Atahan Bozkuş 28471 Project-3 Report

TASK 1)

The draw function is used to draw this node and its child nodes.

- The “draw(mvp, modelView, normalMatrix, modelMatrix)” function takes the matrices required for drawing (MVP, ModelView, Normal and Model Matrices) as parameters.

- “var transformationMatrix = this.trs.getTransformationMatrix();” row calculates the transformation matrix of the node. This matrix represents the position, scaling, and transformation of the node.

- “var transformedMvp = MatrixMult(mvp, transformationMatrix);” The line calculates the node-specific transformed MVP matrix by multiplying the MVP matrix by the node's transformation matrix.

- “var transformedNormals = MatrixMult(normalMatrix, transformationMatrix);” The row calculates the node-specific transformed normal matrix by multiplying its normal matrix by the node's transformation matrix.

- “var transformedModelView = MatrixMult(modelView, transformationMatrix);” The line calculates the node-specific transformed ModelView matrix by multiplying the ModelView matrix by the node's transformation matrix.

- “var transformedModel = MatrixMult(modelMatrix, transformationMatrix);” The line calculates the node-specific transformed Model matrix by multiplying the Model matrix by the node's transformation matrix.

- The “if (this.meshDrawer) {...}” statement, if the node has a MeshDrawer object (that is, contains a visual element), performs the operations required to draw that element.

- “this.meshDrawer.draw(...);” line performs drawing of the MeshDrawer object using transforming matrices.

- The “for(var child of this.children){...}” loop repeats the same drawing process for all child nodes of the current node. This ensures that each object in the scene graph is correctly transformed and drawn.

TASK 2)

The fragment shader you have named meshFS is a part of a shader program used in WebGL. This shader is used for lighting and coloring mesh objects.

- “precision mediump float;”: Determines the precision level of floating point numbers to be used in the shader. In this case, “(mediump)” sensitivity was used.

- “varying vec3 vNormal; “ : “varying” keyword is used to transfer data from vertex shader to fragment shader. These lines define variables for the normal vector “(vNormal)”, position “(vPosition)”, texture coordinates “(vTexCoord)” and fragment position “(fragPos)”.

- “uniform sampler2D tex;” : “uniform” variables are constant data sent to the shader by the CPU. These lines define the texture sampling “(tex)” and whether there is a light source “(isLightSource)”.

- “void main() {...}” : Shader's main function. It is called once for each fragment.

- “vec3 normal = normalize(vNormal);” : Normalizes the incoming normal vector.

- “vec3 lightPos = vec3(0.0, 0.0, 5.0);” : Defines the position of the light source.

- “vec3 lightdir = normalize(lightPos - fragPos);” : Calculates and normalizes the light direction.

- “float ambient = 0.35;” : Defines the ambient light intensity.

- “float diff = 0.0; float spec = 0.0;” : Sets initial values for diffuse and specular (reflected) light intensities to zero.

- “float phongExp = 8.0;” : Defines the brightness coefficient to be used in the Phong illumination model.

- “if (isLightSource) {...}” : If the fragment is a light source, assigns the texture color directly to the fragment color. If not, it adjusts the texture color and determines the fragment color using the calculated ambient, diff and spec values.

TASK 3)

How the planet Mars was added and moved around the scene.

- “marsNode.trs.setRotation(0, 0, 1.5 * zRotation);” : This line allows the planet Mars to rotate in the Z axis. The “zRotation” variable is a value that increases over time, and here a rotation of 1.5 times this value is applied. This represents the rotation of Mars around its axis.

- “var marsMeshDrawer = new MeshDrawer();” : A “MeshDrawer” object is created for Mars. The “MeshDrawer” class contains the necessary adjustments and functions to draw a mesh (3D model).

- “var marsTrs = new TRS();” : A “TRS” (Translation, Rotation, Scale) object is created for Mars. This object is used to manage the position and size of Mars in the scene.

- “marsMeshDrawer.setMesh(sphereBuffers.positionBuffer, sphereBuffers.texCoordBuffer, sphereBuffers.normalBuffer);” : This line sets the mesh, i.e. the 3D model, of Mars. Here, position, texture coordinates and normal information are retrieved from “sphereBuffers”, which shows that Mars has a spherical shape.

- “setTextureImg(marsMeshDrawer, "https://i.imgur.com/Mwsa16j.jpeg");” : Sets the surface texture image of Mars. The ”setTextureImg” function loads the image from the specified URL and applies it to Mars' mesh.

- “marsTrs.setTranslation(-6, 0, 0);” : Positions Mars at a distance of -6 units from the Sun on the X axis.

- “marsTrs.setScale(0.35, 0.35, 0.35);” : Scales the size of Mars. A realistic size ratio is achieved by reducing the size by 0.35 on all axes.

- “marsNode = new SceneNode(marsMeshDrawer, marsTrs, sunNode);” : Finally, a “SceneNode” for Mars is created and this is added to the Solar system as a node in the scene graph. “sunNode” is the parent node of Mars, which indicates that Mars will move relative to the Sun.