# Homework 1 - Interactive Graphics

## 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 how you choose the normal and texture coordinates.

The object is composed of twenty-five vertices expressed in four coordinates: (x,y,z,1) and twenty-six faces. Twenty-two faces are formed by two triangles that build a rectangle and four faces are only triangles.
Each vertex has a normal, it is calculated as the cross product of the vectors obtained by subtracting two pairs of vertices because, given that the figure that is made up of triangles, I want that each point on the surface of the face to have the same normal, therefore the vertices themselves will also have the same normal.
It would have been different if I had a sphere, because the effect I wanted to have would have been a variable reflection according to the single point of the surface.

Instead for the texture (see later in 6), for each vertex I will give the corresponding coordinate of the image. For example, for a triangle, given 3 pairs of coordinates, with the texture I will interpolate the coordinates to define all the points of the triangle and consequently I define the coordinates to be applied in those points, i.e. the pixels of the image that they must be to consider.

## 2 Add the viewer position, a perspective projection and compute the ModelView and Projection matrices in the Javascript application.

The viewer position refers to the eye position that is expressed using the polar coordinates which are parameters: radius, theta and phi.
**Radius** is the distance from the origin, **theta** and **phi** are the angles that allow you to change the camera view.

Radius is between **1.0** and **4.0**   Theta is between **-180** and **180**
Phi is between **-180** and **180**

I considered both Theta and Phi between -180 and 180 to give a complete 360 degree viwe of the object.
In order to change the eye value, I have implemented sliders that change the values of theta, phi and radius.

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 eye, as said before, is the position of the camera. All three are 3-dimensional vectors.
Instead the projectionMatrix is computed on the basis of projective projection. Which is very realistic, because it seems center of projection coincided with the user's eyes.
The prospective uses as parameters: *fovy, aspect, near and far.* **Fovy** identifies how wide the eyes open along the y-axis, **aspect** is given ratio of the size of the canvas, **near** control the near plane distance from the camera for the object and **far** control the far plane distance from the camera for the object.

fovy is between **10** and **40**   aspect is between **0.5** and **2**
Near is between **0.0** and **1.0**   Far is between **1.0** and **3.0**

The parameters of the view poisition and perspective projection can be modified by the user through the use of sliders.

## 3 Add two light sources, a spotlight in a finite position and one directional. The parameters of the spotlight should be controllable with buttons, sliders or menus. Assign to each light source all the necessary parameters.

### 3.1 Directional light

Directional lighting assumes the light is coming uniformly from one direction. I set an RBGA for directLightAmbient, directLightDiffuse and directLightSpecular. Instead for the directLightDirection, I have the Cartesian coordinates (x,y,z,0). This variable is exactly the direction, being a vector with component 0 at the end. My light comes from the left in order to highlight the object better.
For directional light the normals and the position are calculated in the vertex shader, which will be supplied as output variables to the fragment shader. The fragment shader interpolates the received variables and calculates the color, hence the Ambient, Diffuse and Specular variables.

### 3.2 Spotlight

In this case the spotlight is no longer directional but fixed light. In addition to the directional light, in the vertex shader I have one more variable, the Spotlight Position because it is also located at a certain point and points in one direction. All parameters except direction vector has as last coordinate 1.

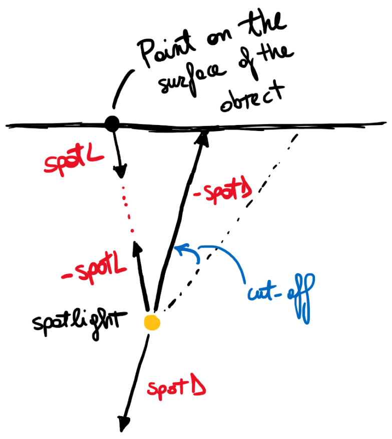

Figure 1: Spotlight

The main difference between the spotlight and the directional is that in the directional what I care about is only the direction, instead in the spotlight I'm interested in the point where exactly that light is located.
From a practical point of view, I always go to use the directions, because the position of the spotlight will always be subtracted from "pos", in the vertex shader, thus obtaining a vector. Then i compute the angle between negation of this vector and the spotlight direction, as we can see in the figure. Consequently I observe that if the cosine of this angle is smaller than the "uSpot", the cut-off, it means that the angle is greater than the points that are inside the cone of the spotlight. It follows that the point I am pointing for, is outside the cone, otherwise it is in the spotlight region.
The parameters of the spotlight can be modified by the user through the use of sliders.

## 4 Assign to the object a material with the relevant properties.

The color that is perceived on the object depends on the color of the light but also on the properties of the chosen material. Both of the two lights described above influence the material, in particular the ambient variable of the spotlight is always present even if I am not considering the points that are inside the cone of the spotlight. In fact, by changing the spotlight's ambient variable the figure change colour.
The **materialAmbient** defines the color of the object in the presence of ambient light, the **materialDiffuse** is consider as primary color of the object, **materialSpecular** can be thought of as the reflection of the light source itself on the surface of the object, and finally the **materialShininess** controls the size of the highlight: larger numbers produce smaller highlights, which makes the object appear shinier.
The material used was taken from the examples illustrated by the professor, I tried other materials but none of them particularly excited me. The values choosen are the following:

```
    var materialAmbient = vec4(1.0, 0.0, 1.0, 1.0);
    var materialDiffuse = vec4(1.0, 0.8, 0.0, 1.0);
    var materialSpecular = vec4(1.0, 1.0, 1.0, 1.0);
    var materialShininess = 20.0;
```

With this I can highlight even more the structure of the object, which should represent a birdhouse.

## 5 Implement a per-fragment shading model based on the shading model.

Here, I have implemented the proposed cartoon shading algorithm. In particular, it is applied on both lights: directional and spotlight. The variables considered were the Ambient, the Diffuse, the Position of the light source and the normal of both lights.
For the directional, the position of the light source consists in the direction component, instead for the Spotlight the position of the light is given by the vector calculated using the component of the position of it (therefore a finite point). The Kd is calculated, i.e. the product between the light source and the normal, and if the value of it is less than 0.5 then consider only the the product between the ambient variable of the light and the ambient of the material, otherwise in addition to this product we have the product between the diffuse of light and the diffuse of material.
Obviously this is done both for the directional and for the spotlight, in fact, there is then a concatenation between the results given by applying this filter on both lights.
In the algorithm I wanted to consider the global ambient variable equal to zero because I am only interested in using the variables of the Ambient of the two lights.

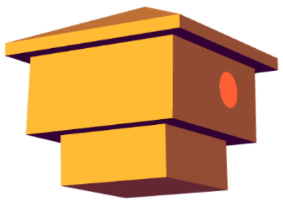
Figure 2: Object without cartoon shading.

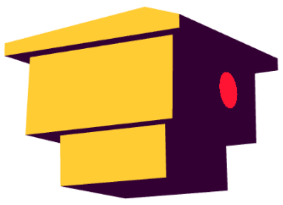
Figure 3: Object with cartoon shading

As shown in the Figure 2,the directional light covers the whole image and is of a yellowish color, instead the spotlight is the red "circle" on the right. In the Figure 3 the effect is to have a kind of flattening of the lights, you no longer have the nuances of the light, but it is as if it were all uniform.
In fact, where even only the light was in dim light, with this *cartoon shading* it turns out to be totally dark.
Cartoon shading can be disabled and enabled by using a special button, more specifically, a control variable has been inserted in the fragment shader, which allows you to apply or not the **shading** to the object. In this way the efficiency is lost because the data is already calculated but it was the plausible solution that I could apply.
The button has been inserted to observe the behavior of the lights even without applying *shading*.

## 6 Add a texture loaded from file, with the pixel color a combination of the color computed using the lighting model and the texture. Add a button that activates/deactivates the texture.

The textures are usually generated from an image, in fact, in this case I apply image base texture mapping. With this technique I have the disadvantage of downloading the image into RAM or GPU memory, but having the image, I will not take much time to render the image and moreover, the object will have more natural and realistic qualities.
Textures in WebGl are arrays and their element are texels and each point of the object is mapped to a specific pixel on the screen. The texture can be enabled and disabled through the use of a button, in the same way it was done for cartoon shading.