Assignment 3: Modeling and Scene Setting

Out: 10th October 2017

Due: 18th October 2017 at 8:59pm

You will work in pairs for this assignment. Both partners will get the same grade, so please make sure everybody contributes equally.

In this assignment you will interact with a scene graph implementation. Several scene graph models have been provided to you in XML files. A program that should read these models and render them is also provided to you. In this assignment, you will learn how to model an object hierarchically using a scene graph. You will then create a 3D world. Finally you will implement a keyboard-controlled camera for the scene.

Overall Design of World

In the next three assignments, you will create an interactive 3D program that has the following aspects:

1. (This assignment) Model a 3D virtual world using hierarchical objects in a scene graph.
2. (Future assignment) Some of your models will be animated.
3. (This assignment) A keyboard-controlled camera that navigates in your virtual world.
4. (Future assignment) An object-stationary camera.
5. (Future assignment) Lighting and texturing, with at least one moving light.

Starting code

A Scenegraphs project has been provided to you. It uses an inbuilt parser to read a scene graph as an XML file.

Javadoc-style documentation has been provided with the project and in each class to help you understand the structure and design of the program. I recommend you step through the program in IntelliJ/Qt Creator to understand how it is working. It is a good way to quickly understand how various parts of the program work.

Several scene graph models have been provided to you that show various features of this program.

1. Face-hierarchy.txt shows a simple Jack-in-the-box model as a scene graph. This is probably the smallest and simplest example.
2. Humanoid.xml draws a simple humanoid model.
3. Two-humanoids.xml shows you how to refer to external scene graph files from within a scene graph.

Part 1: Model the Y-M-C-A poses (20 points)

Start with the humanoid model, and model the “Y”, “M”, “C”, “A” poses from the dance steps to the popular Y-M-C-A song. Go to https://en.wikipedia.org/wiki/Y.M.C.A._(song)#/media/File:YMCA_dance.jpg to see a visual of these poses. Save each pose in separate files: humanoid-Y.xml, and so on.

Create a new virtual model with the four humanoids in these positions standing next to each other. When the program is run to view them, all four poses should be visible clearly.

Part 2: Floor of a building (50 points)

In this part, you will create the beginnings of your “scene”.

This scene will represent at least one floor of a building. The floor may represent a house (residential) or office space. You are free to create a model as you wish, so long as your model has the following constraints:

1. The scene must have at least three rooms.
2. The rooms must be such that it is possible to reach any room from any other. It may not be possible to walk directly from one room to another, but they should be connected in some way. The entries to each room should be doorways. Modeling the actual door is optional.
3. Each room must have at least one window.
4. One out of the three rooms should have the table scene that you modeled (pick any table scene from the group, the other one will have to be incorporated in the next assignment), placed suitably.

Be sure to use the following features of the XML file that will help you model as well as animate:

a. Build the model "bottom up", like we built the fan example in class. Start with the smaller parts. Then move them to their correct positions by adding them to transform nodes, etc. Build this interactively using your program, rather than typing in the entire file and then viewing it!
b. Use descriptive names for nodes appropriately, using the "name=<value>" attribute to any node.
c. Keep parts of the scene in different XML files and assemble the scene in other files, instead of trying to type in the entire scene in one (gigantic) file! Look at "two-humanoids.xml" to see how to do this. **Warning: when the program puts one scene graph as part of a bigger one, it pre-pends all the names of the nodes in the scene graph with the name of the group that is used to import the scene graph.**

This part should not require any Java coding! All you have to do is type an XML file in your favorite text editor and load it in the given program.

Part 2C: Camera (15 points)

For this part, introduce a camera in the scene that can be controlled using keyboard keys. The following keys should work:

1. Left/right arrow keys turn the camera without changing its location (nodding "no").
2. Up/down arrow keys move the camera forward/behind, in the direction it is currently looking.
3. Any other two keys to turn the camera up/down without changing its location (nodding "yes").

Part 3: Understanding the design (15 points)

For this part, please read and understand the design of the code in the "sgraph" namespace/package, that encloses all classes relevant to the scene graph. You must prepare a short write-up that explains the design so that anybody who wants to use this code gets a relatively quick overview of how it is structured.

Make sure your explanation answers the following questions:

1. What is a scene graph made of? What are its various parts and what is the function of each part?
2. How is the scene graph drawn, and how does it ensure the correct transformation gets applied to the correct part?
3. What is the use of the GL3ScenegraphRenderer class? If you wanted to create a textual rendering of the scene graph (e.g. a textual description of each node) how would you write such a renderer?

Preparing program for submission

1. Model your scene so that:
    a. By default, your room scene is visible, with the camera set to look directly at one of the table models inside one room. The camera should respond to the keyboard keys.

b. Pressing “Y” will switch to the YMCA scene with the camera at the appropriate position (camera may or may not respond to keyboard keys in this scene). Pressing “R” should switch the program to the room scene as described in 2a above.

2. Keep your XML files in a separate folder, as organized in the code provided to you.

What to submit

Submit your entire IntelliJ/Qt Creator project and your documentation in as a single zipped file on Blackboard. Only one submission per group is expected.