Interactive Graphics homework 1

Flavia Ferranti 1707897

May 2020

1 Introduction

The purpose of this assignment was to take confidence with the Javascript library WebGL used to create 3D animated figures. In order to realize the homework I based my studies principally on the code seen during lessons time and I investigated more, when needed, using the book: *Interactive Computer Graphics* by Angel and Shreiner. Let's see more in detail the realization of my work and the technique used.

2 Irregular Geometry

The first step was to create an object that had from 20 up to 30 vertices. In order to obtain a more complex figure than a cube I started playing with coordinates for understanding the space and the proportions. At the beginning I stated adding cube over cube and studied the result. Finally, inspiring to the technique used for drawing by hand, where each object it's formed by a composition of basic polyhedrons, I decided to construct one of the most difficult thing to draw on paper: a hand. The first thing made, that constitute the basis of my hand, was the palm; it was created by simply changing the original cube formed by squares and substitute them with rectangles. After the completion of this figure, I added the thumb having approximately the same height of the palm just slightly smaller. Then it came the moment for the other fingers and to make everything more realistic I added a little piece of wrist. So my figure, made of 5 different polyhedron, has 180 vertices in total (considering also the internal one of the triangles composing each face) and 22 distinct visible vertices. Before creating the coordinates of the vertices I sketched the desired result on paper in order to place the origin of the axes in the center of the figure and then I put everything in a structure. After that, to actually construct the figure, I had to add to the function *colorCube* the order of the new vertices, in order to pass them to the vertex shader. This order follows the principle of the clockwise and counterclockwise for indicating respectively the backward faces and the front faces of each polyhedron.

3 Figure viewing

Once the figure was complete it wasn't particularly visible through the canvas because many part were clipped out of the viewing volume, in fact, only those objects that fit inside the angle of view of the camera appears in the scene. From that comes the need to positioning, orienting and projecting the camera.
The model-view transform the object coordinates into, at first, world coordinates and then eye coordinates. Since the camera is pointing in the negative z-direction and the object was already centered in the center of the axis, this means that in order to see everything in the scene the default position and orientation of the camera is not good. The thing to do is to position the frame of the camera with respect to the frame of the objects. My first approach was to move the camera backward along the negative z-axis with a translation matrix, to see fully the object, and in order to observe it from different point of view consider a rotation matrix (that is still implemented for having a clear object rotation along each axis).
A more easier computation with good result, is given by the use of the *look at function*. In this case we just need to specify where we want the camera to be, with the *eye* term, indicate the position of the thing we want to look at, through the *at* parameter and finally define the *up* term, if we want the camera not pointing straight to the object. In my case the *at* point coincides with the origin because my object is centered. Also *up* has been left with the default value of the y direction, while the *eye* parameter can be changed considering the distance from the origin given by radius and considering the angle that can be modified by the interaction between the terms θ and ϕ.
Other than the viewer position we can also specifies the clipping parameters through a projection. When we use perspective projection everything converge to the point of view, that is the origin. We can chose between two different perspective projection *frustrum* and *perspective* but, since with frustrum is often difficult to get the desired view, I decide to use the latter. In this kind of projection I specified four different parameters:

- near: is the distances measured from the center of projection to the front clipping planes. I decided to put it at 2 but it can be decreased to zero in order to see everything form the origin
- far: is the distances to the back clipping plane and so it puts a limit in what you see
- fovy: is the angle of the field of view along the y-axis and more precisely is the angle between the top and bottom planes of the clipping volume.
- aspect: is the ratio between width and height of the projection plane

The first two are defined in terms of distance with the viewer.

4 Light, materials and color

As request, I had to compute two different source of light, the directional and the spotlight. They are both particular case of the point lightning. For what concern the directional: it is characterized by a light source far from the surface (can be considered at infinity), creating the effect of parallel rays of light that illuminates the object. While the spotlight is characterized by a limited angle at which light from the source can be seen, that's why we can only see a portion of this light.
For each light source I had to declare their properties, that depends on the material-light interactions: ambient, diffuse and specular terms for each of the three color in RGB, that are the ones that the human eye sees. Since the specular component was not required in the shading calculation I omitted the computation. The other two properties, instead, was defined in homogeneous coordinates since the material of the object is not opaque. The ambient light is the one that enlighten the scene in a uniform manner, so it's intensity is the same at every point on the surface. For what concern the diffuse component it can be seen as the vertical component of the incoming light. For these reason, in order to compute it right, I had to calculate the angle between the normal of each point in the object and the direction of the light (the angle of incidence). The cosine of this angle is given by the dot product of these 2 vectors only if they are unit-length, that's why I had to normalize both the vectors before doing any computation. Notice also that, if the light source is below the horizon the dot product between the vectors will be negative. So, in order to avoid this I multiplied the diffuse component by a coefficient that takes the value zero instead of a negative one defining the intensity of the color : *coefficient = max(dot(L, vNormal), 0.0)*. This kind of light can also be influenced by an attenuation factor due to the distance of the object from the light source: $attenuation = \frac{1}{k_c + k_l d + k_q d^2}$. In the case of the directional light I consider it positioned at infinity so I omitted the calculus of the attenuation.
As said, the computation of the light requires the use of the normal vector of each face of each polyhedron. Since the surface of my hand is constituted by flat polygons, the normal can be calculated using three noncollinear points and computing the cross product of the subtraction of them. Furthermore, this normal will be equal for all the points of this polygon face and its direction will depend on the order of the vertices specified with the function *quad*.
Summarizing, each light source has a color and a direction, the computation of the color described above is the same for both directional and spotlight, while things change for what concern the location of the light. Taking into account the directional light, at the begging, I realized the computation of its diffuse component in the vertex shader. This because all point receives the same parallel light, because the direction of the source is the same for every point. This means that the position of each of them is not relevant and the diffuse contribution at each vertex is identical.
Since in the fifth point it's required a pre-fragment shading, I moved all the calculus in the fragment. Although having a lighting model applied to each fragment could slow the process of calculation (because the fragment are much more than the vertices, and we have an independent lighting calculation for each fragment). Moving the computation in the fragment requires also to move the calculation of the normalization of each vector, because doing it in the vertex could cause some changes during the interpolation produced by the rasterizer.
In the spotlight we have that the normal is the same at each vertex but since the source is near, the vector from any point on the polygon to the light source will be different, so we take into account this factor while calculating the position of the light: *uSptPosition.xyz - pos* where pos is the position of the vertex considered in eye coordinates.
Even in this case the normalization has been done in the fragment, with the calculus of the coefficients for the diffuse component and the attenuation factor. The real difference between the other light source is that in order to estimate the portion of the object illuminated I had to compute the angle between the position of the light and a vector to a point in the surface. If this angle ϕ is less than the limit angle expressed as the *cutoff* parameter than I will see the diffuse component of the light in that point otherwise this component will be put to zero. In order to make the action of the limit more evident, I add the possibility to control it

and see the spotlight increasing or reducing. Of course when the limit reaches 180 the spotlight will become a point light. Other possible actions with the spotlight will be the possibility of changing it's location on the surface, and to change the intensity of the light distribution. To make it more realistic the light should be clearer on the center of the cone and fading on the border so I added this fading factor that is an exponent added to the cosine of the ϕ angle: *exponent:* $cos^e(\phi)$. At the end it will also be possible to control the attenuation factor coefficients.

All the color parameters specified with the ambient and diffuse coefficients of the light has to be linked with the material properties. A material color can be seen as the quantity of primary color RGB that the object reflects. That's why, also for the material, we have the three terms of ambient, diffuse and specular (last one omitted). The difference between light and material values is that the first represent the intensity of the primary color, while the second how much of that intensity is reflected by the object. So, to create the final color, the material and light color has to multiplied. For this reason I tried to come up with values that would mix up giving as result the color of a human skin. The diffuse and ambient material values are similar: they both reflect principally red and blue with a piece of green in order to create a pink effect. The directional light is based on the idea of a darkened white light, in order to limit the brightness of the object, while the spotlight is based on the yellow color to imitate the color of a torch or a lamp.
To compute the interaction between all this, and so the final color, we sum up the ambient and diffuse contribution considering that the ambient term is given by the product of material and the light source values and the same holds also for the diffuse. Notice that in this case we have two lights so their color terms can be summed up and if the result is bigger than 1 it's automatically reduced.

In particular, to output the final color we use a specified shading model that is the *Cartoon shade*. This model exploit the non-negative incidence angle to calculate which color must be given as output. If this angle, and so the intensity of light, is greater than 0.5 then we compute the color as ambient product plus diffuse product, plus global light (representing the light of the scene when no other lights are present). Otherwise the color is given just by the ambient term. Notice that, since we have two lights, for each point must be considered the angle between both the normal and directional light and also the normal and the spotlight. With this type of model I noticed that, while rotating the figure, some part of it, even if illuminated by the spotlight, are not considered because the light there is not enough bright. In order to avoid this effect I also created a button that consider only the directional condition and so, add to the final color the diffuse component of the spotlight even if the condition is not respected but we are inside the limit angle of the cutoff. Since this model had to be computed for each fragment, as already said, I put the normalization, the computation of the angles and the calculation of each terms and products in the fragment shader.

5 Texture

As last step of this assignment I computed the texture, that as been implemented as a contribution to the color of the figure previously computed.
For making the hand more human-like I added the texture of a human skin, scanned by a photograph and passed to the *JS* file. As required, I used a two dimensional texture pattern, so with texture coordinate of two values. Usually the texture coordinates are scaled to vary over the interval [0,1] and this coordinate are mapped to each face of the geometric object on which they are positioned. For this reason I had to create 4 different kind of texture coordinate. The basic one was used for each small component of the hand because the value range varies over the interval [0,0.5], while for the big components as the palm or the fingers it was used [0,1]. This was necessary to make the weave of the texture homogeneous in every part of the figure. Other than these coordinates, the real problem was introduced by the two distorted figures: the thumb that it's like a truncated pyramid, and the part of connection between the palm and the wrist. For the thumb I had to reduce the x coordinate component of the superior part, while for the other polygon I increased the y-value of the lateral right face with respect to the left.
Each of this coordinates were then stored into an array and send them to the vertex shader so that they can be interpolated by the rasterizer and send them to each fragment. In the fragment we also use a texture object created in the js that is used to specify the image, the image size and defines each parameter of the texture for example how a value outside the range is treated.
All these information are put together in the fragment thanks to the sampler variable, that in our case will be 2D, At this point we are finally ready to multiply the color given by the cartoon shade with the texture and obtain the final life-like result.