Assignment 1 – SceneGraph Framework

The purpose of this exercise is to try out using a scene graph and to develop a small game using OpenGL ES 2.0 as if it was running on an iPhone.

1. Getting started

Download the framework code and open AstroidGame.cpp to see how a minimal setup of the scenegraph looks. It just has a World node (with a standard rendering state). The project should compile, but won't run properly since there is no camera yet. Set up a Camera node and attach it to the World node. Set the projection of the camera to perspective with 90 (pi / 2) degree field of view, 0.8 aspect ratio, 1.0 near plane and 1000.0 far plane. Don't forget to set the camera active on the world node. Run the code.

You'll get a black screen in the iPhone simulator again, but without the error message. The simulator has a memory monitor to the left. It'll help you keep track of your allocations.

Press TAB to switch orientation of the phone and ESC to exit the simulator.

2. Building a scene graph

Write a function (`TriangleMesh* AsteroidGame::createYourMesh()`) that creates a simple TriangleMesh object (for instance a box). To do this you must create:

- **VertexArray:** Contains all the vertex attributes of the mesh. You'll need to add two attributes - “Vertex” (position of vertices, 3 floats) and “Color” (color of vertices, 4 floats) .
- **IndexArray:** An index array contains indices to vertices.
- **Material:** Set up a “MaterialColorful” material to attach to the mesh.

Create a scene, with at least 2 objects, that shows some form of hierarchical animation.

Scene graph data can also be read from a file into a Group node. The files sphere.pwn and spaceship.pwn contains a sphere mesh and a space ship mesh. Load one of these files into a Group node by calling sceneGraph.createGroup(). Since these files only contains one mesh each, you can extract the TriangleMesh by fetching the first child of your node. Replace your object with one of these loaded meshes.

Create and apply a new material “MaterialPhong” to the mesh. Create a Texture resource and attach it to the material's diffuse texture slot. You'll also need to create a Light node and connect it to the mesh (note: not the same as attaching the node as a child node). Your object should now look awesome.

Tip: calling duplicate() on the mesh object is more efficient (in terms of memory and speed) than creating a mesh twice.

3. A simple game

You are now going to make a simple asteroid game. You have been given a few files. Asteroid.cpp and Player.cpp describe an asteroid and a space ship controlled by the player.

Complete the following functions in Asteroid.cpp:

- **randomState:** Complete this function so that it gives the asteroid a random position, random velocity along the z-axis (towards the camera), random rotation axis & velocity, and a random scale. Note that the asteroids should come flying from a far distance towards the player (camera), so the random values must be given in reasonable intervals**.**
- **update:** Complete this function so that it updates the position by moving the asteroid based on the velocity, and rotates it based on the angular velocity. randomState should be called if the

asteroid has passed the camera (which is still at position 0,0,0 if you have not changed it from the exercise 3)**.**

You should also complete the following function in Player.cpp:

- **update:** Complete this function so that it translates the player mesh according to the current ship velocity. The ship only has a velocity in the x and y axes. The velocities are controlled by user input. The input state can be read from the DeviceState_t-struct that is passed in as an in-parameter to RCUpdate().

You must also update the constructors in AstroidGame.cpp, Astroid.cpp and Player.cpp to set up everything up properly.

Tip: You can use the "getTranslation()" function to get the current transform attributes of the Transformable object representing an asteroid, or the player.

4. Playing around with Shaders

Open the MaterialPhong.cpp in the RenderChimp framework. Look for the comment `/* Uncomment me! */` and uncomment the line that follows it. Now open Material.vs and Material.fs and add a uniform float called Time. We now have access to the current time (in seconds) in the Material-shaders. Now, be inventive and use it to achieve a time-dependent effect of your choice.

To pass this assignment you must be ready to show and explain your code.

Extensions

If you have time you could extend the asteroid game with collision detection, shooting, a scoreboard or something else. Use your imagination!