Computer Graphics Lab 5

Assistant Prof. Georgios Papaioannou

Teaching Assistant Konstantinos Vardis

Dept. of Informatics, Athens University of Economics and Business

Introduction

This lab introduces an object loader and scene management using a scene graph structure

Getting started

1. Start Visual Studio and load the solution file (.sln). You should now have a window on the right (or left) hand side of your display that indicates that solution ‘lab1’ has loaded.
2. In the Solution Explorer (on your right or left) locate the Headers and Source filter. The first one contains the file Renderer.h which simply contains the definitions of the functions of Renderer.cpp we wish to expose. The Source filter contains the Main.cpp which is the main file of the application and the Renderer.cpp which contains the implementations of basic OpenGL functions. Double clicking on these files will display their contents the source code editor window.
3. The Shaders filter contains all the shader files necessary for each lab.
4. The ShaderGLSL class provides basic shader functionality (loading and compiling shaders)
5. The HelpLib files contain basic functions and includes in order to make your life easier :).
6. The SceneGraph filter contain the scene graph files
7. The solution produces both 32-bit and 64-bit executables.
8. **Make yourself comfortable with the code!**

Changes From Last Lab

- We will need to use more than one shaders. For this, a Shaders.h file has been created where the shaders are declared. Each shader should contain a pointer to a ShaderGLSL object which is responsible for the shader compilation and linking and the shader’s uniform variables. The shaders are created and compiled at the CreateShaders() function which is called in InitializeRenderer.
- A Lights.h file has been created which contains the declarations for each light source. Normally, the lights should be set as nodes in the scenegraph. This is a more basic approach.
- The SceneGraph has been modified so that it can allow rendering geometry with different shaders. For this, the shader and light objects are being sent to the root node in order for them to be available in the GeometryNode. The Draw function has been altered to accept a shader_type parameter which renders the geometry according to a specific shader.

- A DirectionalLight and an OmniLight shader are included which renders the geometry using a directional and an omni light respectively.
- The **DrawOGLMesh** function in Renderer.cpp is responsible for rendering an OGLMesh by using the BasicGeometry shader.

Key Input

- **ESC**: exit the application
- **F1**: change between rendering modes (fill, wireframe and point rendering)
- **W/S**: camera eye Z translation
- **A/S**: camera eye X translation
- **R/F**: camera eye Y translation
- **UP/DOWN**: world Z translation
- **LEFT/RIGHT**: world X translation
- **PGUP/PGDOWN**: world Y translation

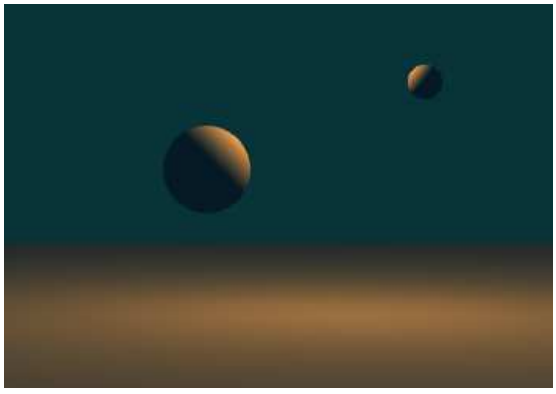
Exercises (Follow these in the order given!!!!!)

1. Look at the code and familiarize yourself with the changes.
2. Uncomment the **BlendingExampleDraw** function and run the project. You should see an opaque red sphere. Uncomment the part of the code for sphere 2 (the blue one). This is also opaque so the result is as expected – the blue color of the second sphere covers completely the red color. Uncomment spheres 3 and 4. These are rendered with blending enabled based on their alpha value (which comes from the material files). Correct blending expects the semi-transparent objects to be rendered in back to front order. If you move closer you will see that the colors between spheres 3 and 4 are not blended correctly. This is since the back to front order is not preserved (sphere 3 is further than sphere 4). To blend the colors correctly, switch the rendering order between spheres 3 and 4.
3. You can change the alpha value for the spheres by changing the **d** parameter in the .mtl file of each sphere (you can open it with a text editor). You can also change the alpha value by changing the final value in the BasicGeometry fragment shader which affects **all** objects. Once you are done, comment the **BlendingExampleDraw**.
4. Uncomment th**e SceneGraphExampleInit** an**d SceneGraphExampleDraw** functions and run the projects. You will see two spheres and a plane. Currently, the root->draw accepts parameter 0. This means that the geometry will be rendered using a directional light. The directional light that is used is called **sunlight** and is created in the **InitializeRenderer** function. Change the initial parameters (such as color, initial_direction to change the intensity and the location of the light).
5. Set the initial direction of the **sunlight** to -1,1,0,0 (so that it comes from the top left). Run the project.
6. Set a rotation transformation to see how the directional light illuminates the scene as it rotates. You can do this by uncommenting the part which does the sunlight rotation

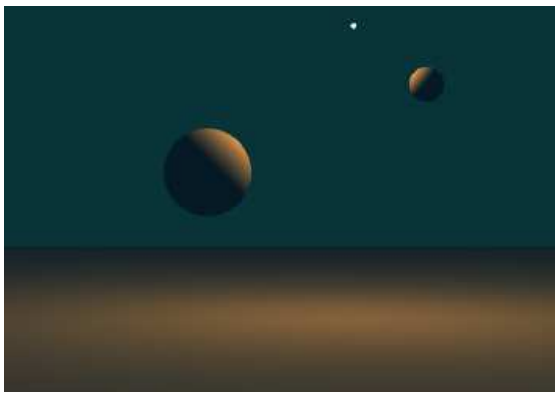
transformation in the Render function. Run the project. What you see should be similar to how a very distant light source (such as the sun) illuminates a scene.

7. Look at the **DirectionalLight** shader and check how the shading equation for directional lights is implemented. Currently, only the diffuse part of the shading equation is implemented since all the non illuminated areas are black. Add some ambient lighting by adding an ambient RGB (vec3) value in the fragment shader. The ambient color should be added to the RGB value of the diffuse color. In general, the ambient value should be low but this is according to the environment. Use a blueish ambient color, such as 0.02, 0.1, 0.15 **(check the slides to remember how the ambient color is applied)**. Also set the background color to something similar to get a more visually pleasing result. Use something like glClearColor(0.08f, 0.17f, 0.2f, 1.0f); The final image should be similar to this:

1. Set the initial position of the **candlelight** to 10,10,0,1 (so that it is placed at the top right). Set the root->draw parameter to 1 (root->draw(1)) to render using the omni-directional shader.
2. Set a rotation transformation to see how the omni light illuminates the scene as it rotates. You can do this by transforming the position of the candlelight in the Render function (similar to the directional light). Run the project. What you see should be similar to how a local point light source with no direction (such as a candle) illuminates a scene.
3. Look at the **OmniLight** shader and check how the shading equation for omni lights is implemented. Currently, only the diffuse part of the shading equation is implemented since all the non illuminated areas are black. Add some ambient lighting by adding an ambient RGB (vec3) value in the fragment shader. The ambient color should be added to the RGB value of the diffuse color. In general, the ambient value should be low but this is according to the environment. Use a blueish ambient color, such as 0.02, 0.1, 0.15 **(check the slides to remember how the ambient color is applied)**. If you haven't already from step 7, set the background color to something similar to get a more visually pleasing result. Use something like glClearColor(0.08f, 0.17f, 0.2f, 1.0f); The final image should be similar to this:



4. Omni lights attenuate with distance. The **OmniLight** fragment shader has an attenuation variable set to 1.0. Different attenuation functions can be used. Enable linear and quadratic attenuation for the omnilight.
5. Use the spherewhitemesh OGLMesh to render the light source, so that you can see where it is located. After all, omni lights have a position. Since there is no light node, you need to do this outside the scene graph, as you did before the fourth lab. To do this, you need to create the necessary transformations that are applied to the omnilight for the lighting calculations and pass them as an object_to_world transformation matrix in the **DrawOGLMesh** function. If the light source is too big, you can apply a scaling transformation as well.



6. Comment th**e SceneGraphExampleInit** an**d SceneGraphExampleDraw** functions and uncomment the **SceneGraphExample2Init** and **SceneGraphExample2Draw**. Position the omni and directional lights in a similar way (the omni light should be positioned higher since the units of the second scene are different).