Interactive Graphics: Homework 1

Luca Polenta 1794787

May 2nd, 2021

All points of the homework have been tested on Windows OS and have proven to work on Chrome, Firefox and Edge (chromium version).

1: Replace the cube with a more complex and irregular geometry of 20 to 30 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 consists of 27 vertices in the following form: (x,y,z,1.0). The vertices are joined to form 3 pyramids with a square base and 4 trapezoids for a total of 31 faces. Triangles are obtained as degenerate squares. All faces not visible from the outside have not been rendered. Each vertex has a normal and it is computed as the cross product of the vectors obtained by subtracting two pairs of vertices, that is, the two sides of each face. All vertices of the same face have the same normal. The texture was generated from an image. Each vertex is associated with a corner of the image and then WebGL maps each point of the object to a specific pixel of the image on the screen.

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 of an irregular object can be computed using the static momentum, which relates the barycenters and volumes of each elementary sub-figure that composes the irregular object. However, the current irregular figure has a high symmetry that compensates for all the displacements of the barycenter, therefore not making it deviate from the origin. The only portion that shifts it is the pyramid at the top of the figure, so the barycenter of the actual irregular figure is in position (0.0, 0.02, 0.0). To rotate around the barycenter it is necessary to pre-multiply the modelview matrix by the translation matrix, which is an identity matrix where the fourth column is the coordinate of the barycenter (with the addition of a 1.0 as fourth element), and then post-multiply it by the inverse of this translation matrix only after having compute the rotation (so by the translation matrix where the last column has the coordinates of the barycenter changed in sign).

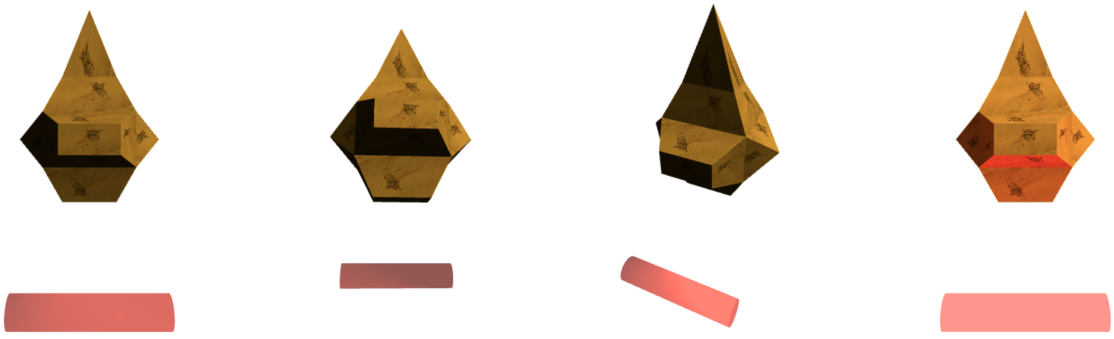
3: Add the viewer position (your choice), a perspective projection (your choice of parameters) and compute the ModelView and Projection matrices in the Javascript application. The viewer position and viewing volume should be controllable with buttons, sliders or menus. Initially the object must be clearly visible and contained in the viewing volume.

The viewer position is referred to the eye position. It is expressed using the polar coordinates and the corresponding parameters are: radius, theta and phi. Radius is the distance from the origin, theta and phi are the angles that allow to change the camera view. The radius can be changed between 0.1 and 4.0, theta between -180 and 180 and phi between -180 and 180.

The modelViewMatrix is computed using the lookAt function, where "at" consists in the position we are looking at, instead "up" is the orientation of the camera and "viewerPos", as said before, is the position of the camera.
The parameters of the perspective view are: Near, Far, Fovy and Aspect. Near controls the near plane distance from the camera for the object and it can be changed between 0.1 and 1.0; Far controls the far plane distance from the camera for the object and it can be changed between 3.0 and 5.0; Fovy identifies how wide the eyes open along the y-axis and it can be changed between 12 and 100; Aspect is the ratio of the size of the canvas and it can be changed between 0.5 and 2.0. All these parameters can be modified by the user through the use of sliders and there exist a button in order to reset the original values.

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 cylinder is obtained through a function present in the book code and that has been placed in a separate javascript file called "geometryCylinder.js". It doesn't rotate with the irregular object and it has been placed in a lower position along y than the other and also translated along the z axis, so it is not exactly below the irregular object. Furthermore, the cylinder is opaque to allow the passage of light as a neon. The predominant colour is red to make its effect more visible when the lights are on. It still appears dark red even when turned off. This is a desired effect for more realism as neon is technically a white gas and if the light is coloured, its aspect depends on the colour of the cylinder in which it is contained. Furthermore, due to the fact that the cylinder is opaque, it is also subject to external light, so when the html page is loaded, the cylinder appears a little bit illuminated by the original light. On the other hand, when theta and phi are changed and the neon light is moved behind or in front of the irregular wooden object, it becomes darker or lighter, and even quite illuminated if directly subject to external light. There is also an emissive component when the 3 lights inside it are on, but it is not too accentuated to avoid having a too bleached red colour. This is also a choice to have a realistic effect: in fact, when the three lights inside the cylinder are on, the shades on the irregular object have been set as an abundant red and only an quite red neon light could produce this effect. With higher emissivity values, a more bleached and less reddish neon would have been obtained, which should have led to a more bleached colour even on the irregular object. There are also some images to show the effects described:



Light Off, but Slightly Illuminated Externally | Dark Because Occluded | Externally Illuminated | Light On

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

The colour that is perceived on the object depends on the light and on the properties of the material. These

properties are: the materialAmbient that defines the colour of the object in the presence of ambient light; the materialSpecular that defines the reflection of the light source on the surface; the materialDiffuse that is the primary colour of the object; the materialShininess that controls the shining of the object. They have been set as follows:

```
var materialAmbient     = vec4(0.3, 0.22, 0.0, 1.0);
var materialSpecular    = vec4(0.32, 0.32, 0.32, 1.0);
var materialDiffuse     = vec4(1.0, 0.775, 0.0, 1.0);
var materialShininess   = 100.0;
```

Given the texture used, the aim of these values was to obtain a wood effect, as a carved object. Moreover, due to the values chosen, it appears dark. Both these values and the light values are not always set at high levels to prevent the object from being overexposed to light when the neon light is on.

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

In per vertex, the computation of the shading is performed for each vertex in the vertex shader (Gouraud Shading), while in the calculation of the per fragment it is carried out in the fragment shader (Phong Shading). The difference is mainly noticeable when calculating curved objects such as spheres and light reflections because with Phong Shading the light and shadows are better defined and the curves are smoother. In the current object there are no curved objects and with the current texture the change in the reflections of lights and shadows is not excessively visible. In any case, as a further proof of correct implementation, the values of the flags and variables that are modified by the user are printed on the console.

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 base code was inspired by the bump texture code in the book. To produce a very rough texture, many small squares were computed as textures via 3 nested loops. Given the small size of the texture components, they are not very visible from a distance, therefore when the rough texture is enabled, fovy is automatically decreased to zoom towards the object, and vice versa when it is disabled. Obviously the user can always change it using the sliders. Furthermore, to give a real sense of roughness, 3 vectors are needed for each vertex: the normal, the tangent computed as the side from which the normal is calculated and a third vector that is computed as the cross product between the previous two vectors. Using these vectors it is possible to emulate the alternation of normal vectors during the rendering process. In addition, to remove some warnings, the cylinder tangents have also been computed, even if they are not used as the texture is applied only to the main object. Finally, this rough texture is available as long as the neon light is off and the per-fragment shader is enabled (as it is a specific feature of this type of shading), so there are a series of automatic controls that prevent these features from overlapping.
Finally, in the code there is a second commented version of the rough texture which has squares twice the size of the current ones. It can create a slightly less rough effect, but it is more visible from further away. However it is commented because the text requires a very rough effect.