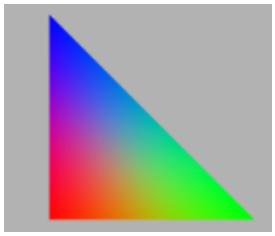
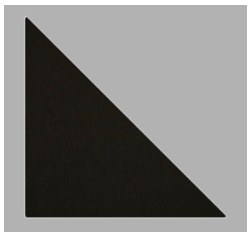
Boulay
Celio

IGR 200 – Solar System Report

The triangles :

At the beginning I ran into an issue, the triangle appeared black. This is because I was executing the executable inside the src/build folder, while the shaders were in the src folder.

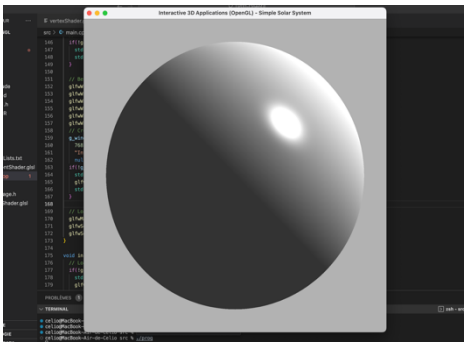
Sphere Mesh :

```
class Mesh {
  public:
  void init();
  void render();
  static std::shared_ptr<Mesh> genSphere(const size_t resolution=16);

  glm::mat4 getMeshMatrix() const {return g_mesh;}
  void setMeshMatrix(const glm::mat4& newMatrix) {g_mesh = newMatrix;}

  private:
  std::vector<float> m_vertexPositions;
  std::vector<float> m_vertexNormals;
  std::vector<unsigned int> m_triangleIndices;
  GLuint m_vao = 0;
  GLuint m_posVbo = 0;
  GLuint m_normalVbo = 0;
  GLuint m_ibo = 0;
  glm::mat4 g_mesh;
  std::vector<float> m_vertexTexCoords;
  GLuint m_texCoordVbo = 0;
};
```

The Mesh class contains all the information needed to create and render the sphere. In the genSphere() method, the coordinates of all vertices are calculated.

Then, according to the Phong Lighting Model, we calculate the lighting of each sphere, in the fragmentShader.

```
void main() {

    vec3 n = normalize(fNormal);
    vec3 lightPos = vec3(0.0, 0.0, 0.0); // Position du Soleil
    vec3 l = normalize(lightPos - fPosition.xyz);

    vec3 v = normalize(camPos - fPosition.xyz);   //view vector
    vec3 r = reflect(-l, n);    //reflection vector

    // Ambient color
    //vec3 ambient = vec3(0.2, 0.2, 0.2);
    //vec3 ambient = 0.2 * objectColor;


    // Diffuse color
    vec3 diffuseColor = vec3(1.0, 1.0, 1.0);
    float diff = max(dot(n, l), 0.0);
    vec3 diffuse = diff * diffuseColor;

    // Specular color
    vec3 specularColor = vec3(1.0, 1.0, 1.0); // Set your specular color here
    float shininess = 32.0; // Shininess factor
    float spec = pow(max(dot(r, v), 0.0), shininess);
    vec3 specular = spec * specularColor;
```



The variables such as the position are calculated in the main.cpp, then passed to the vertexShader, and finally to the fragmentShaders that handles de final calculations.

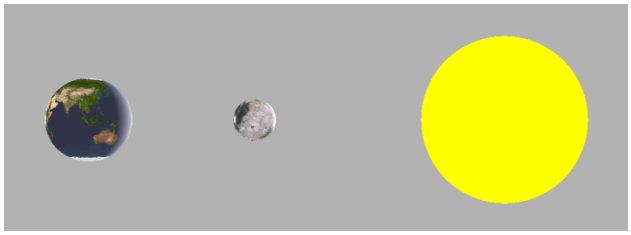
Three Planets :

To generate three different planets, I created 3 spehres : earthMesh, sunMesh and moonMesh, then multiplied them with transformation matrices to adjust the position, the tilt (for the earth), or the size.

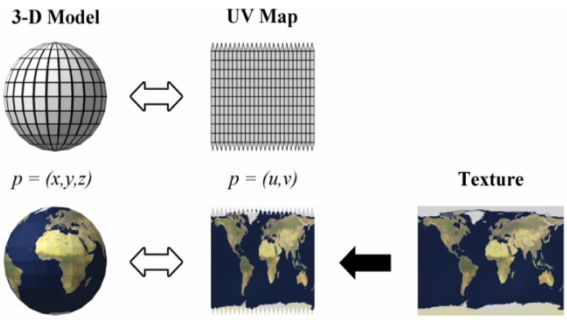
```
glm::mat4 earthSelfRotation = glm::rotate(glm::mat4(1.0f), time * earthRotationSpeed, glm::vec3(0.0f, 1.0f, 0.0f));
glm::mat4 earthOrbitRotation = glm::rotate(glm::mat4(1.0f), time * earthOrbitSpeed, glm::vec3(0.0f, 1.0f, 0.0f));
glm::mat4 earthTilt = glm::rotate(glm::mat4(1.0f), glm::radians(23.5f), glm::vec3(1.0f, 0.0f, 0.0f));
g_earth = earthOrbitRotation * earthSelfRotation * earthTilt * earthMesh->getMeshMatrix();
```

For the animation, I used the glfwGetTime() function, the calculated the mat4 as needed for each transformation.

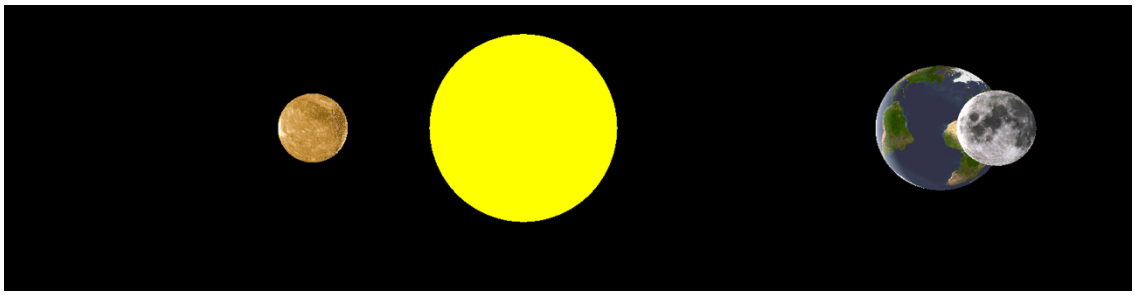
I lost the screenshots of the simple green, blue and yellow colors, so here is straight away the screenshot with the textures.



I mapped the textures with a UV mapping.



Then I changed the background color to black, to make it look more like space. Initially I wanted to add a square in the background and then use a star sky texture, but I didn't work as I wish it would. So I gave up this idea, and instead added another planet : Mercury, smaller and closer to the Sun, and rotation faster than the earth around the Sun.



I then added options to the camera :

```
public:
void zoomIn(){
  this->m_fov-=1.0f;
}
void zoomOut(){
  this->m_fov+=1.0f;
}
```

```
} else if(action == GLFW_PRESS && key == GLFW_KEY_O) {
  g_camera.zoomIn();
} else if(action == GLFW_PRESS && key == GLFW_KEY_P) {
  g_camera.zoomOut();
```

Zoom in or out, but pression O / P, it works by adjusting the field of view of the camera.