Eleonora Beatrice Rossi
1749038

Homework 1

1. Replace the cube with a simple model of a table. Each vertex should have associated a normal and a texture coordinate.

In order to create a table of 35 vertices I first modified the initial cube to transform it into a large and flat parallelepiped.(8 vertices). After which I have calculate the 28 vertices that compose the 4 legs of the table in this way: given figure 1, the vertices of the plane in common with the legs are necessarily A, D, E and H, so these will be my shared vertices. Starting for example from E (see figure 2) I have to calculate the other 7 vertices that are used to create a thin and long parallelepiped. so if E is the vertex (-0.8, -0.05, -0.5) vertex p4 will be a vertex with the same x, the same z, but different y as it is shifted in height: (-0.8, -0.5, -0.5). vertex a3 will be a vertex with the same y, the same z, but different x because it is displaced in width. (-0.6, -0.05, -0.5). the vertex a1 instead will be a vertex with the same x, and the same y but different z as it is moved in depth (-0.8, -0.05, -0.35). After I calculate the vertices I used the already existing function quad to define the faces of the parallelepipeds. Each vertex has normal coordinates calculated by subtracting two pairs of vertices and calculate the cross product between them, in this way each faces of each parallelepiped has the same normal in the same point. For the texture coordinates go to question 7.

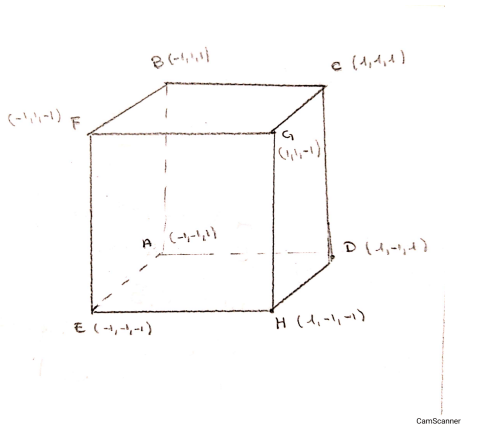

Figure 1: cube vertices

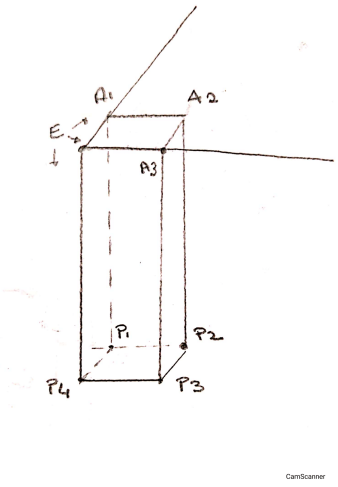

Figure 2: leg vertices

2. Choose an origin (different from the origin of the table) and include the rotation of the table around the origin and along all three axes

For this question I implement the rotation-matrix in the vertex-shader for each axis in the origin (0,0,0). To be able to compute a rotation about a fixed point I implement a translation matrix and its inverse centered in a vertex of my table (-0.6,-0.5,0.35) because i have to move the fixed point to the origin, rotate and then move the fixed point back. I also use in the vertex-shader a scale-matrix to visualize better the table in the canvas. My positions at the end becomes:

```
gl_Position = scale * traslation * rz *ry * rx
        * traslationInverse  * aPosition;
```

3. Add the viewer position, a perspective projection (and compute the ModelView and Projection matrices.

In Order to implement the viewer position i define a variable "eye", that is represented by three parameters : Radius (the distance from the origin), theta and phi (angles that allows you to change the camera view) and to manage these values i implement sliders. I relaize The model View Matrix whit the LookAt function where "at" is the position we are looking at and "up" is the orientation of the camera, meanwhile i realize the Projection Matrix whit four paramiters: "fovy", "aspect", "near" and "far". To manage this paramiters i used for the frist two sliders, and for the second two buttons. To apply this functionalities I also modify the glPosition in the vertex shader that becomes:

```
gl_Position = uProjectionMatrix  * uModelViewMatrix *
(scale * tras * rz *ry * rx  *trasI * aPosition );
```

4. Add a spotlight, place it outside of the view volume/ 5.Assign to the object a material with the relevant properties

For the material properties I follow the slide of the professor and I choose the following paramiters:

```
var materialAmbient = vec4(0.9, 0.9, 0.9, 1.0);
var materialDiffuse = vec4(0.9, 0.9, 0.9, 1.0);
var materialSpecular = vec4(1.0, 1.0, 1.0, 1.0);
```

To implement the spotlight I didn't implement the direction light and the point light but I have implement only the spotlight through the spotCosine function in the fragment shader, calculating the cutOff and the angle of the light . This two paramiters can be change through two buttons , and for the direction I use sliders.

```
float spotCosine =  max(dot(E , -L), 0.0);
        if (acos(spotCosine) > radians(spotLightAngle)){
            spotFactor = 0.0;
```

```
}else{
    spotFactor = pow(spotCosine,spotLightCutOff) * attenuation;
}
```

6. Implement both per-vertex and per-fragment shading models.

I realize per-vertex and per-fragment introducing a Boolean variable "changeShading" and I copy the spotlight functions in the vertex shader. I can switch them through a conditional instruction in both, vertex and fragment shader, but the color, the texture and the light buttons are managed for both in the fragment shader.

7. Find or create a wood texture and load it in the application.

I used the two-dimensional Texture mapping, so I define textCoords and I assign them to each vertex. Then i modify the vertex shader and the fragment shader to apply the texture and i used the ConfigureTexture function to load the image. I can enable and disable the texture through a button. The texture with chrome works only if I start a python server in the workspace and open in the browser the localhost:8000.

```
python -m http.server
```

8. Implement a motion blur effect

I have introduce the frame buffer and the render buffer together with an empty texture (as on the slide of the professor) but I didn't implement the blur effect.