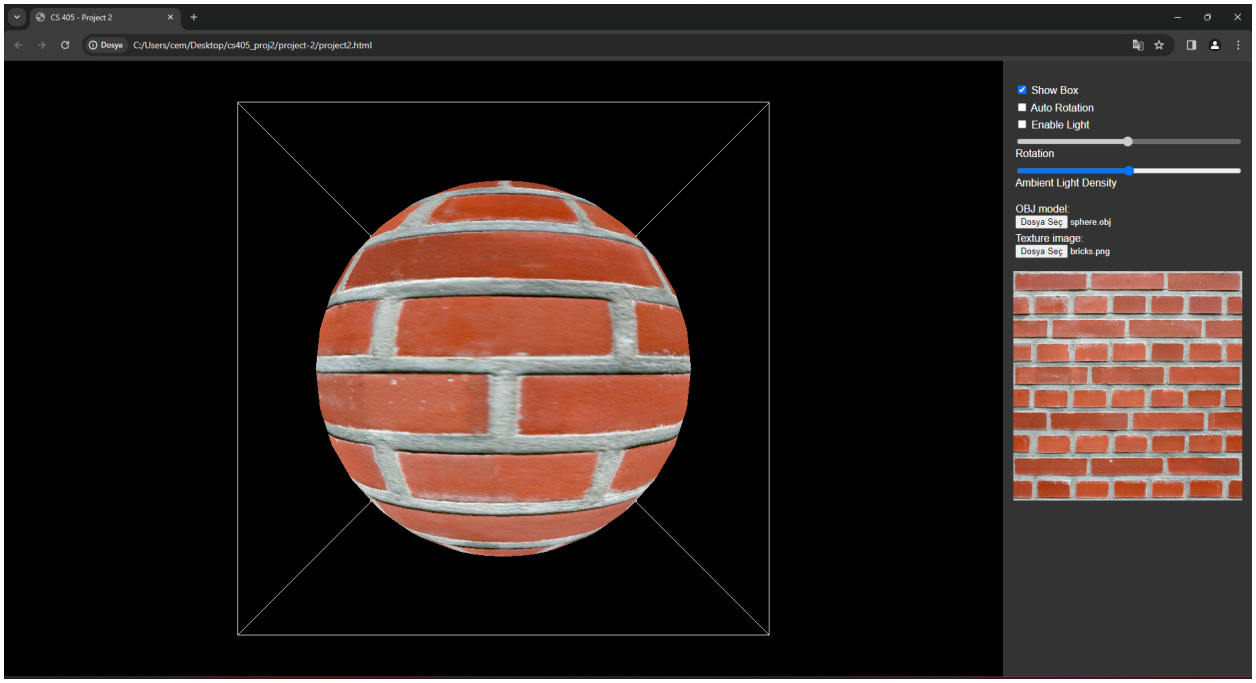
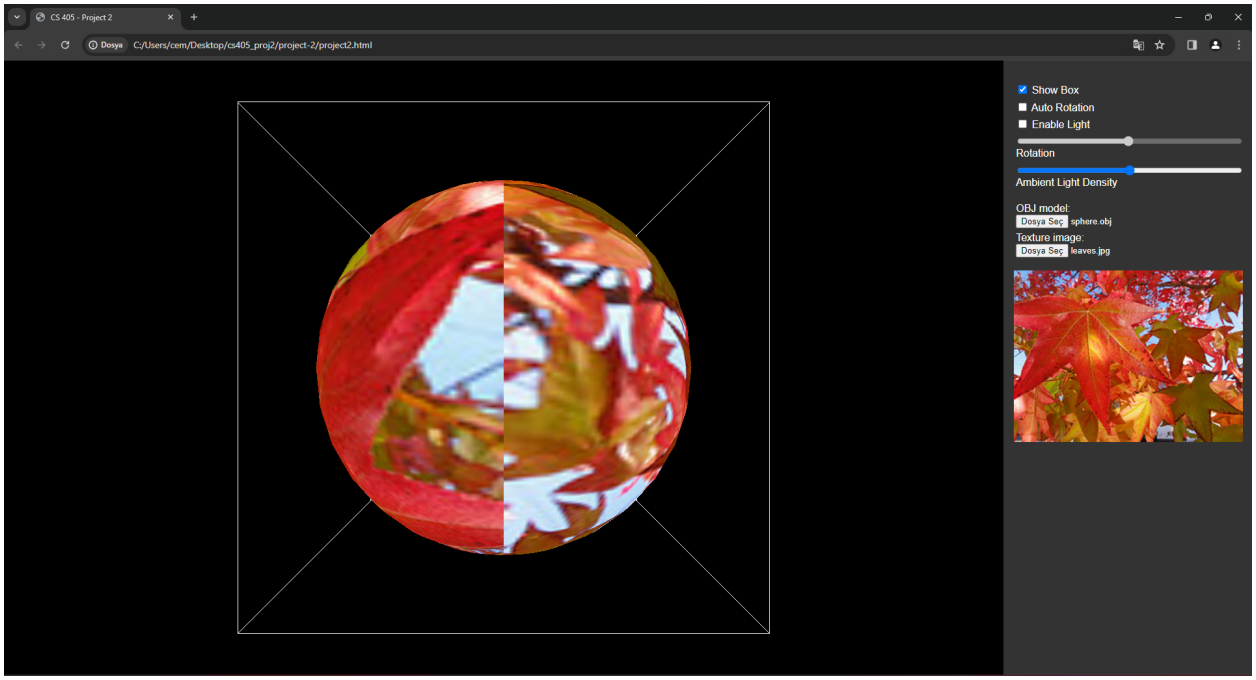
Task1

In Task 1, our goal is to adapt our code to handle non-power-of-two textures, like a 491x491 texture, using Mipmapping and wrapping techniques. The code lines gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE); and gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE); are used to configure the horizontal and vertical texture wrapping. By setting both to gl.CLAMP_TO_EDGE, we stretch the texture's border pixel across the shape's edge. This approach eliminates the repetition of textures and reduces edge artifacts. Additionally, the lines gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR); and gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR); are set for minification and magnification filtering, respectively, to linear. This choice ensures the texture remains smooth when scaled, either down or up, preventing a pixelated look. Consequently, these settings enable the application of the desired texture effectively on non-standard dimensions.

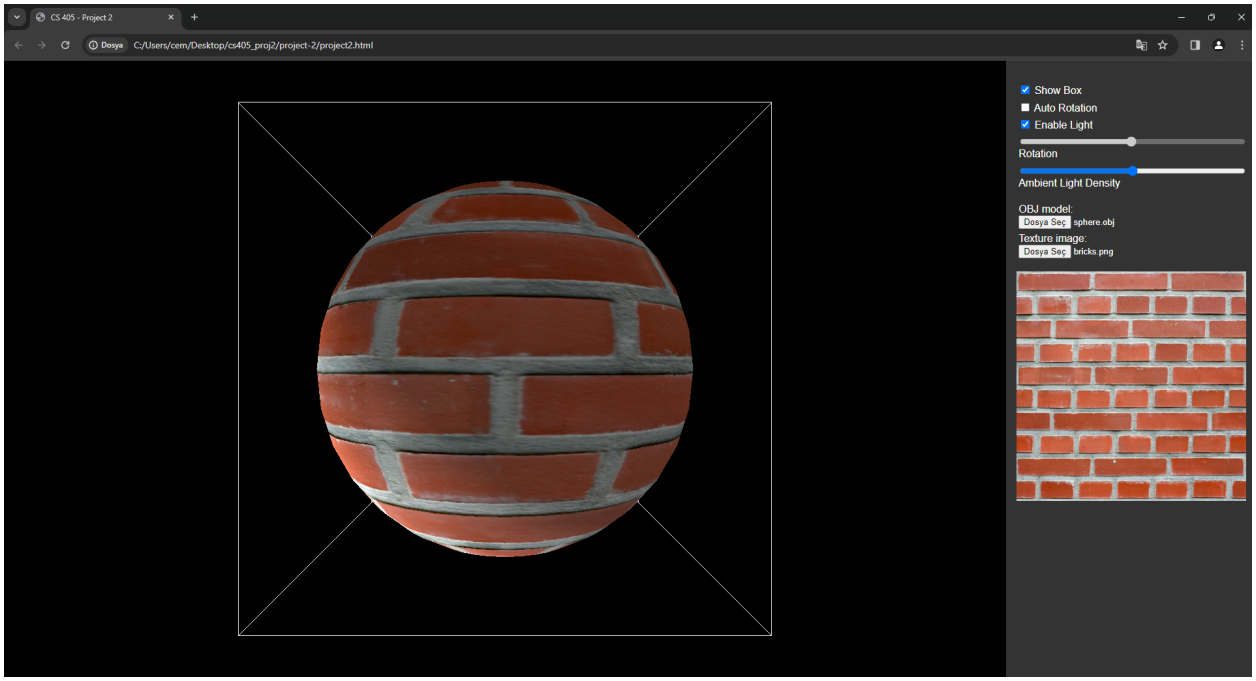


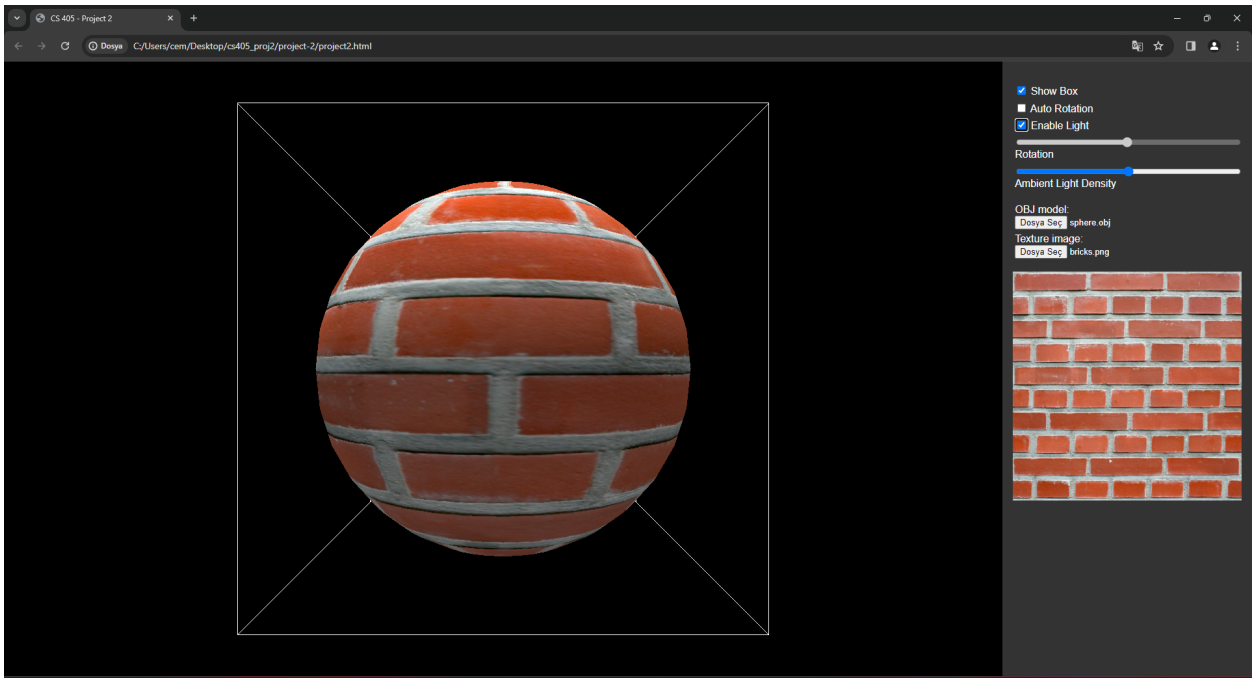


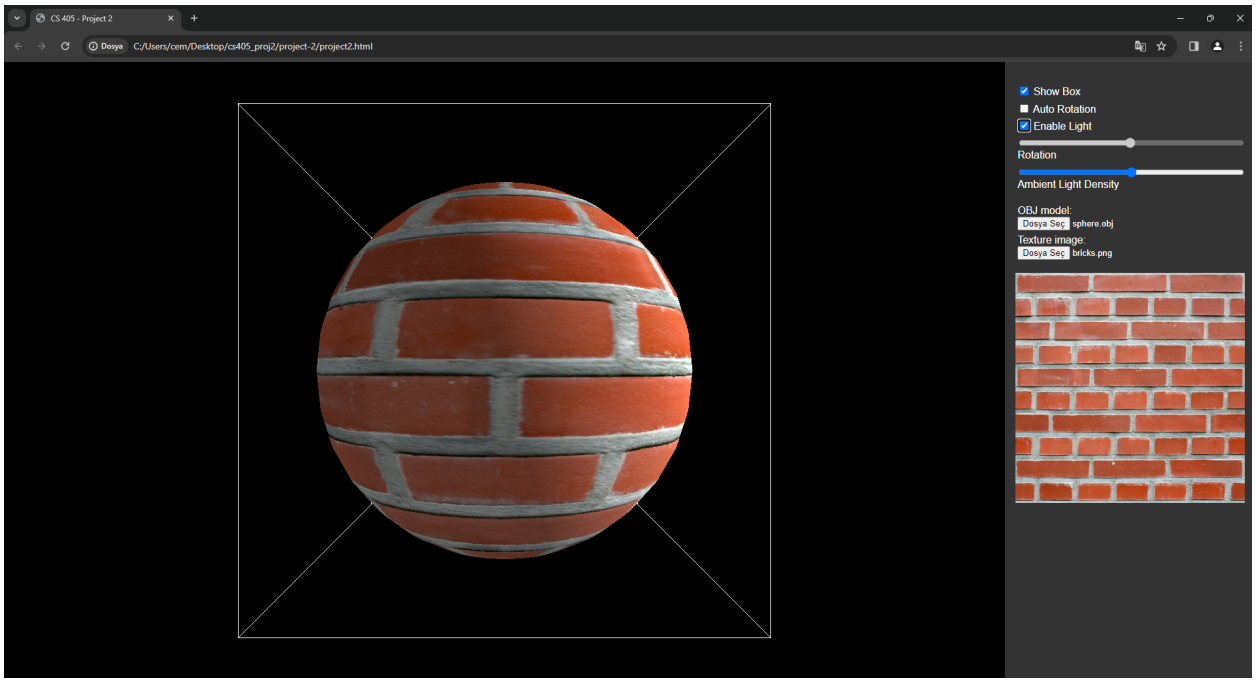

```
// The argument is an HTML IMG element containing the texture data.
setTexture(img) {
    const texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);

    // Set the texture image data
    gl.texImage2D(
        gl.TEXTURE_2D,
        0,
        gl.RGB,
        gl.RGB,
        gl.UNSIGNED_BYTE,
        img);

    // Check if dimensions are power of 2
    if (isPowerOf2(img.width) && isPowerOf2(img.height)) {
        // Generate mipmap for power of 2 textures
        gl.generateMipmap(gl.TEXTURE_2D);
    } else {
        // Set texture parameters for NPOT textures
        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);
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    }

    // Remaining setup is unchanged
    gl.useProgram(this.prog);
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, texture);
    const sampler = gl.getUniformLocation(this.prog, 'tex');
    gl.uniform1i(sampler, 0);
}


showTexture(show) {
    gl.useProgram(this.prog);
    gl.uniform1i(this.showTexLoc, show);
}
```

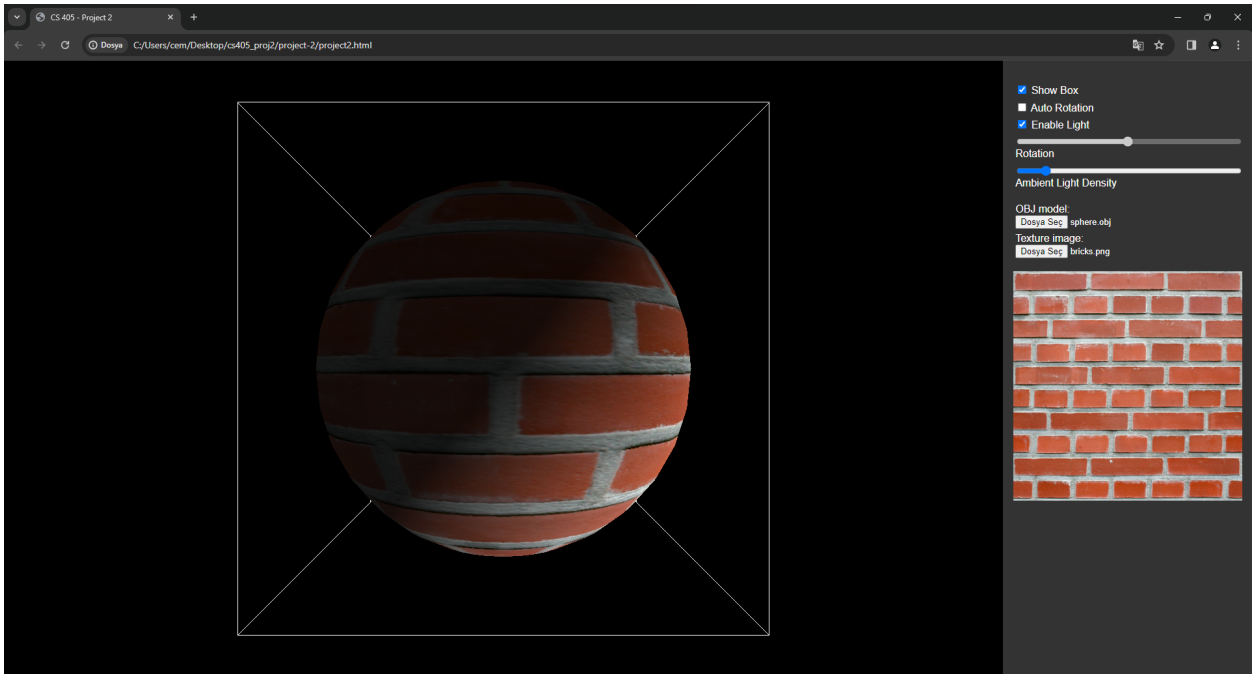
Task2

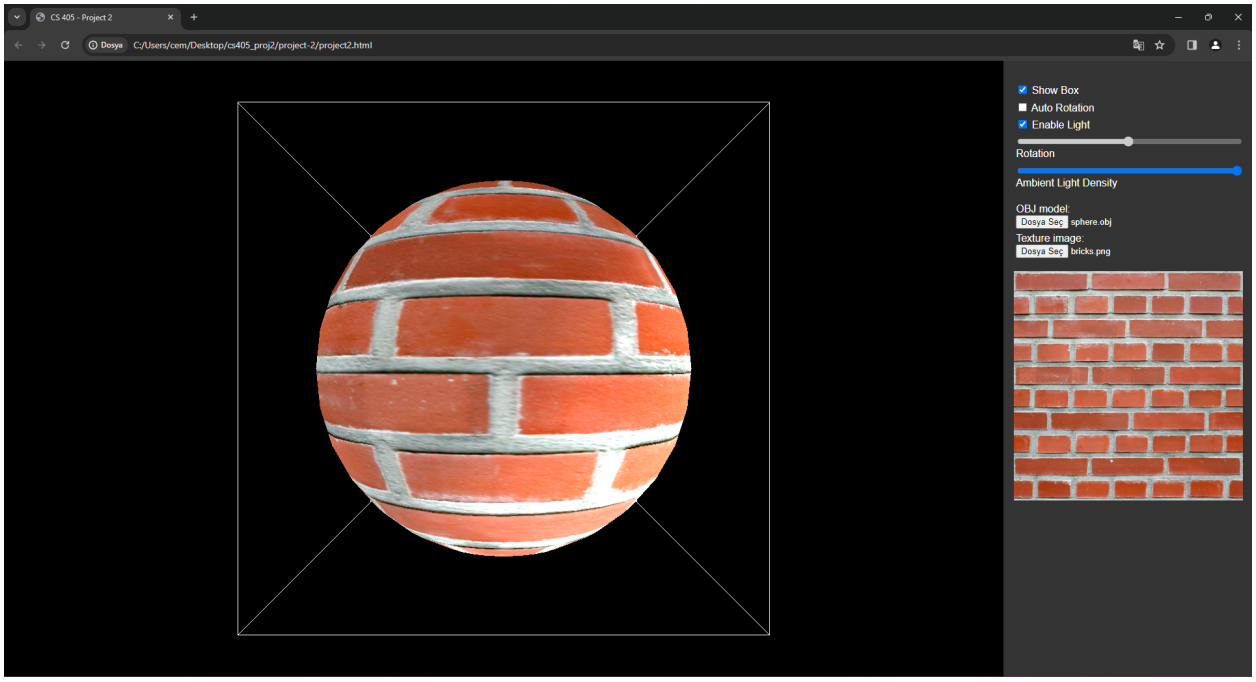
In Task 2, we developed a dynamic lighting system in our graphics application that allows users to adjust the light direction using arrow keys and control light density via a slider in the browser. This functionality was implemented through a combination of JavaScript and GLSL (OpenGL Shading Language), essential for rendering in OpenGL. We established necessary variables, including the light's position and ambient light location. The central component of this implementation is the meshfs function, which calculates the final color of each pixel based on input variables and detailed lighting calculations. Regarding texture handling, if 'showTex' is set to true, the fragment's color aligns with the texture color at specific coordinates; if false, a uniform color is used. The lighting effect is computed when 'enableLighting' is enabled, incorporating both ambient lighting for uniform scene illumination and diffuse lighting to simulate directional lighting. The direction of light is derived from a normalized 'lightPos', which typically represents the difference between the light position and the fragment position. The lighting intensity on surfaces is calculated using the dot product of the light direction and the fragment's normal vector. The final color of each fragment is a combination of ambient and diffuse lighting components. Without lighting enabled, the fragment's color defaults to the texture color or a uniform color, as specified.




CS 405 - Project 2
Dosya
C:/Users/cem/Desktop/cs405_proj2/project-2/project2.html
Show Box
Auto Rotation
Enable Light
Rotation
Ambient Light Density
OBJ model:
Dosya Seç
sphere.obj
Texture image:
Dosya Seç
bricks.png


CS 405 - Project 2
Dosya
C:/Users/cem/Desktop/cs405_proj2/project-2/project2.html
Show Box
Auto Rotation
Enable Light
Rotation
Ambient Light Density
OBJ model:
Dosya Seç
sphere.obj
Texture image:
Dosya Seç
bricks.png

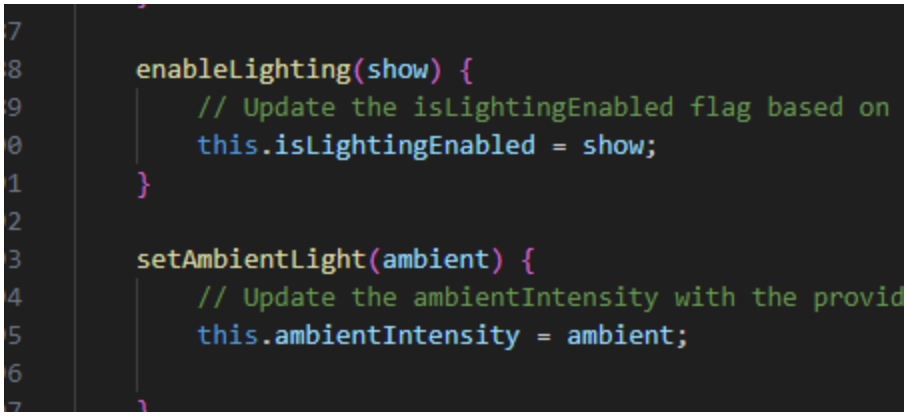

CS 405 - Project 2
Dosya
C:/Users/cem/Desktop/cs405_proj2/project-2/project2.html
Show Box
Auto Rotation
Enable Light
Rotation
Ambient Light Density
OBJ model:
Dosya Seç
sphere.obj
Texture image:
Dosya Seç
bricks.png


CS 405 - Project 2
Dosya
C:/Users/cem/Desktop/cs405_proj2/project-2/project2.html
Show Box
Auto Rotation
Enable Light
Rotation
Ambient Light Density
OBJ model:
Dosya Seç
sphere.obj
Texture image:
Dosya Seç
bricks.png

```
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.normalLoc = gl.getAttribLocation(this.prog, 'normal'); // Location for normal vector

    this.vertbuffer = gl.createBuffer();
    this.texbuffer = gl.createBuffer();
    this.normalbuffer = gl.createBuffer(); // Buffer for normal vectors

    this.numTriangles = 0;

    // Lighting related uniforms
    this.lightPosLoc = gl.getUniformLocation(this.prog, 'lightPos'); // Location for light position
    this.ambientLoc = gl.getUniformLocation(this.prog, 'ambient'); // Location for ambient light intensity
    this.enableLightingLoc = gl.getUniformLocation(this.prog, 'enableLighting'); // Location to enable/disable lighting

    // Initial values for lighting (can be adjusted later)
    this.lightPos = [2.0, 1.0, 1.0]; // Default light position
    this.ambientIntensity = 0.5; // Default ambient light intensity
    this.isLightingEnabled = false; // Flag to enable/disable lighting



}
```

```
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);

    // Update normal coordinates
    if (normalCoords) {
        gl.bindBuffer(gl.ARRAY_BUFFER, this.normalbuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(normalCoords), gl.STATIC_DRAW);
    }

    this.numTriangles = vertPos.length / 3;
}
```

```
// by the GetModelViewProjection function above.
draw(trans) {
    gl.useProgram(this.prog);

    // Set the transformation matrix
    gl.uniformMatrix4fv(this.mvpLoc, false, trans);

    // Bind and set up the vertex position attribute
    gl.bindBuffer(gl.ARRAY_BUFFER, this.vertbuffer);
    gl.enableVertexAttribArray(this.vertPosLoc);
    gl.vertexAttribPointer(this.vertPosLoc, 3, gl.FLOAT, false, 0, 0);

    // Bind and set up the texture coordinate attribute
    gl.bindBuffer(gl.ARRAY_BUFFER, this.texbuffer);
    gl.enableVertexAttribArray(this.texCoordLoc);
    gl.vertexAttribPointer(this.texCoordLoc, 2, gl.FLOAT, false, 0, 0);

    // Bind and set up the normal vector attribute
    if (this.normalLoc !== undefined) {
        gl.bindBuffer(gl.ARRAY_BUFFER, this.normalbuffer);
        gl.enableVertexAttribArray(this.normalLoc);
        gl.vertexAttribPointer(this.normalLoc, 3, gl.FLOAT, false, 0, 0);
    }

    // Update and set lighting related uniforms
    gl.uniform3fv(this.lightPosLoc, this.lightPos);
    gl.uniform1f(this.ambientLoc, this.ambientIntensity);
    gl.uniform1i(this.enableLightingLoc, this.isLightingEnabled);




    // Update the light position if necessary
    updateLightPos();


    gl.uniform3fv(this.lightPosLoc, this.lightPos);
    // Draw the mesh

    meshDrawer.lightPos = [-lightX, -lightY, 0.5];

    gl.drawArrays(gl.TRIANGLES, 0, this.numTriangles);
    //meshDrawer.lightPos = [lightX, lightY, 1.0];


}
```


enableLighting(show) {
// Update the isLightingEnabled flag based on
this.isLightingEnabled = show;
}
setAmbientLight(ambient) {
// Update the ambientIntensity with the provid
this.ambientIntensity = ambient;

```
const meshFS =
precision mediump float;

uniform bool showTex;
uniform bool enableLighting;
uniform sampler2D tex;
uniform vec3 color;
uniform vec3 lightPos;
uniform float ambient;

varying vec2 v_texCoord;
varying vec3 v_normal;

varying vec3 v_worldPos;


void main() {

    vec4 texColor = texture2D(tex, v_texCoord);
    vec4 finalColor;

    if (showTex) {
        finalColor = texColor;
    } else {
        finalColor = vec4(color, 1.0);
    }

    if (enableLighting) {
        // Ambient lighting
        vec3 ambientColor = ambient * finalColor.rgb;

        // Diffuse lighting
        vec3 norm = normalize(v_normal);
        //vec3 lightDir = normalize(lightPos - vec3(gl_FragCoord.xyz));
        vec3 lightDir = normalize(lightPos );
        float diff = max(dot(norm, lightDir), 0.0);
        vec3 diffuseColor = diff * finalColor.rgb;

        // Combine the two components
        vec3 combinedColor = ambientColor + diffuseColor;

        gl_FragColor = vec4(combinedColor, finalColor.a);
    } else {
        gl_FragColor = finalColor;
    }
}
`;
```