Howework 1
Interactive Graphics

Ibis Prevedello

May 2018

1 Introduction

The goal of this homework is to use WebGL with control code written in JavaScript in order to draw an interactive 3D Cube in an HTML canvas in a web browser. The source code for this project can be accessed from my GitHub page.

2 Development

A series of improvements were done in the base code in order to achieve the following effects:

1. Add a button that changes the direction of the current rotation.
2. Move the transformations matrices from the shader to the Javascript application, so that the ModelView and Projection matrix are computed in the application and then transferred to the shader.
3. Include a scaling (uniform, all parameters have the same value) and a translation Matrix and control them with sliders.
4. Define an orthographic projection with the planes near and far controlled by sliders.
5. Define a perspective projection, introduce a button that switches between orthographic and perspective projection. The slider for near and far should work for both projections.
6. Introduce a light source, replace the colors by the properties of the material (your choice) and assign to each vertex a normal.
7. Implement both the Gouraud and the Phong shading models, with a button switching between them.

2.1 Item 1 - Change rotation direction

For the implementation of this item, it is necessary only to decrement the value of the angle instead of increment it, based on a variable chosen by the user.

```
if(direction)
  theta[axis] += 2.0;
else
  theta[axis] -= 2.0;}
```

2.2 Item 2 - Move the transformation matrices

In order to move the transformation matrices from the *shader* to the *JavaScript* application it was implemented multiplying the matrices as shown below.

```
modelViewMatrix = lookAt(eye, at, up);
modelViewMatrix = mult(modelViewMatrix,
    rotate(theta[xAxis], [1, 0, 0]));
modelViewMatrix = mult(modelViewMatrix,
    rotate(theta[yAxis], [0, 1, 0]));
modelViewMatrix = mult(modelViewMatrix,
    rotate(theta[zAxis], [0, 0, 1]));
gl.uniformMatrix4fv(gl.getUniformLocation(program,
    "modelViewMatrix"), false, flatten(modelViewMatrix));
```

2.3 Item 3 - Include scaling and translation

For the scaling and translation matrix it is possible to use the same functions in the *modelViewMatrix*.

```
modelViewMatrix = lookAt(eye, at, up);
modelViewMatrix = mult(modelViewMatrix,
    scalem(sliderScale, sliderScale, sliderScale));
modelViewMatrix = mult(modelViewMatrix,
    translate(sliderTX, sliderTY, sliderTZ));
modelViewMatrix = mult(modelViewMatrix,
    rotate(theta[xAxis], [1, 0, 0]));
modelViewMatrix = mult(modelViewMatrix,
    rotate(theta[yAxis], [0, 1, 0]));
modelViewMatrix = mult(modelViewMatrix,
    rotate(theta[zAxis], [0, 0, 1]));
gl.uniformMatrix4fv(gl.getUniformLocation(program,
    "modelViewMatrix"), false, flatten(modelViewMatrix));
```

2.4 Item 4 - Orthographic projection

For the orthographic projection it is possible to implement using the function *ortho*.

```
projectionMatrix = ortho(left, right, bottom, ytop, near, far);
gl.uniformMatrix4fv(gl.getUniformLocation(program,
    "projectionMatrix"), false, flatten(projectionMatrix));
```

2.5 Item 5 - Perspective projection

For this item it is also necessary to be possible to change the projection with a button. For this case the projection can be changed based on a flag and the projection changed between orthogonal and perspective.

```
if(perspec)
    projectionMatrix = perspective(fovy, aspect, near, far);
else
    projectionMatrix = ortho(left, right, bottom, ytop, near, far);
    gl.uniformMatrix4fv(gl.getUniformLocation(program,
        "projectionMatrix"), false, flatten(projectionMatrix));
```

2.6 Item 6 - Introducing the light source

For the light source, it is a little bit more complex because some more serious modification is necessary.

First, it is necessary to define light position, and ambient, diffuse and specular lights as well as material and shininess. This variables are passed to the *shader* and there they are used to create the light effect. Because this piece of code is a little more complex, it will not be showed here, but the full code can be found in my GitHub repository.

2.7 Item 7 - Gouraud and Phong shading models

For the Gouraud and Phong shading models, both are implemented in the *shader*, a button was added to make it possible to change from one model to another. Because the code is also a little bit complex, it can be verified in my GitHub repository.

One of the things that were not mentioned here is the process to create the vertex normals, for the normals it is possible to create it manually, what is not difficult for the case of a cube, however, for more complex models it becomes impossible. So, it can be added to the function *quad* and the normals are created automatically, as shown below.

```
var t1 = subtract(vertices[b], vertices[a]);
var t2 = subtract(vertices[c], vertices[b]);
var normal = cross(t1, t2);
var normal = vec3(normal);
normal = normalize(normal);
```

3 Conclusion

After implement all the features it was possible to see the evolution of the cube as more features were being added, reaching the final result of a real realistic cube, with perspective and lightning, compared with the base code.

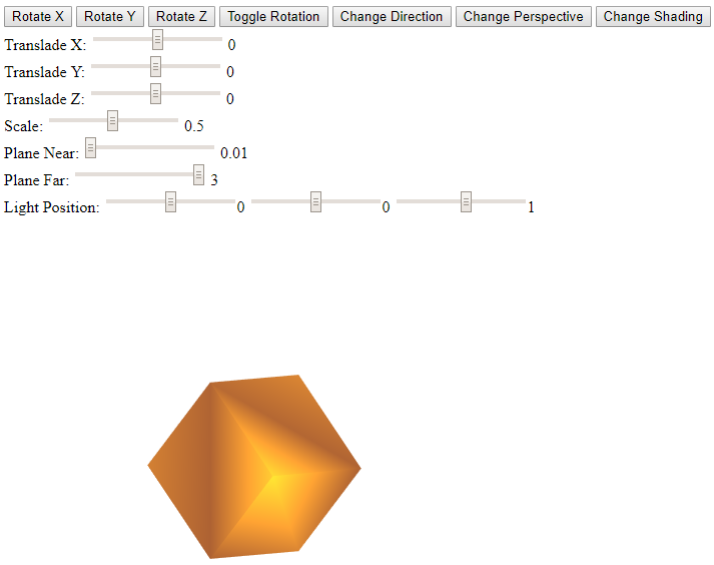

Figure 1: Cube 3D screenshot