**CS405-Project 2 Report**

**Arda Güney - 28997**

**Task -1:**

Improving the **setTexture** method to handle textures with non-power-of-2 dimensions was the main goal of Task 1. .I only changed the else block of the function as indicated in the Java I added settings to deal with problems arising from textures that are not powers of two. First, **CLAMP\_TO\_EDGE** was selected as the texture wrapping option for the vertical (T) and horizontal (S) axes. By selecting this option, undesirable wrapping artifacts are avoided since texture coordinates that fall outside of the typical [0, 1] range are clamped to the texture's boundaries. Furthermore, **LINEAR** configuration was used for the magnification and minification filters. By this I got a smooth interpolation between texels.

**Task-2:**

To make it easier to include lighting capabilities in Task 2, a few significant changes were made to the **MeshDrawer** class constructor. At first, three new uniform variables were added: **ambientLoc** controlled ambient light intensity, **enableLightingLoc** controlled lighting enable/disable status, and **lightPosLoc** stored light position. The constructor was used to specify default values for these variables, which established initial states for lighting activation, ambient light intensity, and light position. Second, **normalCoordLoc**—the attribute location for vertex normals—was established. This property is essential for communicating each vertex's orientation information in relation to lighting computations. Furthermore, a specific WebGL buffer object called **normalbuffer** was developed in order to effectively hold normal vectors.

Additions were made to the **setMesh** method in order to integrate vertex normal data with vertex positions and texture coordinates. Extra lines were added to handle the updating of vertex normals, while the current capability for updating texture coordinates and vertex positions was kept. Using this all-inclusive method guarantees that all necessary vertex attribute data—positions, texture coordinates, and normals—are transferred to the GPU buffers for rendering and lighting computations in an efficient manner.

To smoothly incorporate lighting effects into the rendering process, the draw method received changes. In addition to the standard vertex position and texture coordinate rendering setup, **gl.uniform3fv(this.lightPosLoc, normalize([lightX, lightY, 5]))**; was included as a critical line to dynamically adjust the light position uniform in response to user input. This makes sure that the produced mesh has realistic and precise lighting effects. Furthermore, the technique now includes the line **gl.uniform1i(this.enableLightingLoc, true)**; to activate lighting effects, enabling the realistic representation of lighted surfaces. A line was added to disable lighting, **gl.uniform1i(this.enableLightingLoc, false);**, to ensure correct rendering states in following draw calls. In addition, the **updateLightPos()** function was used to make it easier to dynamically modify the light location while rendering.

To provide dynamic control over lighting effects in the WebGL application, changes were made to the **enableLighting** and **setAmbientLight** functions. A new line **gl.uniform1i(this.enableLightingLoc, show);** was added to the **enableLighting** method to update the uniform variable that controls the lighting's enable/disable state based on the supplied argument. This makes it possible to smoothly activate or deactivate lighting effects, increasing the rendering process' adaptability. In a similar vein, the **setAmbientLight** method now includes a line **gl.uniform1f(this.ambientLoc, ambient);** to update the uniform variable indicating ambient light intensity. This update adds to the realism and customization of the lighting effects in the produced scene by enabling dynamic adjustments to ambient light levels.

Changed also done to the **const meshFS**’s if statement. The surface normal **v\_normal** is first normalized guarantee reliable and consistent illumination calculations. The contribution from diffuse illumination is computed. Their alignment is measured by the dot product of the normalized surface normal and the normalized light direction **lightPos**. In order to stop negative lighting contributions, the max function is used to constrain negative dot products (which occur when light is behind the surface) to zero. This modification enhances the diffuse lighting calculation's authenticity. Secondly the texture color **texColor** is fetched and modulated using the total of the ambient illumination **ambient** and diffuse lighting **light**. By using this modulation, the final pixel color will accurately represent the combined effect of the lighting and texture effects. To sum up, this **if** statement makes sure that lighting effects are only taken into account when both lighting and texture rendering are turned on.