Interactive Graphics

Homework 1 (works fine only in Google Chrome browser)

Made by Alessandro Lambertini

Mat: 1938390

1 First Step

I choose to replace the Cube with a geometry with:

- 23 Vertices;
- 46 Triangles.

My shape also has for each vertex:

- position (vec4);
- normal (vec4);
- color (vec4);
- texture coordinate (vec2);
- Tangent (vec4, this will be explained in chapter 6).

To evaluate the normal, for each triangle I apply the dot product between 2 edges, each of these edges has been evaluated by subtracting 2 vertices.

```
var t1 = subtract(figure.vertices[b], figure.vertices[a]);
var t2 = subtract(figure.vertices[c], figure.vertices[b]);
var normal = normalize(cross(t1, t2));
normal = vec4(normal[0], normal[1], normal[2], 0.0);
```

Furthermore, for the texture coordinate I decided to divide my texture in 4 squares, I used the first for my shape and the second for the cylinder.

```
// My Shape
vec2(0.05, 0.05), vec2(0.05, 0.45), vec2(0.45, 0.45), vec2(0.45, 0.05),
// Cylinder
vec2(0.55, 0.05), vec2(0.55, 0.45), vec2(0.95, 0.45), vec2(0.95, 0.05)
```

2 Barycenter

To compute the barycenter I choose to consider the density uniform distributed, so for each triangle I computed the average for the x, y and z coordinate and then the area of the triangle.

$$Pos_i = \frac{Vert_{0,i} + Vert_{1,i} + Vert_{2,i}}{3} \qquad Area_i = \frac{1}{2} * \left\| edge_{0,i} \times edge_{1,i} \right\|$$

And then to compute the final position I sum all over the positions times the area.

$$FinalPos = \sum_{i=0}^{n} Pos_i * Area_i$$

I compute the barycenter only for my shape because is the only shape that has to move/rotate.

For the cylinder I set the position of the barycenter to (0.0, 0.0, 0.0, 1.0).

To rotate and translate the figure I before compute the rotation/translation matrix, and then apply the rotation around a fixe point, so translation to the barycenter, rotation, translation to the -barycenter.

```
mult(translate(finalBaryPos), rotationMatrix);
mult(rotationMatrix, translate(-finalBaryPos));
```

3 ModelView and Projection

For the model view I used the **lookAt(**eye, at, up**)** function with constant "at (vec3(0.0, 0.0, 0.0) the origin)" and "up (vec3(0.0, 1.0, 0.0))".

For the parameter "eye" I choose to use 3 sliders to change the radius (distance from "at"), the angle theta (vertical angle) and the phi angle (Horizontal angle), in order to change the point of view of the scene.

For the projection I used the **perspective**(fovy, aspect, near, far) function with all the 4 parameters changeable with sliders.

In order to makes clearly visible, the object I choose as initial parameters:

- Radius: 4;
- Theta: 0,3490;
- Phi: 0;
- Near: 0.1;
- Far: 10.

I choose a low value for Near and an high value for Far in order to make my shape clearly visible.

4 NeonLight

I have approximated the cylinder as an octagon with:

- 18 Vertices;
- 32 Triangles.

The cylinder has an emissive propriety as vec4(5.0, 5.0, 5.0, 1.0) in order to increase the light inside of it and to my shape I set an emissive proprieties vec4(1.0, 1.0, 1.0, 1.0) in order to have a normal behaviour.

I also add 3 lights in 3 different points of the cylinder, with the same proprieties for the ambient (vec4(0.2, 0.2, 0.2, 1.0)), diffuse (vec4(1.0, 1.0, 1.0, 1.0)) and specular (vec4(1.0, 1.0, 1.0, 1.0)).

I choose these parameters in order to have a strong white light and the surfaces that are not illuminated are black.

5 Material

I used only one material both for my shape and the cylinder.

I choose these parameters to get a lucid plastic surface, and these parameters are:

- Ambient: vec4(0.2, 0.2, 0.2, 1.0);
- Diffuse: vec4(0.7, 0.7, 0.7, 1.0);
- Specular: vec4(0.7, 0.7, 0.7, 1.0);
- Shininess: 200.0.

6 BumpMap

In order to create a rough surface, I choose to set the first square of the texture with random numbers between 3 possible numbers.

To use the bump map, I add the tangent to the set of buffers, as tangent I took one edge of the triangle.

The bump map has been applied only in the per-fragment because it is not applicable in the per-vertex, so in the per-fragment I added an if in order to compute the light in a different way, also considering the bump map.