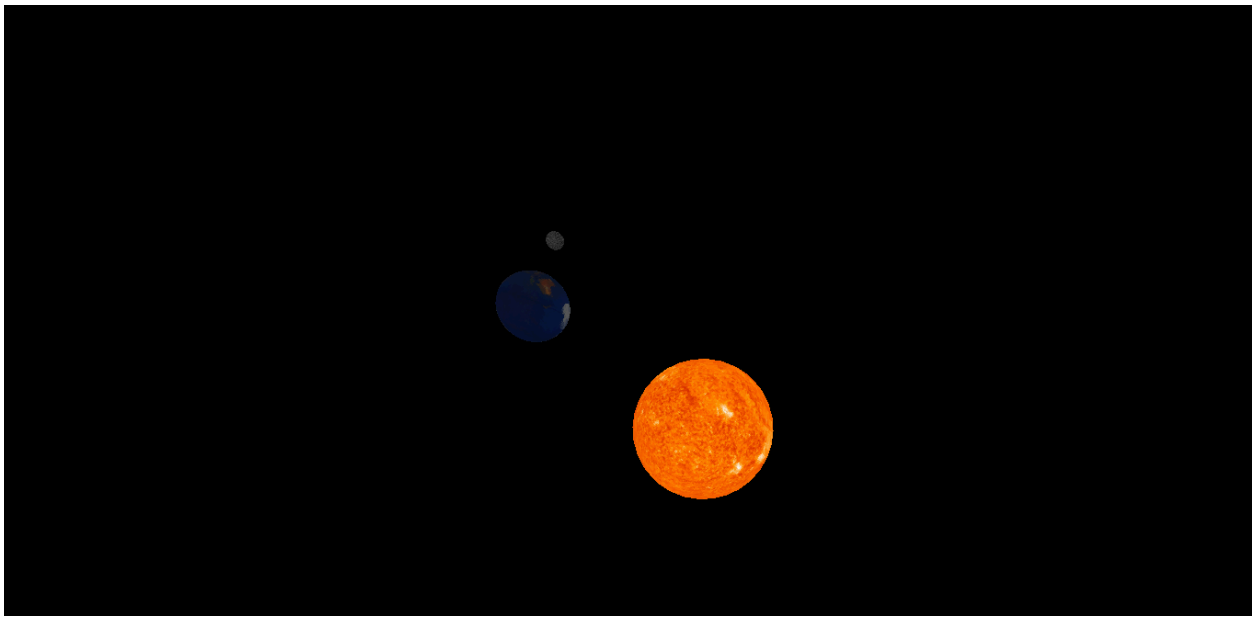
CS405 Project 3

Task1

To implement the draw function in a rendering context, you would design it to accept modelview, model matrix, and normal matrix as primary inputs. The function would first apply the model matrix, which transforms the model's vertices from their local coordinate space into world space, positioning and scaling them within the scene. Next, the normal matrix, often a transformation of the model matrix, is used to correctly transform the normals of the model for lighting calculations. Finally, the modelview matrix, a combination of the model and view matrices, is applied to transform the vertices from model space to view space. This process positions and orients the object correctly relative to the camera. The draw function also propagates these transformations to the object's children, ensuring that the entire hierarchy of objects in the scene is correctly rendered with consistent transformations.

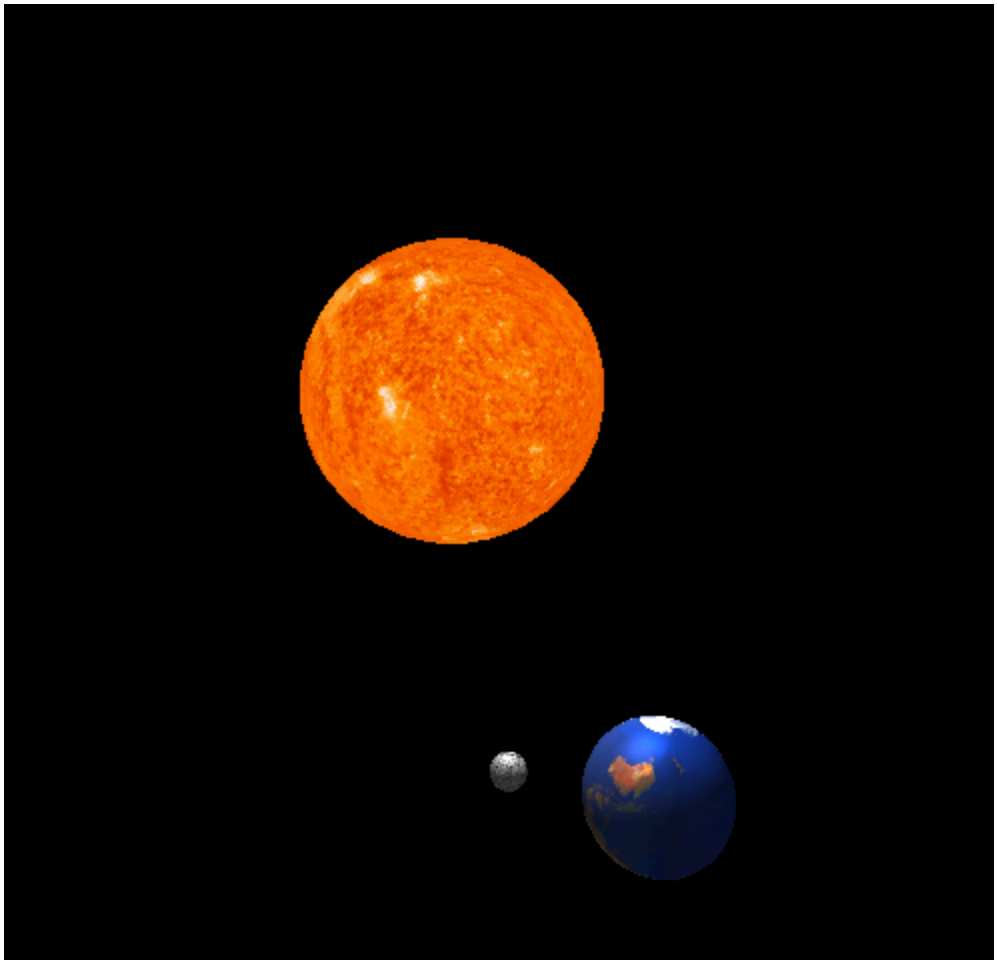
```
9
10  class SceneNode {
11      constructor(meshDrawer, trs, parent = null) {
12          this.meshDrawer = meshDrawer;
13          this.trs = trs;
14          this.parent = parent;
15          this.children = [];
16
17          if (parent) {
18              this.parent.__addChild(this);
19          }
20      }
21
22      __addChild(node) {
23          this.children.push(node);
24      }
25
26      draw(mvp, modelView, normalMatrix, modelMatrix) {
27          let localTransform = this.trs.getTransformationMatrix();
28
29          var transformedMvp = MatrixMult(mvp, localTransform);
30          var transformedModelView = MatrixMult(modelView, localTransform);
31          var transformedModel = MatrixMult(modelMatrix, localTransform);
32
33          if (this.meshDrawer) {
34              this.meshDrawer.draw(transformedMvp, transformedModelView, normalMatrix, transformedModel);
35          }
36
37          for (let child of this.children) {
38              child.draw(transformedMvp, transformedModelView, normalMatrix, transformedModel);
39          }
40      }
41
42
43
44
45  }
```



Task2

In Task2 we calculate the diffusion and specular lighting diffusion is how much light hits a surface, based on its normal vector and light direction. View Direction is the direction from the surface point to the viewer.Reflection vector is the direction of light reflected off the surface.Specular lighting is the shiny highlight on a surface, based on the viewer's position and the reflection direction.

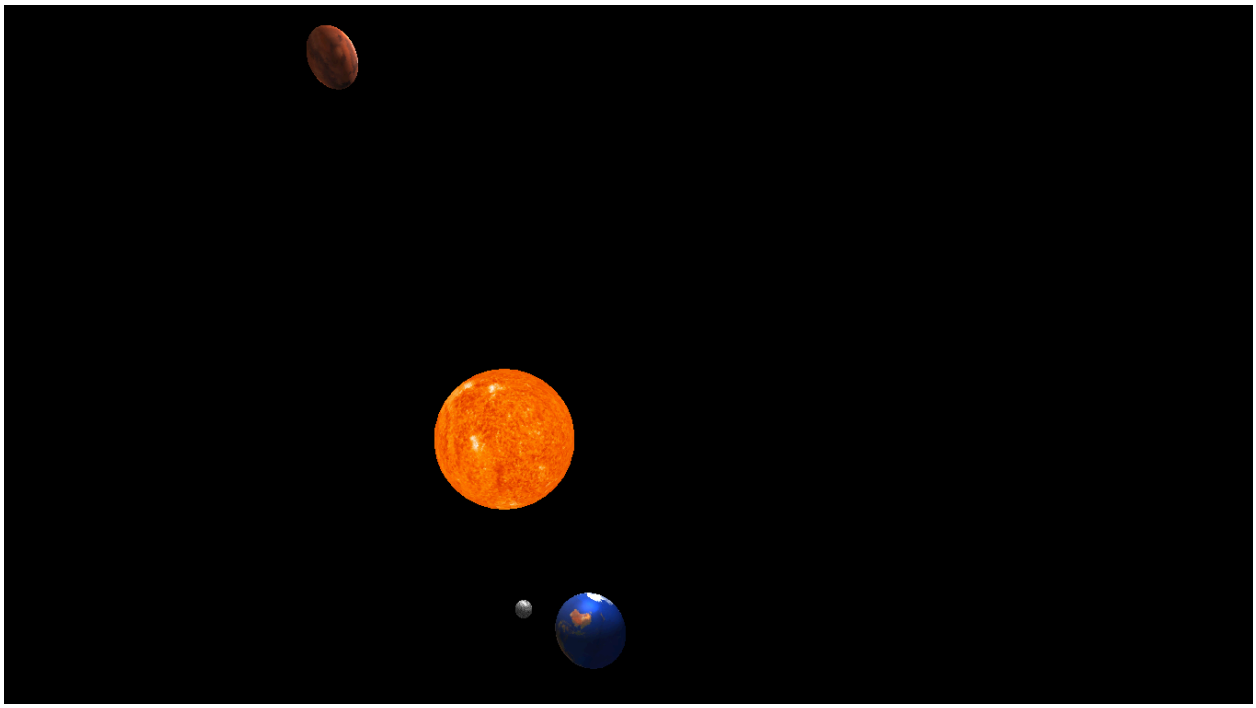
```
void main()
{
    vec3 normal = normalize(vNormal); // Normalize the normal
    vec3 lightPos = vec3(0.0, 0.0, 5.0); // Position of the light source
    vec3 lightdir = normalize(lightPos - fragPos); // Normalize the light direction

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

    ////////////////////////////////////////////////////////////////////////
    // PLEASE DO NOT CHANGE ANYTHING ABOVE !!!
    // Calculate the diffuse and specular lighting below.


    diff = max(dot(normal, lightdir), 0.0);
    vec3 viewDir = normalize(-fragPos);

    vec3 reflectDir = reflect(-lightdir,normal);
    spec = pow(max(dot(viewDir, reflectDir), 0.0), phongExp);









    // PLEASE DO NOT CHANGE ANYTHING BELOW !!!
    ////////////////////////////////////////////////////////////////////////

    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
    }
}
`;
```



Task3

In task3 of a solar system simulation project, we focus on creating a representation of Mars as a child object of the Sun, complete with its orbital dynamics and axial rotation. The primary objective is to accurately simulate Mars' orbit around the Sun, which involves calculating its orbital path and speed. This is achieved by defining Mars as a child object of the Sun in the scene graph, ensuring it inherits the correct positional context. Additionally, we implement Mars' axial rotation, specifically around its z-axis. This involves applying a rotational transformation to Mars, which is typically achieved by updating the rotation matrix in each frame of the

simulation to simulate the planet's day-night cycle. This rotation must be consistent with Mars' real axial tilt and rotation period to create a realistic simulation. The combination of orbital movement and axial rotation adds depth to the simulation, providing a more dynamic and accurate representation of Mars' behavior in the solar system.



```
moonNode = new SceneNode(moonMeshDrawer, moonTrs, earthNode);

/**
 * @Task3 : Add Mars to the solar system
 * Mars should be a child of the sun.
 * Mars should use sphere as the mesh object.
 * Mars should be translated by -6 units on the X axis with respect to the sun
 * Mars should be scaled to 0.35 for x,y and z coordinates
 * use the image on the link below as texture:
 * @link : https://i.imgur.com/Mwsa16j.jpeg
 *
 */
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);
marsNode = new SceneNode(marsMeshDrawer, marsTrs, sunNode);
//marsNode.trs.setRotation(0, 0, 1.5 * zRotation);
```

```
/**
 *@task3 : add rotation to mars on z-axis.
    the rotation should be 1.5 * zRotation
*/


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