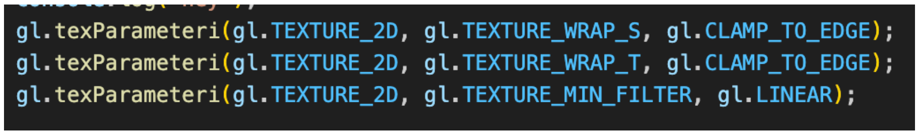
CS405 Project 2 Report **Barış Ulaş Çukur 29461**

Task 1

Handling the non-power of 2 textures required passing *TEXTURE_2D, TEXTURE_WRAP_S, TEXTURE_WRAP_T, CLAMP_TO_EDGE, TEXTURE_MIN_FILTER* and *LINEAR* parameters to OpenGL. These parameters were enough to make the non-power-of-two textures wrap to the object.



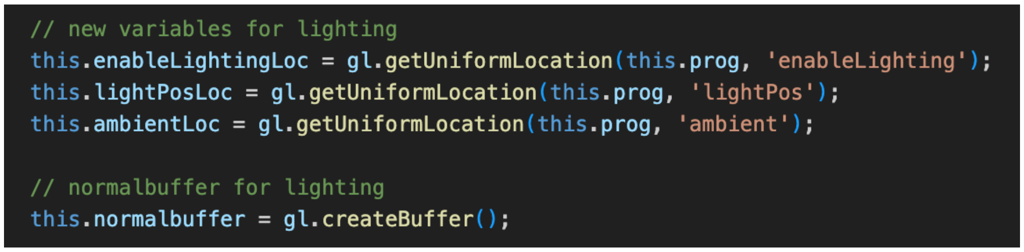
```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
```

Figure 1.1: OpenGL parameters for non-power-of-two textures.

Task 2

Updating MeshDrawer Constructor

I've added locations for *enableLightning normalCoord*, *lightPos* and *ambient* variables. I've also created and initialized *normalBuffer* here to store normals of the diffusion light.

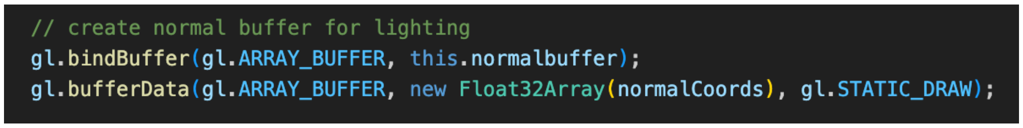


```
// new variables for lighting
this.enableLightingLoc = gl.getUniformLocation(this.prog, 'enableLighting');
this.lightPosLoc = gl.getUniformLocation(this.prog, 'lightPos');
this.ambientLoc = gl.getUniformLocation(this.prog, 'ambient');

// normalbuffer for lighting
this.normalbuffer = gl.createBuffer();
```

Figure 1.2

Updating setMesh Function

Similarly, I've binded *normalbuffer* from the constructor here to *GL_ARRAYBUFFER.* I've also transferred the contents of normal coordinates to this buffer using *gl.bufferData* function.

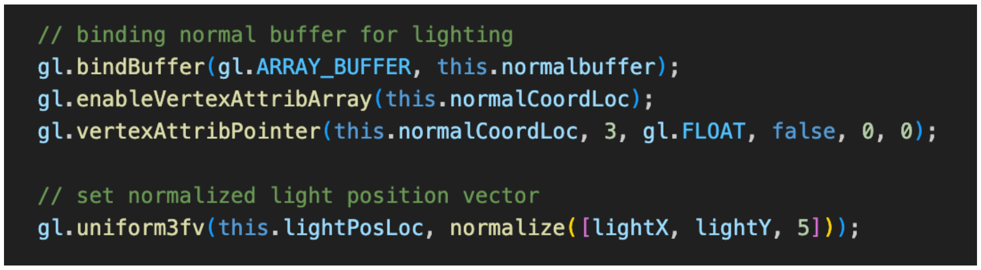


```
// create normal buffer for lighting
gl.bindBuffer(gl.ARRAY_BUFFER, this.normalbuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(normalCoords), gl.STATIC_DRAW);
```

Figure 1.3

Updating draw Function

Here I binded the normal buffer to a new *gl.ARRAY_BUFFER* and used *gl.enableVertexAttribArray* to enable attribute location for the vertex shader. In *gl.vertexAttribPointer*, I've also made necessary changes to the parameters according to the structure of the data. Finally, I've updated the light position using *gl.uniform3fv.*



```
// binding normal buffer for lighting
gl.bindBuffer(gl.ARRAY_BUFFER, this.normalbuffer);
gl.enableVertexAttribArray(this.normalCoordLoc);
gl.vertexAttribPointer(this.normalCoordLoc, 3, gl.FLOAT, false, 0, 0);

// set normalized light position vector
gl.uniform3fv(this.lightPosLoc, normalize([lightX, lightY, 5]));
```

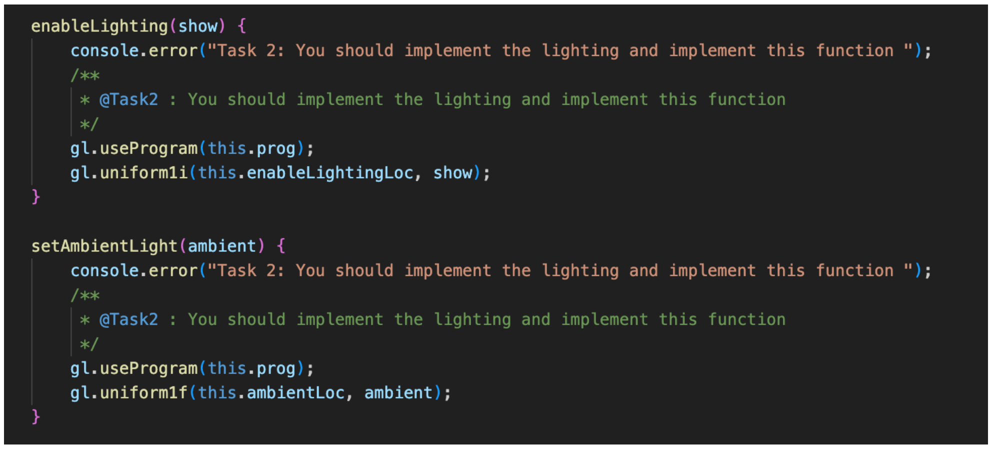
Figure 1.4

Updating enableLightning Function

I've called *gl.useProgram* to specify the currently active shader program, and then called *gl.uniform1i* to change the value of *enableLighting* (inside the shader not the constructor) to the value of the toggle itself (Figure 1.5).

Updating setAmbientLight Function

I've pretty much did the exact same thing here as the *enableLighting* function. Got the current active shader program and changed the value of *ambientLight* to the state of the toggle (Figure 1.5).

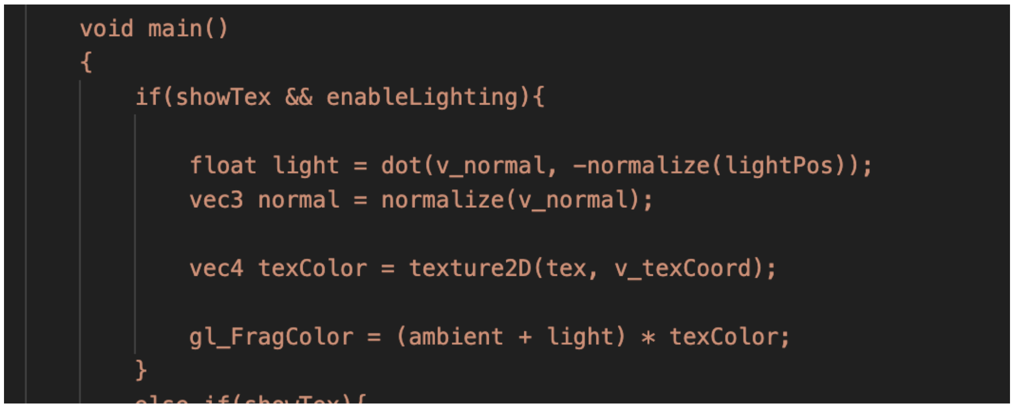


```
enableLighting(show) {
    console.error("Task 2: You should implement the lighting and implement this function ");
    /**
     * @Task2 : You should implement the lighting and implement this function
     */
    gl.useProgram(this.prog);
    gl.uniform1i(this.enableLightingLoc, show);
}

setAmbientLight(ambient) {
    console.error("Task 2: You should implement the lighting and implement this function ");
    /**
     * @Task2 : You should implement the lighting and implement this function
     */
    gl.useProgram(this.prog);
    gl.uniform1f(this.ambientLoc, ambient);
}
```

Figure 1.5

Updating Fragment Shader

First, I obtained the normal vector by using the *normalize* function given in the js file. I then calculated the diffuse light between the normal vector and the negated normal light direction "lightPos". Later, I've used texture2D function to get the texture colors of the current position of the object, which is called *texColor*. Finally, I've combined them together by multiplying the *texColor* with the sum of light and ambient (Figure 1.6).

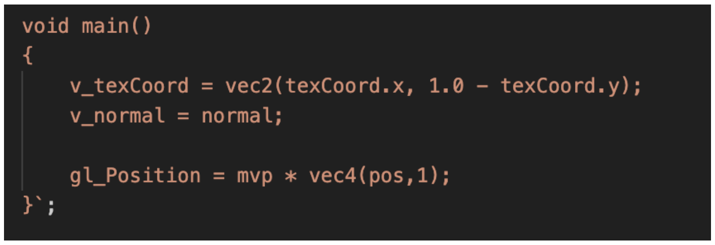


```
void main()
{
    if(showTex && enableLighting){

        float light = dot(v_normal, -normalize(lightPos));
        vec3 normal = normalize(v_normal);

        vec4 texColor = texture2D(tex, v_texCoord);

        gl_FragColor = (ambient + light) * texColor;
    }
    else if(showTex){
```

Figure 1.6

Extra Updates

Upon completion of the first task, I've realized that the non-power-of-two image I upload gets flipped 180 degrees when applied to the object as a texture. To fix this issue, I've updated *v_texCoord* in vertex shader to flip the texture upside down before applying it to the image (Figure 1.7).



```
void main()
{
    v_texCoord = vec2(texCoord.x, 1.0 - texCoord.y);
    v_normal = normal;

    gl_Position = mvp * vec4(pos,1);
}`;
```

Figure 1.7