Computer Graphics – Lab Session 2

Introduction
The goal of this lab session is to help you understand the following notions: Shading, Per-pixel effects, Scene Graphs, Render to Texture and Post-processing. To do so, we will create and animate a solar system with Earth, Moon and Sun .

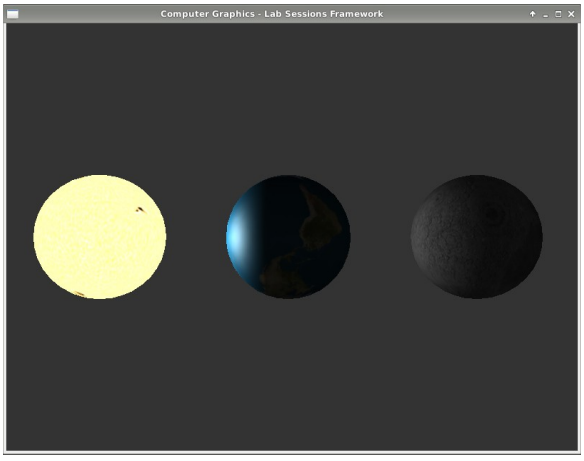
You first need to download the “LabSession2Files.zip” archive with updated files, .obj models and textures from the unit website. Extract it and add/update the files in the the LabFrameworkLinux64 project folder. You will see a new templateFrameworkRTT.cpp . This is going to be your new main file. Exclude your templateFramework.cpp from the build (Properties->C/C++ build->Exclude resource from build).

Exercise 1 : Phong Shading and Texture
The purpose of this exercise is to help you understand Phong Shading and texture mapping.

a) Create three new objects of the ShadedOBJObjectPointLight class: earth, moon and sun (Look at how objectA and objectB are created and used in templateFrameworkRTT.cpp)
b) Call the constructors using sphere.obj as the geometry and the corresponding texture (.bmp file)
c) Set their positions so that the sun is at the center and the earth and moon on each side (we will adjust their positions and scales later)
d) Compile and run, you should see the three planets, but the shading is the same for the three of them

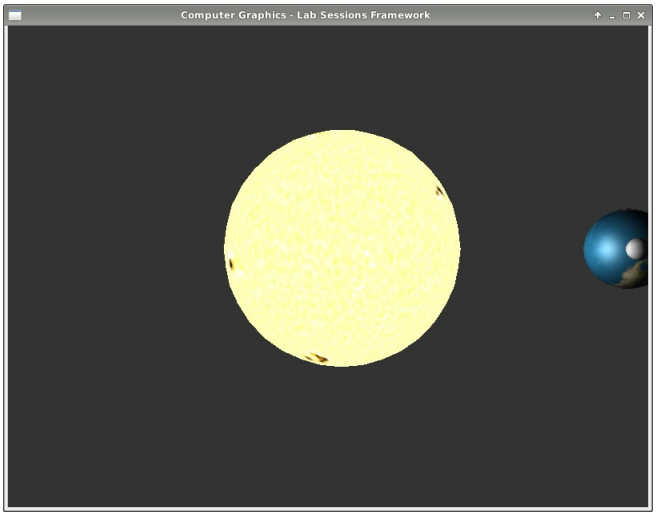
You are now going to correct the shading, by adjusting it to the “properties” of each planet.
-The Earth should have specular reflections only on the oceans
-The Moon should not have any specular reflection
-The Sun should always be lit
Observe how the Phong Shading and Texture Mapping are implemented in PointLightShading.vertexshader and PointLightShading.fragmentshader. Pay special attention to the attributes passed between both, how they are intepolated between the different fragments of the primitives and how the material color is retrieved from the texture.

e) In order to differentiate between the three planets in the fragment shader, you are going to add a “planet” uniform in the shader and set it in the C++ class (allocateOpenGLResources) depending on the texture name.
f) Adjust the shading for each planet in the Fragment Shader depending on the value of the uniform.
g) Finally find where the light position is defined and place the light at 0,0,0 in world coordinates (where the sun is)
You should now have the three planets with textures and appropriate shading.



Exercise 2 : Fragment Shader Manipulation
The purpose of this exercise is to show you how you can interactively manipulate the Fragment Shader by creating a movement of the surface of the Sun.
a) As done in the first lab session (when we animated the morphing pyramid), create a "time" uniform in the C++ class, update it in the Render method and pass it to the Fragment Shader of the planets.
b) For the Sun, use this time to animate the texture over time (you can use the sin function for example to make it oscillate)

Exercise 3 : Scene Graph
The goal of this exercise is to help you understand and use SceneGraphs. Look at the slides in Lecture 2 for more details on them.
a) Look at the SceneNode.h class in the content folder. How does it work ?
b) Starting from a root Node of the Scene, model the organisation of our simplified solar system (Earth orbiting around the sun and rotating around its center, Moon orbiting around the Earth)
c) Using this class and the classes and shaders you have created in the previous exercises, build the scene graph and set the transformations of the nodes to replicate the solar system (You will need to first remove all the calls to the earth, sun and moon objects in templateFrameworkRTT.cpp)
d) Animate the solar system by transforming the nodes (setting their model matrices) in the renderWorldScene function of templateFrameworkRTT



Exercise 4 : Render To Texture – Virtual monitor
The purpose of this exercise is to help you understand Render to Texture.
To that extent, you are going to implement a virtual monitor placed in the scene that provides a view of the solar system from above. First observe how the scene is rendered twice in the main function: once with the render to texture framebuffer object (rtt_fbo), then with the default framebuffer object. Observe also how the rtt_fbo is configured.

a) From where will the scene be rendered to a texture ?
b) The rtt_fbo renders to a texture. You can access the handle of this texture with the getRTT() method. Create the rttMonitor as a UnitPolygonTexture, passing it the rtt texture handle. Place the rttMonitor in the scene.
c) The rttMonitor is now placed in the scene. How would you make sure that it is always placed on the screen ? Do the required modifications.