Interactive Graphics
Homework 1

Andrea Panceri 1884749

1 Chair and normals

I delete the cube and replaced it with a chair of 40 vertices. The seat of the chair is formed by 8 vertices. Each leg of the chair is comprised of 7 vertices, with 4 vertices forming the bottom of the leg and 3 vertices forming the top, including the shared vertex with the bottom of the seat. The back of the chair is formed by 8 vertices, with 4 vertices shared with the upper parts of the back legs. Using exactly 40 vertices instead of 36, the chair has the legs attached perfectly to the corners of the bottom of the seat, that give a better rendering of the chair, with 36 some part don't render very well. To draw the chair, i use 6 parallelepipeds and for each one i used the function quad(a, b, c, d) six times. One parallelepipeds for each leg, one for the seat and one for the back. From the fact that i used the quad function for each face of the parallelepiped i used in total 192 point, that are used in combination with gl.TRIANGLES to draw the chair. To compute the normals for each face, i used the formula on the slides n = $(v_2 - v_0) \times (v_1 - v_0)$, where v_0, v_1, v_2, v_3 are the vertices of the corresponding face of the chair. The order of vertices is fundamental, they give the correct lightning on each face. I follow the right hand rule for inserting the vertices to have an outward normal. This is the implementation in JavaScript, i compute the normals in each face and add to normalsArray that will be linked to uniform matrix in the shaders:

```
function quad(a, b, c, d) {
    var t1 = subtract(vertices[b], vertices[a]);
    var t2 = subtract(vertices[c], vertices[b]);
    var normal = cross(t1, t2);
    normal = vec3(normal);

    var indices = [a, b, c, a, c, d];
    for (var i = 0; i < indices.length; ++i) {
        positionsArray.push(vertices[indices[i]]);
        normalsArray.push(normal);
    }
}
```

2 Origin and rotation

Following the instructions, i choose a new origin vec3(0.46,0.46,0.46) different from the origin of the chair vec3(0.0,0.0,0.0), and then i implement the rotation respect to this new origin. For change the origin of the rotation, we must move the updated origin to the original one, do a simple rotation in the origin, and then translate again the origin to the new one, like this: Rotation = Translate(O_{new}) * R(O)Translate * (-O_{new}), where R(O) = R_z * R_y * R_x. In the JavaScript i compute the rotation matrix respect to all three axes, in base of the direction of

rotation i increase or decrease theta_rotation. I added i button to toggle on/off the rotation, default is off, one button for each axis to choose the axis of rotation, a button to change direction of rotation and a slider to change the speed rotation. All handling of rotation are done in this function:

```
function setConfigurationOfRotation() {
    if (rotFlag) theta_rotation[axis] += rSpeed * rotDir;

    rotationMatrix = mat4();
    rotationMatrix = mult(rotationMatrix, rotate(theta_rotation[xAxis], vec3(1, 0, 0)));
    rotationMatrix = mult(rotationMatrix, rotate(theta_rotation[yAxis], vec3(0, 1, 0)));
    rotationMatrix = mult(rotationMatrix, rotate(theta_rotation[zAxis], vec3(0, 0, 1)));

    //translate origin of rotation from the origin
    rotationMatrix = mult(rotationMatrix, translate(0.46, 0.46, 0.46));
    rotationMatrix = mult(translate(-0.46, -0.46, -0.46), rotationMatrix)

    gl.uniformMatrix4fv(rotationMatrixLoc, false, flatten(rotationMatrix));
}
```

We can see that i use a flag 'rotFlag' that when is true, increment/decrement the angle of rotation on the actual axis of rotation, 'rSpeed' for change with the slider the speed, and 'rotDir' for change the direction, in the remaining lines there is the implementation of rotation matrix. We compute the rotation matrix in JavaScript to have less computation on GPU and use more CPU.

3 Viewer and projection

Following the example see at lesson, the camera is composed by three parameters: eye, at and up, that are given as inputs to the function 'LookAt(eye,at,up)' for create the modelViewMatrix. To move the camera we can use the slider for change phi, theta and radius. We use for the projection a perspective type, that i implement, following the examples of slides, with the given function 'perspective(fovy, aspect, near, far)' that take in input the parameters fovy, aspect, near and far that is possible to change with sliders, in particular we can change zNear and zFar to obtain that the chair is partly or completely outside of the View volume. The function implement for us the projectionMatrix. This function in JavaScript handle the setting of the camera:

```
function setConfigurationOfCamera() {
    eye = vec3(radius * Math.sin(theta) * Math.cos(phi),
        radius * Math.sin(theta) * Math.sin(phi), radius * Math.cos(theta));
    modelViewMatrix = lookAt(eye, at, up);
    projectionMatrix = perspective(fovy, aspect, near, far);

    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(modelViewMatrix));
    gl.uniformMatrix4fv(projectionMatrixLoc, false, flatten(projectionMatrix));
}
```

Initially the chair is visible, but dark because there in no light turned on. We compute the ModelView matrix in JavaScript to have less computation on GPU and use more CPU.

4 Spotlight

For obtain the final spotlight i used, like see at lesson, five vec4 that are position, direction, ambient, diffuse and specular.I setting three slider to change each coordinate of light direction, that are x, y and z, the same for the position one slider for coordinate. The range i choose is very short because i want that user can see effectively the change in direction. I put also two other variables, the 'lightAttenuation' and the 'lightValue', where the first allow to make the spotlight darker or have more strong light effect while the second allow to make angle of the spotlight wider or narrower. Light value must be changed for obtain the corresponding angle. There is also a button to turn on/off the spotlight, default is off, in fact is totally black at start the chair, there is only ambient light. Important is that the position of light is outside the viewing volume (z=-6.0), very far from the object, for this reason the angle is large when turn on the light. These are my settings:

```
var lightDirectionX = 0.0;
var lightDirectionY = -0.5;
var lightDirectionZ = 4.0;
var lightPositionX = 0.0;
var lightPositionY = 0.8;
var lightPositionZ = -6.0;
var lightPosition = vec4(lightPositionX, lightPositionY, lightPositionZ, 1.0);
var lightDirection = vec4(lightDirectionX, lightDirectionY, lightDirectionZ, 0.0);
var lightAmbient = vec4(0.3, 0.3, 0.3, 1.0);
var lightDiffuse = vec4(1.0, 1.0, 1.0, 1.0);
var lightSpecular = vec4(1.0, 1.0, 1.0, 1.0);
var lightAttenuation = 0.02;
var lightValue = 4.1;
```

For the attenuation i follow the formulas of the slides, and implement it like this, in the shaders:

```
float attenuationFactor = uLightAttenuation * pow(length((uLightPosition.xyz) - pos), 2.0);
float attenuation = 1.0 / (1.0 + attenuationFactor);

...

fColor = ambient + attenuation * (diffuse + specular);
```

The light is positioned in front of the chair and directed at the back of the chair, because i want to visualize better the chair,the various effect of the light and properties of the material. We compute all product in JavaScript of ambient, specular and diffuse matrices to have less computation in GPU.

5 Material

For the material i search for the most similar material to the wood, because the chair i build with different blocks want to emulate a wood chair. I choose from the possible default material the 'copper' that is defined by these three vectors and shininess:

```
var materialAmbient = vec4(0.19125, 0.0735, 0.0225, 1.0);
var materialDiffuse = vec4(0.7038, 0.27048, 0.0828, 1.0);
var materialSpecular = vec4(0.256777, 0.137622, 0.086014, 1.0);
var materialShininess = 0.1;
```

With this property i obtained a color near to brown, and like wood we have a low shininess.

6 Per-vertex and per-fragment shading model

I implement the spotlight in per-vertex and per-fragment shader added all needed variables and using the formulas on the slides, and combine the two result like this:

```
fColor = ((ambient + attenuation * (diffuse + specular)) * uPercentageFragment) + vColor * (1.0 - uPercentageFragment);
```

'ambient + attenuation * (diffuse + specular)' this is the result of per-fragment spotlight and we multiply for 'uPercentageFragment' that is the value take from the slider (0 to 100) that represent the percentage of per-fragment in the final result and combine with vColor the result of per-vertex then multiply for the '(1.0 - uPercentageFragment)'. I also find from the documentation of glsl that exist a function mix(v1,v2,p) that do the same thing. We can notice that when per-vertex percentage is near to 100% the light is more diffuse respect to per-fragment result, that using pixels is more precise.

7 Quantization

I implement quantization using eight colors that can be chosen by the user using colors picker, and this must be converted from hex to a vec4(r,g,b,a) using a simple function. The array of vec4 colors is linked to an uniform array of vec4 in fragment shader 'uniform vec4

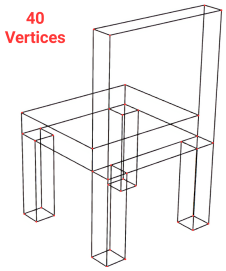
uChosenColors[8];'. For do the quantization i compute for each color the distance with the result of per-fragment 'fColor', and choose the color with minimum distance as result for each fragment.I used the formulas:
distance = $\sqrt{(color1.x - color2.x)^2 + (color1.y - color2.y)^2 + (color1.z - color2.z)^2}$
This is the code that i used, that is very modular and efficient:

```
float distanceColors(vec4 firstColor, vec4 secondColor) {
  float distanceX = firstColor.x-secondColor.x;
  float distanceY = firstColor.y-secondColor.y;
  float distanceZ = firstColor.z-secondColor.z;
  float distance = sqrt(distanceX*distanceX + distanceY*distanceY + distanceZ*distanceZ);
  return distance;

  ...

if(uIsQuantizationOn) {
    vec4 colorChosen = uChosenColors[0];
    float distance = distanceColors(fColor, colorChosen);
    float currDistance;

    for(int i = 1; i < 8; i++) {
      currDistance = distanceColors(fColor, uChosenColors[i]);
      if(currDistance < distance) {
        distance = currDistance;
        colorChosen = uChosenColors[i];
      }
    }

    fColor = colorChosen;
    }
}
```

Also i used a button that is linked to 'uIsQuantizationOn' that toggle the quantization effect, default is off.

8 Final comments

Want to outline that the code is very modular and reusable in the JavaScript file, also the css could be added in a separated file, but we can not add other files. Also want to notice that the page is responsive we can see the chair and adjust all settings in any kind of device.



(a) Light on

(b) Quantization on