WebGL Shaders

Authors

André Mourato 84745 Nuno Félix 80330

***Abstract* - This report describes the development and demonstration of the WebGL Shaders project chosen by our group for the Visual Computation subject.**

I. INTRODUCTION

The objective of this project was to learn and create shaders not used in the class. We created a program which lets you dynamically change the state of a scene with several objects on display, in which you can apply a set of predefined shaders and visualize the effects in real time.

II. DEMONSTRATION

As referred in the introduction, WebGL Shaders allows the user to dynamically change the position of a light source in the **Shadows** mode, change the texture and the fog of the objects dynamically in the **Textures and Fog** mode. User can move the camera freely in both modes.

For this demonstration we used a predefined scene 'Room Scene' in which the objects were from by files.

A. Shadow Mapping and Camera View

In **Shadows** mode you can observe phong lightning locally on each object surface and shadows of the objects dynamically changing as the light source moves across the room.

Fig. 1 - Shadow Mapping

You can move the camera with **w**, **a**, **s**, **d** keys for the XX axis and the YY axis.
For left and right rotation use **q**, **e** keys, respectively.
You can also move the light bulb position with ↑↓→← keys, in order to observe the shadows changing dynamically.

Fig. 2 - Shadow Map after moving light bulb

B. Textures and Fog

In **Textures and Fog** mode you can use dropdowns to select which object you want to change and apply a new texture for each object, from the list of available textures. A new texture can be added by placing an image in the folder src/assets/textures. Once the page is refreshed the texture will be available.

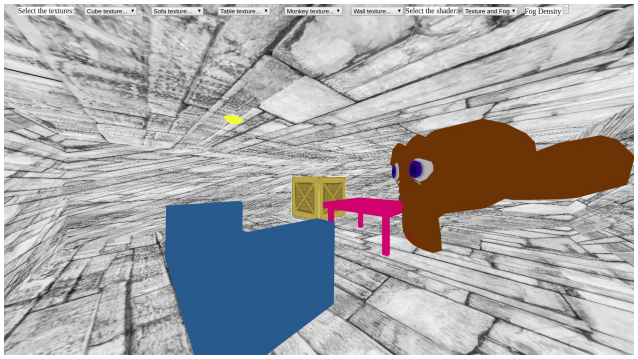


Fig. 3 - Texture and Fog mode

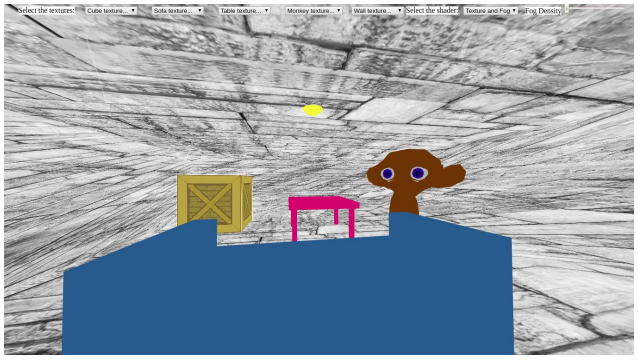


Fig.4 - Changing camera view

Fig. 5 - Changed object textures

Fig. 5 - Fog setting 0%

You can select the fog setting by using a range input

Fig. 6 - Fog setting 70%

III. DEVELOPMENT

A. Loading 3D Models

In order to read existing 3D models, we setup an HTTP NodeJS server (whose source code can be found in folder src/server/index.js) that loads the models from files (.obj, .json and .txt). These files contain information about the vertices, faces, indices, texture coordinates and normals of a given model (Cube, MonkeyMesh, WallMesh,etc.). This information is requested by the client that fetches the results from the server endpoints `/models_obj` ,`/models_json and /models.`

The server not only loads the 3D models of objects but also loads their position from the file at src/assets/scenes/room.txt, available at endpoint `/scenes.`

Note that **there is a clear difference between models and objects**, since multiple objects can be of the same model (examples: cube1, cube2 are of model Cube defined in Room.json)

B. Setup a scene

We can change the state of the scene by changing the parameters of file /assets/scenes/room.txt with the following layout:

```
0.9 0 3.1 0 0 0 0.2 0.2 0.2 LightBulbMesh
Yellow 255 255 255
1.57 -0.7 0.49 0 0 0 1 1 1 TableMesh Pink
-3.28 0 0.77 0 1 0 1 1 1 SofaMesh Blue1
2.07 -0.98 1.75 0 0 -1 0.1 0.1 0.1 Suzanne
MonkeyTexture
0 0 0 0 0 0 1 1 1 WallsMesh WhiteWall
0 0 0 0 0 0 1 1 1 Cube Crate
```

This file 'room.txt' setups the declared objects 'Room Scene'

```
0.9 0 3.1 0 0 0 0.2 0.2 0.2 LightBulbMesh
Yellow
```

This line defines the attributes of an object in a Scene. These attributes are respectively: Position X, Position Y, Position Z, Angle X (degrees), Angle Y (degrees), Angle Z (degrees), Scale X, Scale Y, Scale Z, Model name and Texture Name, R in shadow mode (optional), G in shadow mode and B value in shadow mode

C. Loading Textures

We can load textures to the project simply by adding an image to the folder /src/assets/textures. The image will be available to load the next time the page is refreshed (since the server fetches the information before rendering the objects)

A. Loading Shaders

We use three shader programs:

- noShadowProgram - for applying textures and computing the fog
- shadowProgram - for objects surface phong lightning
- shadowMapProgram - for calculating objects shadows

These shaders are initiated in the runWebGL() function and are later used when drawing a scene in the respective draw functions of each shader (drawSceneTextures(),etc.)

A. Camera

The camera is defined by 3 main properties: the position in the world (x,y,z coordinates), the point where it's "looking at" and the vector that defines the upwards direction.

The forward pointing vector is calculated by subtracting the position where the camera is initially looking at and subtracting to the current position of the camera.

```
vec3.subtract(this.forward, lookAt,
this.position);
```

Once we have calculated the forward pointing vector it is trivial to calculate the right pointing vector since the cross product between all 3 vectors has to be 0. Note that it was unnecessary to calculate the up vector since it was already given as an argument in the beginning, but we will do it anyways.

```
vec3.cross(this.right, this.forward, up);
vec3.cross(this.up, this.right,
this.forward);
```

And then we normalize in order to get the unit vector of the 3 vectors (forward, up, right):

```
vec3.normalize(this.forward, this.forward);
vec3.normalize(this.right, this.right);
vec3.normalize(this.up, this.up);
```

In order for the user to move forward we must add the forward unit vector multiplied by a constant (the speed) to the current position vector (x,y,z)

```
vec3.scaleAndAdd(this.position, this.position,
this.forward, speed);
```

This process is the same for the remaining directions

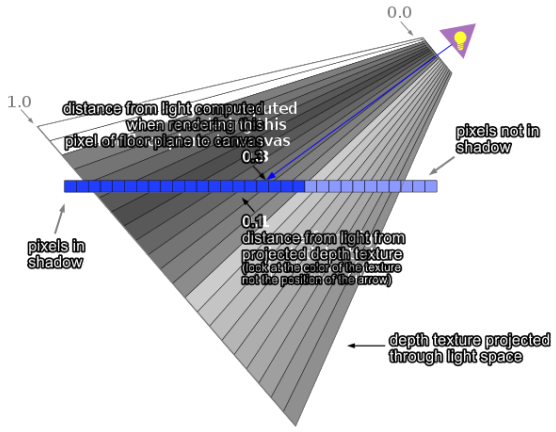
In order for the user to be able to roam freely in the scene, the only thing that remains is to update the ViewMatrix:

```
camera.GetViewMatrix(viewMatrix)
```

A. Shadow Mapping

In order to explain shadow mapping we will start by explaining the concept of a shadow itself. A shadow can be observed when an opaque object blocks the light originating from a light source. This results in a darker section in the surface that is positioned behind the object. Therefore we need to determine which object is in front. The way we do this is by calculating, for each pixel, if the object to whom the pixel belongs is the closest to the Light Source. If it is, we know that that pixel is in the light and every pixel from the other objects in that position are in the shadow. This calculation can be performed by the following shader:

```
vec3 toLightNormal =
normalize(pointLightPosition - fPos);
float fromLightToFrag = (length(fPos -
pointLightPosition) - shadowClipNearFar.x) /
(shadowClipNearFar.y - shadowClipNearFar.x);
float shadowMapValue =
textureCube(lightShadowMap, -toLightNormal).r;
```



In order to calculate the shadows all around the light source we instantiate 6 new "cameras": 2 for the XX axis (the positive and the negative of the object), 2 for the YY axis, 2 for the ZZ axis.

```
shadowMapCameras = [ // +X new Camera(
currentSceneLightPosition,
vec3.add(vec3.create(),
currentSceneLightPosition, vec3.fromValues(1,
0, 0)), vec3.fromValues(0, -1, 0) ),
 // -X new Camera( currentSceneLightPosition,
vec3.add(vec3.create(),
currentSceneLightPosition, vec3.fromValues(-1,
0, 0)), vec3.fromValues(0, -1, 0) ),
 // +Y new Camera(...), // -Y new Camera(...),
 // +Z new Camera(...), // -Z new Camera(...),
];
```

Each camera will use the shader explained above to determine which pixels are in the shadow and which are not.

A. Textures

The textures are loaded by the loadTextures() function that can be found in 'src/assets/js/index.js'. First this function calls fetchTextures() function that makes a request to the Node.js server and parses all .jpeg files in 'src/assets/textures/', secondly for each texture found in the folder loads asynchronously the image.

The following code is used in loadTextures() to create a Texture and copy the image loaded to the created texture.

```
var texture = gl.createTexture()

var textureName =
textures_available[id].name var texture =
textures_available[id].texture

gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL,
true) gl.bindTexture(gl.TEXTURE_2D,
texture); gl.texParameteri(gl.TEXTURE_2D,
gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
gl.texParameteri(gl.TEXTURE_2D,
gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
gl.texParameteri(gl.TEXTURE_2D,
gl.TEXTURE_MIN_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_2D,
gl.TEXTURE_MAG_FILTER, gl.LINEAR);
gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGBA,
gl.RGBA, gl.UNSIGNED_BYTE,
textures_available[id].image );
gl.bindTexture(gl.TEXTURE_2D, null);
```

When **Textures and Fog** mode is selected drawSceneTextures() is invoked and the projection matrix is updated as shown in the code below.

```
gl.clear(gl.COLOR_BUFFER_BIT |
gl.DEPTH_BUFFER_BIT);
gl.useProgram(noShadowProgram)
gl.uniformMatrix4fv(noShadowProgram.uniform
s.mProj, gl.FALSE, projMatrix);
```

For each object in the scene this excerpt will be executed and the object position in the world view is updated

```
obj.updatePosition()
gl.uniformMatrix4fv(
    noShadowProgram.uniforms.mWorld,
    gl.FALSE,
    obj.world);
refreshTexture(obj)
gl.bindBuffer(gl.ARRAY_BUFFER, null);
generateSceneTextures(obj)
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER,
obj.ibo);
```

In generateSceneTextures() we associate Vertices, Texture, Indices to the vertex Shader and then we draw the elements.

```
gl.drawElements(gl.TRIANGLES,
    obj.indices.length,
    gl.UNSIGNED_SHORT,
    0);
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER,
null);
}
```

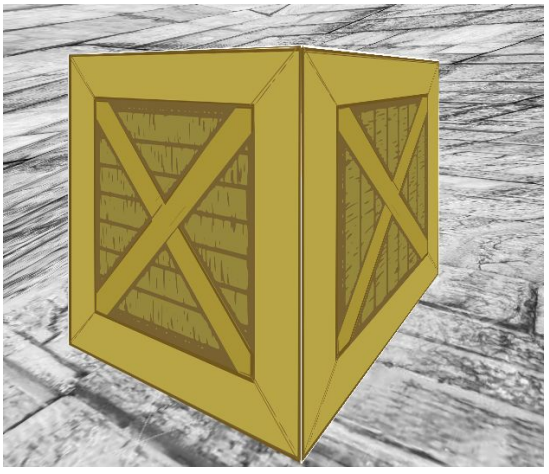


Fig. 5 - Example of a Cube with 'Crate' texture

A. Fog

Fog is computed in the same shader program as the textures by the following formula:

$$gl\_fragColor = originalColor + (fogColor - originalColor) * fogAmount$$

where originalColor is the texel, color value of the texture for the given coordinates and fogAmount is a value between 0 and 1, when fogAmount is 0 returns originalColor if 1 returns fogColor. In between 0 and 1 you get a percentage of both colors.

When executing drawSceneTextures() we run this piece of code that allows us to set the fogColor and the fogAmount

```
gl.uniform4fv(noShadowProgram.uniforms.fogC
olorLocation, fogColor);
gl.uniform1f(noShadowProgram.uniforms.fogAm
ountLocation, fogAmount);
```

A. Deployment

Throughout the development of WebGL Shaders the group used Git as a version control system. More information can be found in the repository of the project: https://github.com/andremourato/ShadersWebGL

To deploy our project you need to install some prerequisites (NodeJS and NPM):

```
sudo npm cache clean -f
sudo npm install -g n
sudo n 12.13.0
```

Installing dependencies

```
npm install
```

Running Server

```
cd src
npm run server
```

For running the client you need to open src/index.html file with the browser.

IMPORTANT NOTE

This project was developed using the Linux Ubuntu operating system and Google Chrome, strangely we encountered some problems while rendering the **Texture and Fog** mode in macOS, for some reason that we couldn't find or fix, some objects are not rendered correctly.

REFERENCES

Use the Style "referencia" to the references. Example:

[1] https://webglfundamentals.org/webgl/lessons/webgl-shadows.html
[2] https://webglfundamentals.org/webgl/lessons/webgl-3d-textures.html
[3] http://juliorlunar.com/lighting1/
[4]http://sweet.ua.pt/jmadeira/CV/CV_08_Illumination_Shading_BSS_JM.pdf
[5]https://webglfundamentals.org/webgl/lessons/webgl-3d-camera.html