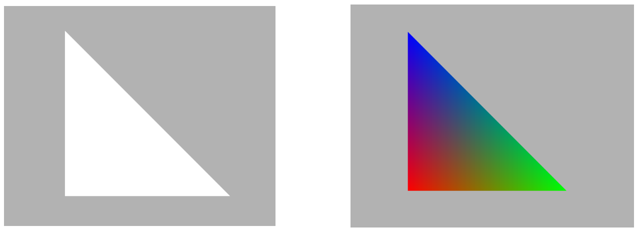
Project Report: Developing an Interactive 3D Solar System Using OpenGL

Berti Cecilia

Telecom Paris
Interactive 3D Application Development

1 A single Triangle

To create the triangle, I generated and filld three arrays: the vertex position array, the triangle index array, and the color array. The vertex position array contains the coordinates of the triangle's vertices, while the triangle index array defines the order in which these vertices are connected to form the triangle. Additionally, a separate color array is defined to specify the color of each vertex.



2 Sphere Mesh

For this section, I created a new class `Mesh.cpp` (with the corresponding header Mesh.hpp) where I implemented the `genSphere()` method. This method generates a spherical mesh by computing vertex positions and normals using spherical coordinates, and then creates triangle indices to define the mesh's geometry. I then set up the buffers, associating them with a vertex array object (VAO) to manage the data layout in the GPU. Finally, the mesh is initialized in the `main.cpp` file by calling the `init()` function.

2.1 Shading

To ensure the normals were correctly oriented, I first modified the shaders to visualize the normals as colors. Then, to reproduce the Phong lighting effect, I updated the fragment shader to incorporate the camera position and the contributions of the lighting effects.

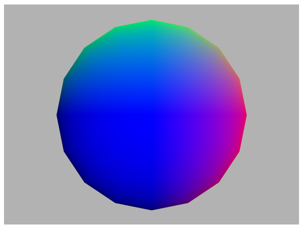


Fig. 1: Resolution=16

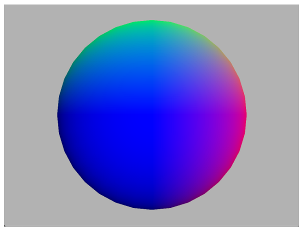


Fig. 2: Resolution=32

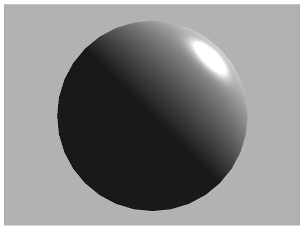
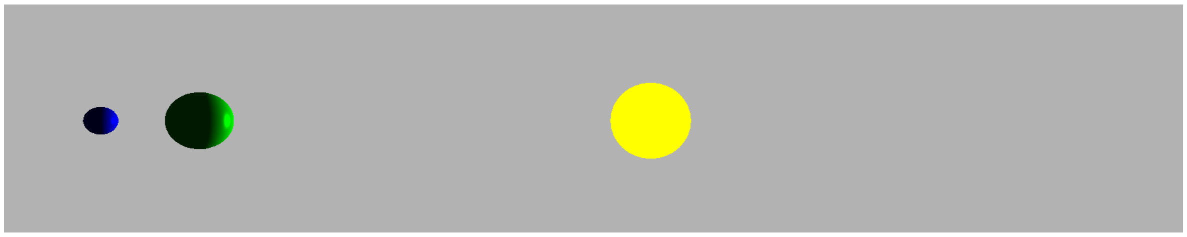


Fig. 3: Phong Lightning

3 Three planets

To implement this step, I rendered the same sphere mesh three times to represent the three planets. I defined three model transformation matrices and modified the `render()` function in `main.cpp` accordingly. Additionally, I introduced a boolean variable `isSun` to enable the fragment shader to differentiate between applying ambient lighting for the sun and using Phong lighting for the other planets. At this step I also needed to adjust the camera position to visualize all three planets.



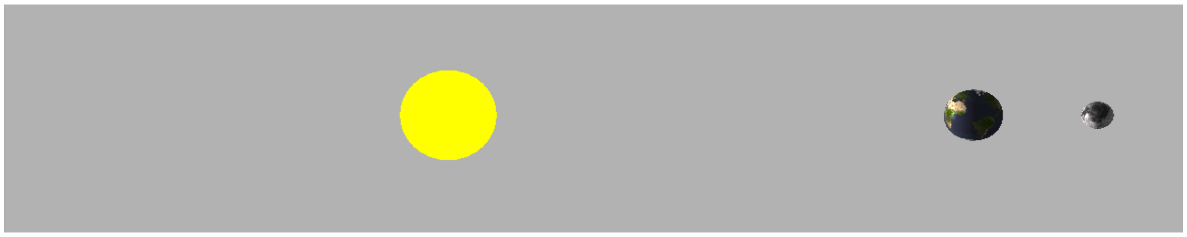
4 Animation

Given the radius information from the previous step and the new details about the orbital periods, I implemented the `update()` function to calculate the transformations for Earth and its moon based on their rotation and orbital speeds. The Earth's and moon's model matrix incorporates translation, scaling and rotation.

5 Planet Textures

In this part, I modified the `Mesh.cpp` class to compute the texture coordinates (u, v) for the sphere mesh. Initially, I encountered issues with the texture coordinate calculations, which led me to adjust my fragment shader to visualize the texture direction. After some troubleshooting, I discovered that the problem stemmed from how the vertex buffer was generated and the texture loading path. I then followed the provided instructions to update the program, added a vector for texture coordinates in the shaders, and adjusted the final color calculation in the fragment shader by multiplying the texture color with the lighting components. I also had to modify the y and z axis because the orientation was initially wrong.

```
vec3 result = ambient + diffuse + specular;
FragColor = vec4(texColor * result, 1.0);
```



6 Extension

For the extension, I added two additional planets, Mars and Venus, using adequate proportions. I also changed the background color to a darker tone to create a more realistic space environment. Lastly, I updated the `KeyCallback()` function to include keyboard shortcuts:

↑ Zoom in
↓ Zoom out
→ Rotate to the right
← Rotate to the left