Ahmet Burak Kurtulmuş – 28996

CS 405 Project 2 Report

Task 1:

I began the project by working on task 1. This task involved modifying the `setTexture` method in the `project2.js` file. The existing implementation only accepted images with dimensions that are powers of two. The necessary modification was to enable the use of images of any size as textures. I initiated the changes by altering the `else` part of the `setTexture` method.

Similar to Recitation 7, I modified the code using methods such as `gl.CLAMP_TO_EDGE` and `gl.LINEAR`. If the dimensions of the image are not powers of two, the code now adjusts texture parameters to accommodate textures of non-power-of-2 sizes.

I utilized an image named `leaves.jpg` under the resources directory to test the implementation. The following is a screenshot of the model:

Task 2:

I started with the constructor. Firstly, i have initialized this.normalLoc and this.normalbuffer to handle normals, which are crucial for lighting calculations. Second, this.enableLightingLoc, this.lightPosLoc, and this.ambientLoc represent uniform locations in the shader program. They are utilized to transmit lighting parameters (such as whether lighting is enabled, light position, and ambient light density) from the JavaScript code to the WebGL shaders.

```
class MeshDrawer {
    // The constructor is a good place for taking care of the necessary initializations.
    constructor() {
        this.prog = InitShaderProgram(meshVS, meshFS);
        this.mvpLoc = gl.getUniformLocation(this.prog, 'mvp');
        this.showTexLoc = gl.getUniformLocation(this.prog, 'showTex');

        this.colorLoc = gl.getUniformLocation(this.prog, 'color');

        this.vertPosLoc = gl.getAttribLocation(this.prog, 'pos');
        this.texCoordLoc = gl.getAttribLocation(this.prog, 'texCoord');


        this.vertbuffer = gl.createBuffer();
        this.texbuffer = gl.createBuffer();

        this.numTriangles = 0;

        this.normalLoc = gl.getAttribLocation(this.prog, 'normal');
        this.normalbuffer = gl.createBuffer();
        this.lightingEnabledLoc = gl.getUniformLocation(this.prog, 'enableLighting');
        this.lightPosLoc = gl.getUniformLocation(this.prog, 'lightPos');
        this.ambientLoc = gl.getUniformLocation(this.prog, 'ambient');
    }
```

Secondly, there were tasks related to binding and setting the vertices buffer. Similar to Recitation 7, I bound and set the vertices buffer in the setMesh method.

```
setMesh(vertPos, texCoords, normalCoords) {
    gl.bindBuffer(gl.ARRAY_BUFFER, this.vertbuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertPos), gl.STATIC_DRAW);

    // update texture coordinates
    gl.bindBuffer(gl.ARRAY_BUFFER, this.texbuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(texCoords), gl.STATIC_DRAW);

    this.numTriangles = vertPos.length / 3;

    gl.bindBuffer(gl.ARRAY_BUFFER, this.normalbuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(normalCoords), gl.STATIC_DRAW);

}
```

Thirdly, in this step, I activated the normal buffer to be able to use normal vectors. Additionally, following the approach we took in Recitation 7, I updated relevant uniform values such as light position, and ambient density. Firstly, I set whether the light is on or off. Then, I determined the light intensity and position, assigning a value of 1.0 to this variable: gl.uniform3f(this.lightPosLoc, lightX, lightY, 1.0).

```
draw(trans) {
    gl.useProgram(this.prog);

    gl.uniformMatrix4fv(this.mvpLoc, false, trans);

    gl.bindBuffer(gl.ARRAY_BUFFER, this.vertbuffer);
    gl.enableVertexAttribArray(this.vertPosLoc);
    gl.vertexAttribPointer(this.vertPosLoc, 3, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, this.texbuffer);
    gl.enableVertexAttribArray(this.texCoordLoc);
    gl.vertexAttribPointer(this.texCoordLoc, 2, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, this.normalbuffer);
    gl.enableVertexAttribArray(this.normalLoc);
    gl.vertexAttribPointer(this.normalLoc, 3, gl.FLOAT, false, 0, 0);

    gl.uniform1i(this.lightingEnabledLoc, true);
    gl.uniform1f(this.ambientLoc, 0.5);
    gl.uniform3f(this.lightPosLoc, lightX, lightY, 1.0);


    updateLightPos();
    gl.drawArrays(gl.TRIANGLES, 0, this.numTriangles);
```

I updated the methods to enable/disable lighting (`enableLighting`) and set ambient light intensity (`setAmbientLight`).

```
enableLighting(show) {
    var lightingEnabled = true;
    gl.uniform1i(this.lightingEnabledLoc, lightingEnabled);
}

setAmbientLight(ambient) {
    gl.uniform1f(this.ambientLoc, ambient);
}
```

I rearranged this fragment shader to include lighting calculations. Prior to the second task, it was directly applying the texture. With Task 2, based on the control of the `enableLighting` variable, if lighting is enabled, diffuse and ambient light calculations are performed. The coloring process is then carried out using these calculated values. Values like `diff` and `ambientColor` are added to `gl_FragColor`, resulting in the colored output.

```
void main()
{
    if(showTex && enableLighting){
        vec3 lightDir = normalize(lightPos - v_normal);
        float diff = max(dot(v_normal, lightDir), 0.0);
        vec3 diffuse = diff * texture2D(tex, v_texCoord).xyz;
        vec3 ambientColor = ambient * texture2D(tex, v_texCoord).xyz;
        gl_FragColor = vec4(diffuse + ambientColor, 1.0);
    }
    else if(showTex){
        gl_FragColor = texture2D(tex, v_texCoord);
    }
    else{
        gl_FragColor =  vec4(1.0, 0, 0, 1.0);
    }
```

As a result, the following changes were made in relation to Task 2:

constructor (MeshDrawer): Added necessary lighting-related variables and locations.

setMesh (MeshDrawer): Added a normal buffer and updated the setMesh function to handle normal vectors.

draw (MeshDrawer): Enabled the normal buffer and set lighting uniforms for ambient light intensity and light position.

enableLighting (MeshDrawer): Implemented the enableLighting function to toggle lighting on/off.

setAmbientLight (MeshDrawer): Implemented the setAmbientLight function to set the ambient light intensity.

Fragment Shader (meshFS): Updated the fragment shader to handle lighting when both texture and lighting are enabled.

As a result, the model looks like this with the light:

Without Light: