Emre Zeytinoğlu 28190

**CS405 PROJECT 2**

Task 1 involves improving the setTexture function in the WebGL implementation to smoothly support non-power-of-2 sized textures. This job is significant because it addresses a major limitation in graphics programming in which mipmapping, which is typically used for smoother texture rendering, historically needs textures to have dimensions that are powers of 2 (e.g., 2x2, 4x4, 8x8).

The code-implemented technique was designed to handle textures with dimensions that do not meet the power-of-2 restriction. This is accomplished by modifying the texture parameters to correspond the dimensions of the loaded image. If the image's width and height are not powers of 2, the code sets the texture wrapping mode to CLAMP\_TO\_EDGE. This modification displays the code's adaptation to a greater range of textures, contributing to the WebGL application's adaptability.

A screenshot of a computer

Description automatically generated

For second task, I implemented some code snippets into MeshDrawer class, enableLightning function, setAmbientLightning function, meshFS class as it stated in the assignment pdf. You can see implementations and explanations below:

**1-** This section shows the necessary variables for lighting effects that I created in the MeshDrawer class.

/\*\*

\* @Task2 : You should initialize the required variables for lighting here

\*/

this.enableLightingLoc = gl.getUniformLocation(this.prog, 'enableLighting');

this.ambientLoc = gl.getUniformLocation(this.prog, 'ambient');

this.normalbuffer = gl.createBuffer();

this.normalLoc = gl.getAttribLocation(this.prog, 'normal');

this.lightPosLoc = gl.getUniformLocation(this.prog, 'lightPos');

this.lightPos = [0.0, 0.0, 1.0];

This initialization section creates the necessary structure for introducing lighting effects to the WebGL application. It starts connections between JavaScript variables and WebGL shader variables.

**2-** This code snippet appears to be part of the rendering process that I created in the draw method of the MeshDrawer class.

gl.bindBuffer(gl.ARRAY\_BUFFER, this.normalbuffer);

gl.enableVertexAttribArray(this.normalLoc);

gl.vertexAttribPointer(this.normalLoc, 3, gl.FLOAT, false, 0, 0);

if (lightX > 20) {

lightX = 20;

} else if (lightX < -20) {

lightX = -20;

}

if (lightY > 20) {

lightY = 20;

} else if (lightY < -20) {

lightY = -20;

}

gl.uniform3fv(this.lightPosLoc, [lightX, lightY, 0]);

updateLightPos();

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

This section of code prepares the normal vectors for vertex shading, checks that the light position is inside the constraints provided, changes the required WebGL uniforms, and renders the mesh using WebGL's drawArrays function. This step is critical for introducing lighting effects into the 3D scene's visual representation.

**3-** The enableLighting method in the provided code is responsible for checking the lighting effects on or off.

enableLighting(show) {

gl.useProgram(this.prog);

gl.uniform1i(this.enableLightingLoc, show);

}

The enableLighting method provides a convenient way to turn lighting effects on or off by updating the corresponding boolean uniform in the shader program.

**4-** The setAmbientLight method in the provided code is responsible for setting the ambient light intensity in the shader program.

setAmbientLight(ambient) {

gl.useProgram(this.prog);

gl.uniform1f(this.ambientLoc, ambient);

}

The shader program's ambient light intensity can be dynamically modified with the help of the setAmbientLight method. This helps to control the scene's baseline lighting, which enhances the overall lighting effects given to the produced model.

**5-**The provided fragment shader (meshFS) is responsible for determining the color of each fragment (pixel) in the rendered mesh.

void main()

{

if(showTex && enableLighting){

// UPDATE THIS PART TO HANDLE LIGHTING

vec3 ambientLight = ambient \* vec3(texture2D(tex, v\_texCoord));

vec3 norm = normalize(v\_normal);

vec3 lightDir = normalize(lightPos - fragPos);

float diff = max(dot(norm, lightDir), 0.0);

vec3 diffuseLight = diff \* vec3(1);

vec3 lighting = ambientLight + diffuseLight;

gl\_FragColor = vec4(lighting, 1.0) \* texture2D(tex, v\_texCoord);

}

else if(showTex){

gl\_FragColor = texture2D(tex, v\_texCoord);

}

else{

gl\_FragColor = vec4(1.0, 0, 0, 1.0);

}

}

Conditional blocks are included in the shader to handle scenarios in which texture and lighting are either enabled or disabled. The fragment is given a default color if both lighting and texturing are turned off. Because of this shader's adaptability, the rendering pipeline can change its behavior depending on whether texture and lighting effects are present or not.

