**CS 405 Project 3: Scene Graph + Illumination Report**

**Çağrı Can Saraçaydın 30984**

In the third project, we implemented a basic solar system model using WebGL, including scene graph transformations, lightning, and object creation features. The project consists of three tasks, by completing these tasks we step by step complete our basic solar system model with Sun, Earth, Moon, and Mars. In this report, I explained each tasks in detail.

**Task 1: Draw function implementation**

Initially, with the provided codebase, our project is simply a pitch-black screen. We started with implementing the draw function for the scene graph transformation system in the respective “sceneNode.js” file. Our goal is to utilize the hierarchy between the objects applying the transformations respectively in order to propagate the transformations between them. This is important especially for the rotation of the Moon and Earth while also rotating between their own axis. The multiplication order successfully works with the hierarchy between the objects. After the ModelView transformations completed then we calculate the Normals to correctly calculate their normals after the transformations. For the drawing we used for loop to iterate over all of the children with their respective transformed matrices.

draw(mvp, modelView, normalMatrix, modelMatrix) {

var nodeTransform = this.trs.getTransformationMatrix();

var transformedModel = MatrixMult(modelMatrix, nodeTransform);

var transformedModelView = MatrixMult(modelView, nodeTransform);

var transformedMvp = MatrixMult(mvp, nodeTransform);

var transformedNormals = getNormalMatrix(transformedModelView);

// ...

}

After our draw function implementation, we got:

- The Sun rotating around its own axis.

- The Earth orbits around the Sun while rotating around its own axis.

- The Moon orbits around the Earth following the Earth.

- There is no lighting still we will handle it the next task.

**Task 2: Lighting Implementation**

For Task 2, we implemented the Phong lighting model in the fragment shader. This is achieved by calculating both diffuse and specular lighting components to create realistic lighting of the planets.

// Calculate diffuse lighting

diff = max(dot(normal, lightdir), 0.0);

// Calculate specular lighting

vec3 reflectDir = reflect(-lightdir, normal);

vec3 viewDir = normalize(-vPosition);

spec = pow(max(dot(viewDir, reflectDir), 0.0), phongExp);

The lighting implementation creates realistic shading on the planets, with proper highlights and shadows based on their position relative to the sun. The diffuse light handles the general brightness of the surface with a light source towards it. The dot product between the direction of light and the normal vector calculates the cosine angle. The higher the cosine angle brighter the illumination of that object. Using max function we ignore the darker non illuminated side of the objects. The specular lighting controls the shininess of the object’s surface. By dot producing reflection vector and viewer’s direction inside the pow function, we control the shininess of the surface. As phongExp increases highlight effect becomes smaller. By combining these lighting components, the Phong lighting model creates a realistic visual effect on the planets. Diffuse lighting provides smooth shading across the surface, while specular lighting adds bright highlights that change based on the viewer's perspective.

**Task 3: Mars Implementation**

In Task 3, finally we enhanced our small solar system with the addition of the Mars . This involved creating a new celestial body with specific transformation parameters and proper parent-child relationships for achieving the realistic movement in the solar system. For it’s texture we added it from the provided link.

// Mars setup

marsMeshDrawer = new MeshDrawer();

marsMeshDrawer.setMesh(sphereBuffers.positionBuffer,

sphereBuffers.texCoordBuffer,

sphereBuffers.normalBuffer);

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

marsTrs = new TRS();

marsTrs.setTranslation(-6, 0, 0);

marsTrs.setScale(0.35, 0.35, 0.35);

In its implementation, we used the MeshDrawer similar to other objects within the solar system. We fed the MeshDrawer with utilizing the setMesh function with position, texture coordinate and normal. Provided texture image has been mapped using the setTextureImg function to achieve realistic representation. By following the task details about the Mars object with TRS, we set its location at -6,0,0 and reduced its size to make it even more realistic. Additionally, we set the rotation rate to 1.5x the base rotation speed.

**Results**

In the final implementation we got:

1. Hierarchical transformations allowing proper orbital moving relations

2. Realistic lighting that enhances the visual quality

3. Correct scaling and positioning of celestial bodies similar to existing system

4. Smooth rotation and orbit animations for the celestial bodies.