Scene Graph

CIS 460/560 Homework 2

Due Wednesday, October 16, 2013 at 11:59 am

1 Goal

This assignment's purpose is to gain experience in creating a data structure to store a three-dimensional, hierarchical scene based on a text file description. You will then render the data structure's contents using OpenGL.

2 Bigger Picture

Moving on from clouds, you will be making a room editor that arranges furniture and in the process get some practice with OpenGL 3.2. This furniture will be built up of primitives, namely boxes, spheres, and cylinders. In the following assignment you will add onto the items placed in the room with procedurally generated meshes. The final stage in this chain of assignments will result in a raytracer which turns your OpenGL rendering into a sort of "preview" before you commit to a high-quality rendering.

3 Software Environment

Your code must be able to be compiled and run on the computers in the Moore 100 Windows labs without modification. An OpenGL 3.2 framework for Qt and freeglut is available on Canvas and assumes you have the proper drivers installed on your own computer and the ability to use the proper OpenGL functionality. If your computer cannot support the necessary OpenGL functionality you will have to use the Moore 100 labs. The framework will include simple shaders and a linear algebra library. If you have a Linux/Apple/Unix computer, you may try to use Qt Creator or pursue freeglut for your operating system but your submission must compile using Visual Studio in Moore 100, so plan accordingly as it takes time to transfer a Qt Creator solution to a Visual Studio solution.

4 Supplied Code

- Two versions of the same graphics codebase will be available on Canvas, one for Qt and one for freeglut. The code contains examples of techniques 1

for drawing in OpenGL using Vertex Buffer Objects and starter shaders. Neither the C++ source code nor shader code written in GLSL 1.5.0 use deprecated functionality. The OpenGL API calls will be the same regardless of whether you use Qt or freeglut as a framework.

- A folder containing GLM 0.9.2.6 is provided. This is an open source linear algebra library that was written to allow C++ vectors and matrices to be treated like GLSL vectors and matrices. Example usage is in the starter code, no linking against libraries is required. This folder is in both starter kits and includes a folder with documentation.

- GLEW 1.7.0 set to compile in static mode is included in both versions. This means you just need to include all GLEW code files in your source code folder and include glew.c/h in your solution to use it. The change to compile as static is in the glew.h header, so if you download it yourself from the internet it won't be the default.

- freeglut is, obviously, only incuded in the glut-based starter kit. It requires a dll be in the working directory whenever you run, and comes packaged that way for running in Visual Studio. It also requires edits to the linker settings, which are already done for you but be aware of it if you use it on your own.

5 Requirements

5.1 Scene Graph (30 points)

A scene graph is used to organize geometry in graphics. While scene graphs can be implemented various ways, they all store persistent information from frame to frame. A node in the scene graph should consist of a link to the parent node, links to children nodes, and one link to geometry. In this assignment your geometry is furniture. Any of these links could be NULL if not used, i.e. the root's parent is NULL. The node will also store translation in x- and z-dimensions, rotation about the node's y-axis, and scaling in all 3 dimensions. Translation in the ydimension should be handled by the graph's hierarchical structure. Your scene graph should only be built once. If you construct your scene graph in each frame of your rendering, you've done it wrong. Your program does not need to support more than one scene graph.

5.2 Handling Transformations (20 points)

Scene graphs are recursively processed by preorder traversal. Use scoping to push/pop the transformation matrix. If you pass-by-copy, then the recursive calls' scoping will automatically pop off the changes to the matrix. Use the linear algebra library's matrix class to store the transformation matrices. Don't forget to make sure your modelview matrix starts as the identity matrix every

frame and remember to pass any updated matrices along to the shaders with glUniform before rendering.

5.3 Shaders (10 points)

Part of a Lambertian (or, diffuse) shader is provided in your starter kit. The vertex and fragment shaders are complete enough to be functional, but could still use some work. Do the following:

- Add a uniform for light position and replace the hard coded light position in the vertex shader. Set the light position somewhere in your C++ code to a value of your choice that makes sense.
- Add an ambient lighting term to the fragment shader. This should not require additional shader input.

5.4 Furniture (25 points)

You need to create a chair, table, filing cabinet, and lamp. You should use a box, sphere, and cylinder at least once between all these pieces of furniture; these are considered your geometric primitives. We are not looking for impressive artistic masterpieces, so don't worry about perfection with your modeling. Each piece of furniture should be instantiated only once by your scene graph, but can be linked to via pointers any number of times. The scene graph will use the same geometry but cause it to appear in many places. For example, a classroom with a table and 30 chairs can be made with only one instance each of a table and chair.

It will be easiest in future assignments if you create your scene graph and furniture such that each piece of furniture is itself a small set of primitives. An example of a chair from many years past will be posted on Canvas, but will need updating to bring it into OpenGL 3.2, it's just to give you an idea of dimensions and how simple the furniture can be.

5.4.1 Floor

The root node's link to geometry should always be a floor. This can simply be a stretched and thin box.

Storing Furniture

It is probably easiest to make a class for each kind of furniture, and inside the class store arrays for the position, color, and normal of each vertex. Then when the furniture needs to draw, buffer the data to the proper VBOs and use glDrawElements. Skip deleting the array to save time, but make sure the array has sufficient scope to not cause a memory leak. Further optimization can be found in the extra credit.

Creating Primitives

A box should be straightforward enough to create out of vertices. A unit size cylinder and sphere also need to be defined for building some furniture. There are countless references to approximation of spheres online, and your textbook has some information on this concept as well. Once you can make a sphere, a cylinder is a closely related concept. Use an IBO to index vertices in such a way that you render with GL_TRIANGLES at all times.

5.5 File Input (10 points)

You will be given 2 example files of the following format. These files will be parsed by your program when it starts in order to construct the initial scene graph. The files are of the following format: the first line consists of three integers:

xSize zSize numItems

xSize and zSize establish the size of a discretized grid to help organize the scene and suggest the size of the floor. numItems lets you know how many pieces of furniture will be added to this grid. Thus there are numItems of the following form:

furnitureType
red green blue
xIndex zIndex
rotation
xScale yScale zScale

furnitureType will be a string, either "chair" "table" "cabinet" or "lamp" (no quotes).
red, green, and blue are floats and are optional for you to support, and define the furniture's color if you support it.
xIndex and zIndex are integers that describe where in the discretized grid the furniture should go. If there is already a piece of furniture at that location, this new piece of furniture should be visually stacked on top of it and become a child of it in your scene graph. This can continue any number of times. rotation is a float that defines the y-axis rotation in degrees for this node around the node's origin.
xScale, yScale, and zScale are floats and define the respective amount of scaling in each dimension. Scaling does not have to be uniform. You are NOT responsible for detecting whether or not two nearby scaled objects end up intersecting each other. So if location 2, 3 holds a table stretched to double its width and 3, 3 is supposed to hold a chair, you do not need to stack the chair on the table.
You must take the filename from the command line.

5.6 Interactivity (5 points)

Once the file is parsed and the scene rendered, you need to provide an interface to modify the scene. The following keyboard commands should be used:

- n: select the next node in the preorder traversal. To facilitate this, the root should be automatically selected at program start. You need to visually indicate the node that is currently selected. This is easiest if you change the color of the selected object.
- a: translate the selected node (and by nature of the scene graph, its children) along the negative x-axis half a unit.
- d: translate the selected node along the positive x-axis by half a unit.
- w: translate the selected node along the positive z-axis half a unit.
- s: translate the selected node along the negative z-axis half a unit.
- x, X: increase/decrease scale in the x-axis by 0.5.
- y, Y, z, Z: Same as x, X for the remaining axes.
- r, R: increase/decrease the rotation of the selected node by 10 degrees.
- e: remove this node and all its children from the graph, then select the next node in the preorder traversal. Avoid memory leaks and make sure this doesn't break your traversal! You may reset the traversal at the root to make things easier, but future traversals should not cause crashing.

6 Extra Credit (20 points maximum)

6.1 Efficient VBOs (10 points max)

Use a more advanced approach to VBOs. For each type of furniture, store one VBO that takes care of position, normal, and color. Rather than replacing all the data everytime you draw a different piece of furniture, just switch the VBO in use.

6.2 Bounding boxes (10 points max)

Show the selected node by drawing a tight bounding box around the geometry. A tight bounding box is one that scales and rotates along with the geometry. You will have to learn how to draw lines with OpenGL on your own.

6.3 Multiple unit sized objects (20 points max)

Provide support for larger-than-unit sized objects. For example, add "multitable" to your file parser and provide a file that uses it. An example of a multitable would be a 1x2 table, so that if multitable with address 2, 3 is specified, a lamp at 3, 3 becomes a child of the multitable in the correct location and is stacked accordingly.

7 Example Usage

A grader opens your code and puts in the "test.dat" filename for his classroom scene. He compiles and runs the program, and is presented with his scene of 30 chairs, a front table, and lamps in 3 of the corners of the room. The fourth corner has a filing cabinet with a lamp on top. This is exactly what he defined, and a floor neatly fits under all the geometry. The unit sized table looks too small though, so he traverses the scene graph until he reaches it and scales along one axis and translates to recenter it. He then decides the filing cabinet would be easier to use if the door wasn't right up against the wall, so he selects it and rotates it 45 degrees, and notices the rectangular base on the lamp rotates as well. Then the grader decides the filing cabinet is tacky anyhow and deletes it entirely, also removing the lamp in the process. He then exits the program.

8 Deliverables

You will submit a zipped folder via Canvas containing your Visual Studio project and a readme file with anything you think the graders need to know. Please make note of the extra credit you implemented, or you may not get credit for it. As always, please test your code in Moore 100 before submitting!