CS405 PROJECT-3 REPORT

MELİH ÇAĞAN ARI

28426

IMPORTANT NOTE: As there were errors in the original JPG links associated with the Sun, Earth, Moon, and Mars at the beginning of the project, I have replaced these links with color PNG files to ensure proper visualization of the animation. Here's the representations:

- Yellow=> Sun, Blue=> Earth, White=> Moon, Yellow-Red=> Mars

INTRODUCTION

In the pursuit of advancing computer graphics proficiency, a set of tasks was undertaken to implement a scene graph and rendering functionalities. These tasks, assigned by Selim Hoca, delve into fundamental aspects of computer graphics, ranging from hierarchical transformations to shader-based lighting computations. The objective is to construct a dynamic and visually immersive graphics environment by developing a structured scene graph and incorporating advanced shader techniques. Through these tasks, the aim is not only to achieve practical implementation but also to deepen understanding and expertise in the intricate aspects of computer graphics programming.

TASK-1

The objective of this task is to implement the draw function for the **SceneNode** class. The draw function is responsible for rendering the associated **MeshDrawer** object, considering transformations applied to both the current node and its parent node in the scene graph.

1. **Parent Node Transformation:**
   - Check if the current node has a parent (**this.parent**).

- If a parent exists, retrieve the parent's transformation matrix using **this.parent.trs.getTransformationMatrix()**.
- Apply the parent's transformation to the input transformation matrices (**mvp**, **modelView**, **normalMatrix**, **modelMatrix**) using the **matrixMultiply** function.
- Log a confirmation message to the console for verification.

2. **Current Node Transformation:**
    - Obtain the current node's transformation matrix using **this.trs.getTransformationMatrix()**.
    - Apply the current node's transformation to the input transformation matrices using the **matrixMultiply** function.
    - These updated matrices (**transformedMvp**, **transformedModelView**, **transformedNormals**, **transformedModel**) now represent the combined transformations of both the parent and current node.
3. **Drawing MeshDrawer:**
    - Check if the current node has a **MeshDrawer** object (**this.meshDrawer**).
    - If a **MeshDrawer** exists, call its **draw** function with the updated transformation matrices (**transformedMvp**, **transformedModelView**, **transformedNormals**, **transformedModel**).
4. **Recursive Drawing for Children:**
    - Iterate through each child node (**this.children**) of the current node.
    - Recursively call the **draw** function for each child, passing the updated transformation matrices.

TASK-2

The objective of this section of the vertex shader (**meshVS**) is to compute the diffuse and specular lighting components for a given vertex in the 3D space. This is part of the vertex processing stage in a graphics pipeline.

1. **Diffuse Light Calculation:**
    - Compute the dot product between the normal vector (**normal**) and the light direction (**lightdir**).
    - Use the **max** function to ensure that the diffuse term is always non-negative.
    - Assign the result to the variable **diff**, representing the diffuse lighting component.
2. **Specular Light Calculation:**
    - Compute the view direction (**viewDir**) by normalizing the negative vertex position (**-vPosition**). This assumes that the viewer is looking along the negative z-axis.
    - Compute the reflection direction (**reflectDir**) using the **reflect** function, considering the negative light direction and the normal vector.
    - Compute the specular component (**spec**) using the Phong reflection model, raising the result of the dot product between the view direction and reflection direction to a high power (128.0 in this case).
    - The **pow** function is used for the high power operation.

TASK-3

The objective of Task 3 is to create a scene node for the planet Mars, configure its transformation, associate a mesh drawer, and set texture and material properties.

1. **Setting Rotation:**
    - Use the **marsNode.trs.setRotation(0, 0, zRotation * 1.5);** line to set the rotation of the Mars node around its own axis.

- The **zRotation** variable is likely a dynamic parameter controlling the rotation, and it's scaled by 1.5 to achieve a faster rotation compared to the standard rate.

2. **Setting Mesh for Mars:**
    - Use **marsMeshDrawer.setMesh(sphereBuffers.positionBuffer, sphereBuffers.texCoordBuffer, sphereBuffers.normalBuffer);** to associate the mesh data (position, texture coordinates, normals) with the Mars mesh drawer (**marsMeshDrawer**).
    - This assumes that **sphereBuffers** contains the necessary buffers for a spherical mesh.
3. **Setting Texture:**
    - Call the **setTextureImg** function to set the texture for the Mars mesh drawer.
    - The provided URL (**"https://media.istockphoto.com/id/1455300694/tr/foto%C4%9Fraf/coders-colleagues-working-on-new-project-together.jpg?s=1024x1024&w=is&k=20&c=gEe8gYrVlpyDnpDNR9CRN79dJ-47Q3ZSzDGguZlBRNU="**) is used to load the texture image.
4. **Setting Transformation (Translation, Scaling):**
    - Create a new transformation object (**marsTrs = new TRS();**) for Mars using the **TRS** class.
    - Use **marsTrs.setTranslation(-6.0, 0, 0);** to set the translation of Mars, positioning it at an offset of (-6.0, 0, 0) from its parent (likely the sun).
    - Use **marsTrs.setScale(0.35, 0.35, 0.35);** to set the scaling of Mars, making it smaller compared to its original size.
5. **Creating Scene Node:**
    - Instantiate a new scene node (**marsNode = new SceneNode(marsMeshDrawer, marsTrs, sunNode);**) for Mars.
    - Associate the Mars mesh drawer and transformation with the scene node.

- Set the parent of Mars node to be the **sunNode**, assuming a hierarchical scene graph structure.

CONCLUSION

In conclusion, the series of tasks undertaken for the implementation of a scene graph and rendering functionalities for a graphics environment exhibits a systematic and modular approach. Task 1 focused on implementing the draw function for the **SceneNode** class, allowing for hierarchical transformations to be applied seamlessly from parent to child nodes. The methodology employed ensures flexibility and modularity in rendering, with the successful incorporation of parent-child relationships in the scene graph.

Task 2 delved into the vertex shader (**meshVS**), providing a clear and concise methodology for the computation of both diffuse and specular lighting components. The vertex shader's implementation enhances the visual realism of the rendered objects by simulating the interaction of light with the surface, contributing to a more immersive graphics experience.

Finally, Task 3 involved setting up a scene node for the planet Mars within the scene graph. This encompassed configuring its transformation, associating a mesh drawer, and applying texture and material properties. The methodology for Task 3 showcased a well-structured approach, from setting up the rotation and scaling of Mars to associating the appropriate mesh and texture.

The overall conclusion underscores the accomplished integration of Mars into the scene graph, showcasing a collective achievement in constructing a dynamic and visually compelling graphics environment. Furthermore, the project assigned by Selim Hoca has significantly deepened my understanding of scene graphs and shaders. The tasks undertaken not only contributed to the practical implementation of hierarchical transformations, lighting computations, and texture mapping but also provided valuable insights into the intricate aspects of computer graphics. This hands-on experience has proven instrumental in expanding my knowledge and proficiency in the realm of computer graphics programming.