CS 405 Project 2: Textures + Illumination Implementation

Introduction

In this report, I outline the key features and methodologies I employed in the implementation of Project 2 for the CS 405 course. The project focused on enhancing a WebGL-based 3D rendering application with advanced texture handling and basic lighting models.

Methodology

Task 1: Texture Handling

The first task required me to modify the `setTexture` method in `project2.js` to support textures with non-power-of-two dimensions. Originally, the application could only handle textures where both width and height were powers of two. My modifications included:

- Implementing `gl.CLAMP_TO_EDGE` for both `gl.TEXTURE_WRAP_S` and `gl.TEXTURE_WRAP_T`, preventing the texture from repeating.
- Setting `gl.LINEAR` for both `gl.TEXTURE_MIN_FILTER` and `gl.TEXTURE_MAG_FILTER` to ensure smooth transitions between texture pixels.

These enhancements allow for greater flexibility in the types of textures the application can handle, accommodating a wider variety of image sizes.

Task 2: Basic Lighting

For the second task, I was responsible for implementing basic lighting in the application, including ambient and diffuse light. During diligently working on this step for days, I managed to implement an ambient light effect. While the result slightly deviates from the exact specifications outlined in the homework document, it represents a significant accomplishment in the context of my learning and application development process. This step required several changes:

Constructor and Uniform Variables

In the `MeshDrawer` class constructor, I initialized new properties and uniform locations:

- I defined `this.lightPosition`, `this.lightColor`, and `this.ambientIntensity` to control the light's position, color, and ambient intensity.
- I retrieved uniform locations for these properties (`this.lightPosLoc`, `this.lightColorLoc`, `this.ambientIntensityLoc`, and `this.enableLightingLoc`) using `gl.getUniformLocation`.

Setting Mesh and Lighting Data

In the `setMesh` method, I created and bound a buffer for normal vectors (`this.normbuffer`) to pass normal data to the shaders, essential for calculating lighting effects based on the surface orientation.

Drawing with Lighting
In the `draw` method, I set the uniform variables for lighting using `gl.uniform` calls. This included dynamically updating the light position with `updateLightPos`.

Fragment Shader
I modified the fragment shader (`meshFS`) to include lighting calculations:

- I computed the ambient component as a minimum ambient value to ensure no part of the mesh was completely dark.
- I calculated the diffuse component using the dot product of the normalized normal vector and light direction.
- I determined the final color by combining the texture color with both ambient and diffuse lighting components.

Ambient Light Adjustment
I implemented the `setAmbientLight` method to dynamically adjust the ambient light intensity.

Light Position Update
I added a function `updateLightPos` to enable dynamic adjustment of the light position using arrow keys, affecting the lighting effect on the mesh.

Conclusion
Task 1 enhances the application's flexibility in handling a variety of textures, while Task 2 introduces basic lighting that significantly improves the visual quality of the rendered scene.