**Task 1**

The function starts off by figuring out different transformation matrices. To acquire modified MVP matrix, it multiplies the input mvp matrix by the node's transformation matrix (this.trs.getTransformationMatrix()). To create a converted model-view matrix, same process is repeated for the modelView matrix. Also, it uses the modified model view matrix to compute a normal matrix (transformedNormals) which is required for accurate lighting computations in 3D graphics.

The function then determines if the node in question contains a meshDrawer, which is an object in charge of drawing the geometric mesh. The function invokes the meshDrawer.draw function, supplying modified matrix as parameters. The mesh of node is really rendered in this stage.

The function loops over each of vertex's children. It executes each siblin's draw function recursively sending identical converted matrices along to each sibling.

draw(mvp, modelView, normalMatrix, modelMatrix) {

/\*\*

\* @Task1 : Implement the draw function for the SceneNode class.

\*/

var transformedMvp = MatrixMult(mvp, this.trs.getTransformationMatrix());

var transformedModelView = MatrixMult(modelView, this.trs.getTransformationMatrix());

var transformedNormals = getNormalMatrix(transformedModelView);

var transformedModel = MatrixMult(transformedMvp, this.trs.getTransformationMatrix());

// Draw the MeshDrawer

if (this.meshDrawer) {

this.meshDrawer.draw(transformedMvp, transformedModelView, transformedNormals, transformedModel);

}

for (var x = 0; x < this.children.length; x++) {

this.children[x].draw(transformedMvp, transformedModelView, transformedNormals, transformedModel);

}

}

**OutputA screenshot of a computer

Description automatically generated**

**Task 2**

Based on the characteristics of the material and illumination, we determined color of each pixel. For precise illumination calculations, normal vector is first normalized. The shader determined direction of light in relation to position of fragment and establishes a light source.

After that, ambient lighting is computed to simulate in direct light in scene by giving the item a base color. Diffuse lighting comes next that gives a item sense of depth and contours by altering its hue according on angle at that light source is positioned in relation to it’s surface. For the replicate bright patches of light on object that mimic way light reflects off glossy surfaces, specular lighting is also calculated. The more exactly view direction lines up with light's reflection direction, the more noticeable this effect is. The texture of object is then combined with these lighting effects by using shader. That utilizes texture color directly for pieces recognized as light sources. Later, final color of fragment is calculated by multiplying the texture color by summation from lighting effects.

Code:

const meshFS = `

precision mediump float;

varying vec3 vNormal;

varying vec3 vPosition;

varying vec2 vTexCoord;

varying vec3 fragPos;

uniform sampler2D tex;

uniform bool isLightSource;

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 ambientStrength = 0.35;

vec3 ambient = ambientStrength \* vec3(1.0, 1.0, 1.0); // Ambient color

float diffIntensity = max(dot(normal, lightDir), 0.0);

vec3 diffuse = diffIntensity \* vec3(1.0, 1.0, 1.0); // Diffuse color

vec3 viewDir = normalize(-fragPos); // Assuming the view position is at the origin

vec3 reflectDir = reflect(-lightDir, normal);

float specIntensity = pow(max(dot(viewDir, reflectDir), 0.0), 8.0); // Using phongExp for shininess

vec3 specular = specIntensity \* vec3(1.0, 1.0, 1.0); // Specular color

vec3 result = ambient + diffuse + specular;

vec3 finalColor = result \* texture2D(tex, vTexCoord).rgb;

if (isLightSource) {

gl\_FragColor = vec4(texture2D(tex, vTexCoord).rgb, 1.0);

} else {

gl\_FragColor = vec4(finalColor, 1.0); // Set the fragment color

}

}

`;

**Result:**

**A group of planets in space

Description automatically generated**

**Task 3**

MeshDrawer initialized with marsMeshDrawer = new MeshDrawer(); then I set mesh with with buffers. The Mars object is then placed 6 units to left of the origin and scaled to 35% of it’s original size in all dimensions using transformation object that has been created and configured to do so. Later, new scene node for Mars is created, incorporating mesh drawer with Mars texture and the transformation settings. This Mars node is then added as a sibling to sun node, creating a hierarchical relationship in the scene graph that represents model of the solar system.

marsMeshDrawer.setMesh(sphereBuffers.positionBuffer, sphereBuffers.texCoordBuffer, sphereBuffers.normalBuffer);

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

var marsTrs = new TRS();

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

marsTrs.setScale(0.35, 0.35, 0.35);

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

**Output:**

A group of planets in space

Description automatically generated