Solar System Simulation Report

Eray Öztürk 29097

December 19, 2024

1 Introduction

This report outlines the methodology followed to implement the three tasks for the Solar System Simulation project. Each task involved addressing a specific functionality:

- Task 1: Implementing the `draw` function for the scene graph.
- Task 2: Enhancing the fragment shader for proper diffuse and specular lighting calculations.
- Task 3: Adding Mars as a child node to the solar system with appropriate transformations and texture.

2 Task 1: Implementing the `draw` Function

The `draw` function was implemented to propagate transformations from parent nodes to child nodes in the scene graph. This ensured hierarchical transformations were applied correctly.

Code Implementation

```
    draw(mvp, modelView, normalMatrix, modelMatrix) {
        /**
         * @Task1 : Implement the draw function for the
             SceneNode class.
         */

        // Compute the node's transformation matrix
        const nodeTransform = this.trs.getTransformationMatrix
            ();

        // Update the transformation matrices
        const transformedModel = MatrixMult(modelMatrix,
            nodeTransform);
        const transformedModelView = MatrixMult(modelView,
            nodeTransform);
        const transformedMvp = MatrixMult(mvp, nodeTransform);

        // Use the provided normalMatrix to compute transformed
             normals
        const transformedNormals = MatrixMult(normalMatrix,
            nodeTransform)

        // Draw the MeshDrawer
        if (this.meshDrawer) {
            this.meshDrawer.draw(transformedMvp,
                transformedModelView, transformedNormals,
                transformedModel);
        }

        // Recursively call draw on all children
        for (const child of this.children) {
            child.draw(transformedMvp, transformedModelView,
                transformedNormals, transformedModel);
        }
    }
```

Listing 1: Implementation of the `draw` function.

Methodology

- Calculated the transformation matrix for the current node using its `TRS` object.
- Updated the model, model-view, and MVP matrices by multiplying with the current node's transformation.
- Recomputed the normal matrix for proper lighting effects.
- Recursively called the `draw` function for child nodes, passing the updated matrices.

3 Task 2: Enhancing the Fragment Shader

The fragment shader was updated to include diffuse and specular lighting calculations in addition to the ambient lighting already present. This provided realistic lighting effects.

Code Implementation

```
void main() {
    vec3 normal = normalize(vNormal); // Normalize the normal
    vec3 lightPos = vec3(0.0, 0.0, 5.0); // Position of the
        light source
    vec3 lightdir = normalize(lightPos - fragPos); // Normalize
         the light direction

    float ambient = 0.35;
    float diff = 0.0;
    float spec = 0.0;
    float phongExp = 8.0;

    /////////////////// BEGINNING OF TASK 2//////////////

    // Diffuse lighting calculation
    diff = max(dot(normal, lightdir), 0.0);

    // Specular lighting calculation
    vec3 viewDir = normalize(-fragPos); // View direction from
        the fragment position
    vec3 reflectDir = reflect(-lightdir, normal); // Reflect
        the light direction about the normal
    spec = pow(max(dot(viewDir, reflectDir), 0.0), phongExp);

    ////////////////END OF TASK 2/////////////////

    if (isLightSource) {
        gl_FragColor = texture2D(tex, vTexCoord) * vec4(1.0,
            1.0, 1.0, 1.0);
    } else {
        gl_FragColor = texture2D(tex, vTexCoord) * (ambient +
            diff + spec); // Set the fragment color
    }
}
```

Listing 2: Fragment shader with diffuse and specular lighting.

Methodology

- Calculated the diffuse component using the dot product of the light direction and the surface normal.
- Computed the specular component using the Phong reflection model, involving the reflection vector and the view direction.
- Combined ambient, diffuse, and specular components to determine the final fragment color.

4 Task 3: Adding Mars to the Solar System

Mars was added to the solar system as a child of the Sun node. Its geometry, texture, and transformations were appropriately configured.

Code Implementation

```
// Create a MeshDrawer for Mars
marsMeshDrawer = new MeshDrawer();
marsMeshDrawer.setMesh(sphereBuffers.positionBuffer,
    sphereBuffers.texCoordBuffer, sphereBuffers.normalBuffer);

// Set Mars texture
setTextureImg(marsMeshDrawer, "src/Mars_Surface.jpg");

// Create TRS for Mars
marsTrs = new TRS();
marsTrs.setTranslation(-6, 0, 0);  // Translate -6 units on the
     X-axis
marsTrs.setScale(0.35, 0.35, 0.35);  // Scale Mars to 0.35

// Create a SceneNode for Mars and add it as a child of the Sun
marsNode = new SceneNode(marsMeshDrawer, marsTrs, sunNode);
---------------------------------------------------------------
// Inside renderLoop, apply rotation to Mars
marsNode.trs.setRotation(0, 0, zRotation * 1.5);
```

Listing 3: Adding Mars to the solar system.

Methodology

- Initialized a `MeshDrawer` for Mars and applied the sphere mesh.
- Configured Mars' texture using `Mars_Surface.jpg`.
- Created a `TRS` object to set Mars' translation, scaling, and rotation.
- Added Mars as a child of the `sunNode` to integrate it into the scene graph.
- Applied dynamic rotation in the render loop to simulate Mars' orbit.