CS 405 Project 3

Ulaş Yıldız
24931

Introduction

This report describes the development and implementation of a Solar System simulation created using WebGL and JavaScript. Three main tasks were addressed in the project: the use of scene graph structures, the development of lighting with shader programming, and the addition of the planet Mars to the simulation. These tasks aimed at the creation of a 3D simulation optimised in terms of both graphical realism and performance. The code combines basic graphics techniques such as hierarchical object management, lighting calculations and shader programming.

Task 1: Stage Graphic Structure

The draw function for the SceneNode class is implemented in sceneNode.js. This function ensures that the transformations applied to the parent node are correctly transferred to all child nodes. The scene graph architecture consists of the following basic stages:

1. **Transformation Matrix Calculation**

Each node calculates its own transformation matrix using trs.getTransformationMatrix(). These transformation matrices are transferred to the model-view-projection (MVP), model-view-projection (ModelView), and normal matrices.

```
    transformedMvp = MatrixMult(transformedMvp, this.trs.getTransformationMatrix());
    transformedModelView = MatrixMult(transformedModelView, this.trs.getTransformationMatrix());
    transformedNormals = MatrixMult(transformedNormals, this.trs.getTransformationMatrix());
    transformedModel = MatrixMult(transformedModel, this.trs.getTransformationMatrix());
```

2. **Drawing the MeshDrawer Object**

If a node has a meshDrawer object, drawing is performed using the draw method.

```
if (this.meshDrawer) {
        this.meshDrawer.draw(transformedMvp, transformedModelView, transformedNormals, transformedModel);
}
```

3. **Recursive Drawing of Sub-Nodes**

The draw function is called recursively on the child nodes of the node.

```
this.children.forEach(child => {
            child.draw(transformedMvp, transformedModelView, transformedNormals, transformedModel);
});
```

This task ensured the proper establishment of the scene graph structure and the correct application of hierarchical transformations. In the Solar System, the rotation and motion of the planets connected to the Sun were successfully modelled.

Task 2: Diffuse and Reflected Lighting

In this task, the fragment shader in meshDrawer.js used in the project has been updated to support diffuse and specular lighting. These operations involve the implementation of the Lambertian and Phong reflection models and provide a more realistic illumination of surfaces relative to the light source.

1. Diffuse Light

Diffuse illumination is calculated with the Lambertian reflection model. This model assumes that light is evenly distributed over the surface and uses the angle between the surface normal and the direction of light. With the dot product method, the brightness of the parts of the surface facing the light was determined:

```
float facingLight = max(dot(normal, lightdir), 0.0);
diff = facingLight;
```

- **Normal Vector (normal):** The normal of the surface determines the angle at which light strikes the surface.
- **Light Direction (lightdir):** The direction vector from the light source.
- **Dot Product:** Calculates the angle between the normal vector and the light direction and determines how far the surface faces the light.

This calculation made the parts of the surface facing the light source brighter and the other parts darker. If the surface is not facing the light, the diffuse component is assigned zero:

```
if (facingLight > 0.0) {
   diff = facingLight;
} else {
   diff = 0.0;
}
```

2. Reflected Light

Specular lighting was implemented with the Phong reflection model. In order to simulate the reflection effects on glossy surfaces, the angle of the reflected light from the surface to the camera is taken into account:

```
vec3 reflectdir = reflect(-lightdir, normal);
```

```
spec = pow(max(dot(viewdir, reflectdir), 0.0), phongExp) * facingLight;
```

- **Reflection Vector (reflectdir):** The reflection of the light direction on the surface normal.
- **View Direction (viewdir):** The direction vector from the camera to the fragment.
- **Phong Exponent (phongExp):** Controls the sharpness of the reflection. Larger values provide sharper and more intense brightness.

Specular light was used to determine the bright spots of the surface. This calculation dynamically adjusted the brightness of the reflections according to the position of the surface between the light source and the camera.

3. **Final Lighting Account**

Diffuse and specular components were combined with ambient light to calculate the final colour of the surface:

```
gl_FragColor =  texture2D(tex, vTexCoord) * ( ambient + diff + spec );
```

- **Ambient Light:** Provides a basic level of illumination that prevents the surface from appearing completely dark.
- **Diffuse Light:** Determines the basic brightness according to the orientation of the surface to the light source.
- **Specular Light:** Simulates reflected light effects and highlights.

This formula provides more realistic lighting by taking into account both the overall brightness of the surface and reflection effects.

With this task, the surfaces were accurately illuminated depending on the light source and camera position. The Lambertian reflectance model determined the overall brightness level of the surface, while the Phong reflectance model enhanced visualisation by simulating bright spots. Normalising the surface normals and light directions improved the accuracy of the lighting calculations and optimised performance. As a result, the lighting model gave the scene a more realistic and impressive visual appearance.

Task 3: Addition of Mars

The planet Mars was added as a sub-node to the Solar System simulation. This process consists of the following stages:

1. **Mesh and Texture Definitions**

Mars was created using a spherical model (sphere mesh) and its texture was loaded from a URL:

```
setTextureImg(marsMeshDrawer, "https://i.imgur.com/Mwsa16j.jpeg");
```

2. Transformation and Positioning

Mars is placed and scaled at -6 units X axis around the Sun.

```
marsTrs.setTranslation(-6.0, 0, 0);
marsTrs.setScale(0.35, 0.35, 0.35);
```

3. Rotation Movement

The rotation speed of Mars is set at 1.5 times the rotation speed of the Sun.

```
marsNode.trs.setRotation(0, 0, zRotation * 1.5);
```

4. Hierarchical Connection

Mars is added as a sub-node to the Sun in the scene chart.

```
marsNode = new SceneNode(marsMeshDrawer, marsTrs, sunNode);
```

Mars was successfully added to the simulation and moved with the other planets in the hierarchical structure.

This project provided an effective application of hierarchy management, shader programming and scene graph structures in computer graphics. Task 1 established the scene graph structure, while Task 2 realised realistic lighting. Task 3 extended the scope of the model and added the planet Mars to the Solar System. The completion of these tasks enabled in-depth application of techniques in 3D graphics applications.