SAPIENZA
UNIVERSITÀ DI ROMA

Artificial Intelligence and Robotics

Interactive Graphics

2020 May

Nijat Mursali | 1919669

HOMEWORK 1

Rome, Italy

Abstract & Introduction

This project put forward a method for solving the tasks and process of building a simple 3D interactive graphics project using WebGL, HTML and Javascript. However, for creating the user interface, we have also used Bootstrap and CSS (stands for Cascading Style Sheets) in order to make our project look good. We describe below the solutions that we have implemented in order to solve the tasks.

Development

Replace the cube with a more complex and irregular geometry of 20 to 30 vertices.

The idea here was to create a new object by replacing the cube element that we had in our homework. I have used Blender 3D which is free software to create the objects. Then, I created my own Python script in order to extract all the information such as vertices and faces to use in my project. The object I created (see in *Fig. 1),* has 24 vertices, 38 faces in overall. It contains 32 triangles and 6 squares. After extracting the vertices and faces, I have used my *forTriangle() and forQuads()* functions in order to combine the vertices. In order to combine the vertices in quad function, we are basically drawing two triangles. In order to calculate the normals we are defining two variables t_1 *and* t_2 where t_1 subtracts vertices *b to a* and t_2 subtracts vertices *c to b.* In order to get the normals, we needed to take the cross product of them and normalize the value before pushing it to our array element for normals. In order to get the texture coordinate, we have initialized the list with four elements/coordinates that start at (0, 0) and end at (1, 1).

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

My application takes the values of theta, phi angles and radius from the buttons and sliders of the HTML page. Then, those values are passed to the javascript applications, they are used to compute the ModelView matrix. We have initially declared an eye three dimensional vector where it takes the values of radius, theta, phi and multiplies them with the *sin and cos* of different variables. Then, we are getting the *modelView as* $modelView = lookat(eye, at, up);$ which we have declared them in the beginning of the JavaScript code as $at = vec3(0.0, 0.0, 0.0);$ $up = vec3(0.0, 1.0, 0.0);$. For the projection matrix, we have taken the values from the sliders and buttons of the parameters *fovy, near and far,* then put them on the perspective function.

Add two light sources, a spotlight in a finite position and one directional.

As mentioned in slides, we could just simply set an RGBA for the diffuse, specular and ambient components and also for the position. In order to create the lights in finite position we simply made $w = 1.0$. For the spotlight, we needed to take the direction and cutoff into account. We have also used the position and light properties as mentioned "Lighting and Shading in WebGL" in the slide #12 that says we need to choose the light properties as

$$var\ lightPosition\ =\ vec4\ (1.0,\ 1.0,\ 1.0,\ 0.0);$$
$$var\ lightAmbient\ =\ vec4\ (0.2,\ 0.2,\ 0.2,\ 1.0);$$
$$var\ lightDiffuse\ =\ vec4\ (1.0,\ 1.0,\ 1.0,\ 1.0);$$
$$var\ lightSpecular\ =\ vec4\ (1.0,\ 1.0,\ 1.0,\ 1.0);$$

Assign to the object a material with the relevant properties.

In order to add the lights, we have taken the values from the slides like the following properties as we have seen in our slides given in "Lighting and Shading II".As it is said in the slides, the material's properties should match the terms in the light light model. Thus, we have also chosen the properties as following:

$$var\ materialAmbient\ =\ vec4\ (1.0,\ 0.0,\ 1.0,\ 0.0);$$
$$var\ materialDiffuse\ =\ vec4\ (1.0,\ 0.8,\ 0.0,\ 1.0);$$
$$var\ materialSpecular\ =\ vec4\ (1.0,\ 0.8,\ 0.0,\ 1.0);$$
$$var\ materialShininess\ =\ 100.0;$$

Implement a per-fragment shading model based on the shading model described at the end of this document.

The shading model is a simplified version of the one presented by Lake, A., Marshall, C., Harris, M., and Blackstein, M. which consists of three parts. First of all, we needed to calculate the illuminated diffuse color for each material by the function $C_i\ =\ a_g\ x\ a_m\ +\ a_m + d_l\ x\ d_m$. We have chosen all the variables and added them in our shader to make the calculations and mentioned them in our JavaScript files. After calculating the C_i we have also calculated the C_s which is the shadowed diffuse color by the equation of $C_s\ =\ a_g\ x\ a_m + a_l\ x\ a_m$. After we calculate both C_i and C_s we have created the if statement to get the $Max(\vec{L} \cdot \vec{n},\ 0)$. Thus, in that if statement if the value is bigger or equal to 0.5 we assigned C_i, or C_w otherwise.

Add a texture loaded from file, with the pixel color a combination of the color computed using the lighting model and the texture.

We have chosen several textures for this project, but finally used the brick texture that is free to use and taken from Google. Firstly, we have created a function *configureTexture()* as shown in example, in order to configure the texture for our application. We have used the texture flag in order to set it in our *fragment shader* to enable/disable the texture if the button is clicked.

Results

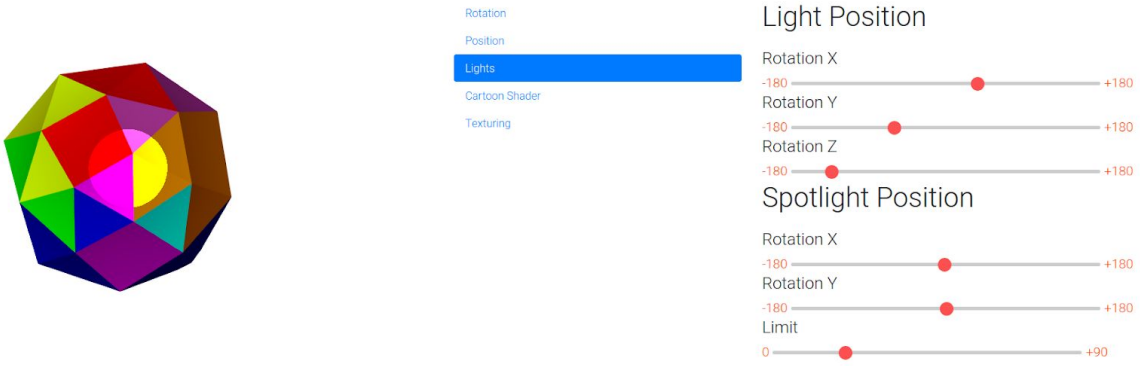

Fig. Object with Colors and Lights

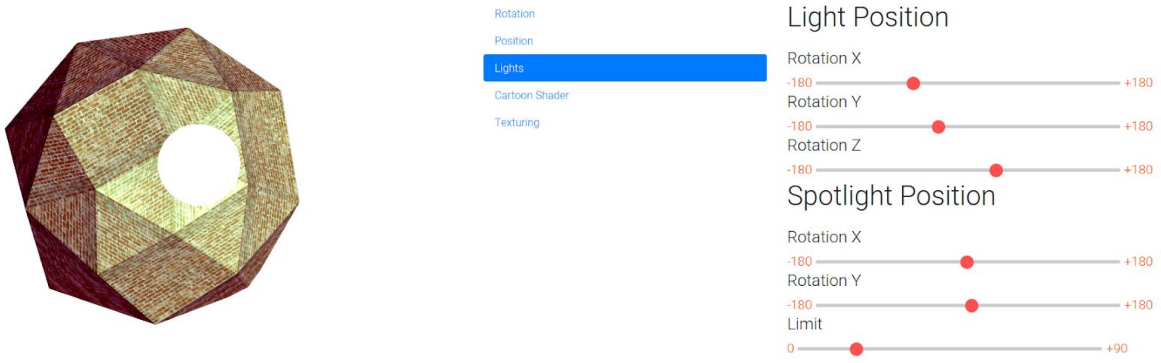

Fig. Object with Texture and Lights

Conclusion

The homework works successfully fine, but there are some limitations and bugs that I will be fixing for future release. As a next release, I will be adding more complex shapes that I have

created in Blender 3D and use the material we have learned to deploy a much better application.