Assignment 3: (Algorithmic) Modeling and Scene Setting

Out: 8th February 2019

Due: 19th February 2019 at 11:59pm

You will work in pairs for this and all subsequent assignments. 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.

Overall Design of World

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

1. (This, future assignments) Model a 3D virtual world using hierarchical objects in a scene graph.
2. (Future assignment) Some of your models will be animated.
3. (Future 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. It also allows you to transform the scene graph using a mouse-based trackball.

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.

This Assignment

As you read through this assignment, you will notice that it actually involves very little of graphics programming. Most of the work will be done outside a conventional OpenGL-type program. This is by design: this assignment is mostly about creating models, which can be rendered using a program provided to you. The only "graphics-specific" programming may be to set up the cameras correctly to view your models.

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: Model of a building (50 points)

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

In this part, you are required to create the model of a building exterior. The building should have an urban look (e.g. West Village H, Prudential Tower, etc.). Specifically, the building should have the following properties:

1. It should look like it has at least three floors.
2. It must have a roof, which may be flat. That is, the building cannot simply be a "hollow and open" column.
3. It must have at least 5 windows. Windows can be rectangular. At least some windows must be "see-through", i.e. it must be a hole in the wall so that the inside of the building, or the other side of the building is visible through it.
4. At least one window may be recessed (i.e. it is not see through, but has a sill). For example, how most buildings look like from the outside when all the windows are shut.
5. Its walls should not be "laminar". That is, each wall should have a thickness rather than just a plane of triangles (another way to think of a wall is a thin cube).
6. The building must rest on a rectangular ground, instead of hovering in space.

You may assume that the user will never navigate inside a building (or if the user does, will not expect the interior to look interesting). In other words, you are expected to confine yourself to designing the exterior of the building, not the interior.

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.**

How to build urban buildings

Urban buildings have a "uniform" structure (there is a reason Manhattan looks boring!). Many buildings are box-shaped, with windows arranged in a grid-like pattern on its façade. Even the details of each window (e.g. the sills) follow a pattern.

This implies that its structure can be built "algorithmically". You may think of writing a program that uses loops and/or recursion to "generate" the XML text for a scene graph for that building, instead of attempting to manually type (or copy-paste) an XML file in a text editor.

Algorithmic generation has another major advantage. For example, you can change the color of a building (or a part) by changing it in code and running it again to re-generate the XML file. This will be much faster than editing (what will likely be huge) XML files.

How to build windows

A window in a wall may be simpler than you think. Instead of thinking about one box for the wall and wondering how to "cut" it, you can model the wall to be 4 boxes, with a gap between them.

Given a box for a wall and a box for a window, it is possible to write code that returns the resulting set of boxes (work this out on paper, it is not unlike intersecting rectangles).

Cameras

You must support two kinds of camera views in your program:

1. Global stationary camera: choose camera arguments that will show the entire loaded model in view, from a suitable perspective.
2. Turntable camera: in this the camera should hover in a circular orbit above the model, showing the model from all sides. The models should not move (i.e. move the camera, do not transform the models).

See the section below on how to set up your program for submission.

Part 3: Understanding the design (10 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 (in the code, not in general)?
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 (10 points)

1. The program accepts a single command-line argument. This argument is the path of an "input configuration" text file. The format of the text file is completely up to you. However it should contain the path of the XML file to be used by your program, and the camera position and orientation it should use.
2. When a correct file is provided, it should run the program to show that model with the appropriate camera position. This should allow us to see the output of your program for the YMCA model and the scene with two buildings by simply using two text files provided by you (and other files necessary to show the scene). We should not have to change any text files for this to work: make sure you have prepared and submitted all files correctly!
3. Pressing the "T" key should switch to the turntable camera. Pressing the "G" key should switch to the "global" stationary camera.

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.

Postscript: Graphics Trivia

Algorithmic generation of architectural models is an exciting and continuing area of research. In its simplest form it usually works as a set of grammar symbols and rules of expansion (Start with F, now apply the rule F -> FG and G->FG several times to get a resulting string. Finally map F and G to some geometric objects or transformations on them).

We saw one such example in the first week of class. Look up “Instant Architecture”, which was one of the first papers in this area. While you are not expected to implement any of this in this course, using loops/recursion to generate a model (the hint above) is a simple way to implement this technique!