula for triangles' normals as the cross product of two vectors, each obtained by subtracting two pairs of vertices; as said, in this way the normal will be outward-pointing and I will use it later for the shading model.

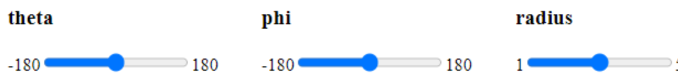
A parametric texture coordinate, formed by 2 coordinates, is also assigned to each vertex. The chosen parametric texture coordinates are used, for each mesh, combining them with the texture image so that we can perform an interpolation to find proper texture elements in world coordinates, therefore assigning a correct pixel of the image to every point of the object (exercise 7).

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.

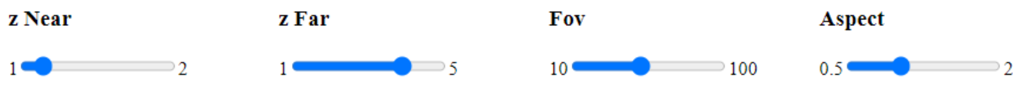
I assumed the mass of the object to be concentrated only in its vertices as if they were point masses: in this way the object's barycenter is easily computable as the average of vertices' coordinates. Buttons have been included to toggle rotation, orientation, axis of rotation (along X, Y or Z axis) and center of rotation (origin or barycenter). Also, two buttons control the rotation velocity. The rotation around the origin is made multiplying the *modelViewMatrix* with three rotation matrices the rotation around the barycenter is done pre-multiplying a translation of the object to the barycenter and post-multiplying a translation of the object to the origin.

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 *modelViewMatrix* is computed using the *lookAt(eye, at, up)* function, where *eye* is the position of the camera (which is clearly the viewer position), *at* is the position we are looking at, *up* is the orientation of the camera. As said, the viewer position refers to the *eye* parameter which is expressed using the polar coordinates and it is controllable by varying *radius*, *theta* and *phi* variables: it is possible to interact with these three values through three sliders (in particular, the angles variables can vary between -180 and 180 to enable a complete 360-degree view of the object):



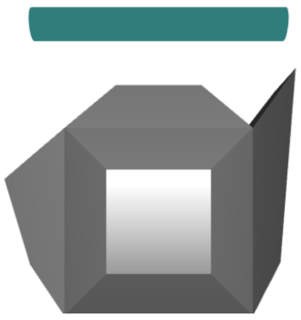
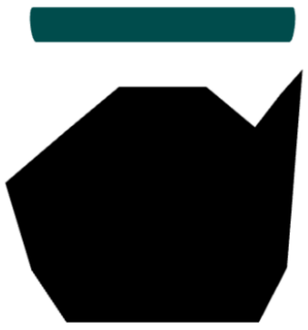
The *projectionMatrix* is computed using the *perspective(fov, aspect, near, far)* function, where *fov* identifies the camera's field of view along the y-axis, *aspect* is the width-height ratio of the size of the canvas, *near* and *far* are the minimum and maximum distances where objects have to be to be in the viewing volume. It is possible to interact with these four values in order to control the viewing volume through four sliders:



The chosen initial parameters ensure that the object is completely visible and contained in the viewing volume and changing parameters I am able to obtain situations where the object is partly or completely outside of the viewing volume.

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.**

I have removed the basic pre-implemented light and I have inserted a cylindrical object inside the viewing volume, placed three light sources in it and I have set all their RGBA parameters (i.e. diffuse, ambient, specular and position), later used for exercise 6. I have added a button to turn the lights on and off: specifically, when the lights are switched off every colour computed by the shaders is reset to zero, therefore my object becomes black (the cylindrical lamp have only its emissive term colour when the lights are off, which is set to (0.0, 0.3, 0.3, 1.0)). Anyways, I kept the canvas background colour to white to make my object visible even with switched off lights. I computed another modelViewMatrix to which the rotations of exercise 2 are not applied and I used it to represent the cylinder: in this way the cylinder does not move in the scene.
These are the results when light is on and off are the following:

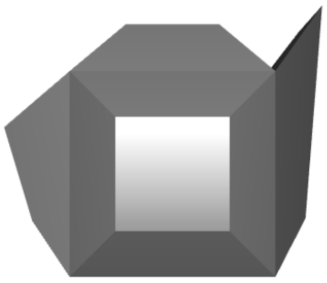
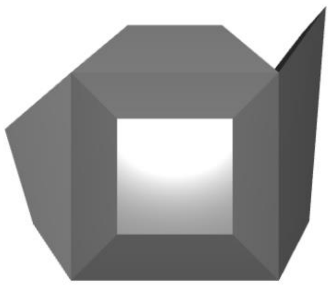



5. Assign to the object a material with the relevant properties (your choice).

As said in exercise 1, I tried to assign the material to my solid (i.e. materialAmbient, materialDiffuse, materialSpecular, materialShininess) so that it is as similar as possible to the teapot image: the final result has a silver colour. I assigned a very large shininess to my object (200) just to increase the visibility of the shading model's effect in exercise 6: in fact, in this way it takes on a metallic appearance which reflects light more clearly.

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

I have implemented Phong shading model by computing and adding all the ambient, diffuse and specular contributions for all of the three lights, both per-vertex and per-fragment in the html file. I created a button to switch between them. In per-vertex shading, the lighting is evaluated at the vertices and interpolated only along them, while in per-fragment shading this is done for each fragment: the results are therefore faster and less computationally heavy using the per-vertex shading but they also appear more tessellated and less precise. By design, I have implemented this exercise so that this shading models do not apply to the cylinder. These are result's examples of the two shading models applied to my object:

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.

Bump mapping is done by perturbing the normals across the whole surface of an object: this is made by displacing the normals in the tangent plane. Therefore, as for exercise 1, I started adding another coordinate for each vertex: a 3D tangent. I created a bump texture assigning random data to it to give it a rough appearance: my bump data is a 32×32 image of random values between 0 and 2 (initially the maximum value was 1 but I incremented it to amplify the rough effect). Finally, a normal texture array is defined using the bump data and configured as texture, which is sent to the fragment shader where the normals' perturbation take place. Using the new normal, as before I computed the resulting colours using the Phong per-fragment shading model. By design, I have implemented this exercise so that this texture does not apply to the cylinder. I created a button to activate and remove the texture from my object, these are some result's examples: