CS535: Assignment #3 –Spatial Subdivision and Culling

Out: October 2, 2007
Back/Due: October 18, 2007

Objective:
Now that you have an understanding of the basic underlying graphics rendering pipeline, the focus of this assignment is to develop an application using OpenGL (and all the hardware supported features) to create and effectively manipulate larger 3D models. You will implement a spatial subdivision data structure and use it for various culling and rendering operations.

Summary:
The assignment is to write a program which loads and subdivides a 3D model using two spatial subdivision data structures. One of the data structures must be an "octree data structure". The other one can be one of binary-space partitioning tree, kD-tree, or another data structure which you can propose but must get approval for. Another option is to use a bounding-volume hierarchy as your second data structure. If you are not sure what a bounding-volume hierarchy is, check out a book reference, ask instructor, or don't use this option. Your program must include several visualization tools so as to verify correct functionality. There will also be a competition for who can render the largest model at the highest frame rate. Aside from clever rendering and usage of OpenGL, there are several enhancements that can be done – see text below.

Specifics:

(0) Scene file format. Several scene geometry files will be provided. These are the same as with the previous assignment. In addition, you will be given an executable which you can run and generate a model of arbitrary size – this will be used for the rendering performance competition. The program is called "makepipes" and is to be used as "makepipe sim <x> <y> <z>" which generates a connected entanglement of pipes of <x> by <y> by <z> pipe units. Try "makepipe sim 4 4 4 > pipes.sim" to get an initial model, then try increasing the size until your program cannot handle it.

(1) GLUI: the GUI must allow the user (a) to interactively highlight information about the data structures, (b) to change the recursion-stop criteria for the data structures, e.g: minimum box size and minimum triangles per box count, and cause the data structures to regenerate on the fly, (c) to freeze view frustum culling so that a bird's eye can be obtained, (d) to enable special additions to render massive models. And of course, the user should have complete camera control so as to navigate through the model using at least the same intuitive camera controls as the previous assignment (and if you had a trackball, that too).

(2) Octtree: you will have to create an octtree data structure for the object upon loading or upon changing an octtree parameter. The first box of the octtree should include the bounding box of the entire object. The tree should recursively subdivide the root box into eight subboxes until either a minimum polygon count per node is reached or the minimum box size is reached. You will have to

implement a visualization scheme such that upon a user-interface “checkbox”, the user can select to see a wireframe rendering of the octtree boxes. This will serve to visualize that if we change the minimum box size and the octtree is recomputed, we can see how the octtree boxes change. Remember the subdivision should be sparse, meaning the boxes containing less than the minimum threshold should not contain subdividing.

(3) Second spatial data structure: implement the second spatial data structure of your choosing in a similar fashion and with a similar set of visualization tools. With this second data structure you can, at will, add extra features that enable rendering very large models (more later in the competition section).

(4) View-frustum Culling: you will have to implement view-frustum culling using the octtree. This means only rendering boxes that are (conservatively) considered to be within the view frustum. To accomplish this you will have to transform each box to determine its visibility. The user interface must include a checkbox to “enable/disable” this feature. We will test your program against *large* models where the effect of disabling view-frustum culling will be very noticeable. The displayed program frame-rate should decrease appropriately. Furthermore, your GUI should support a “freeze-culling” option. This enables the culling to freeze and then the camera to change viewpoint and to see the effect of the culling from a bird’s eye perspective; in other words, by freezing the view frustum and then changing the camera position/rotation, the model should not appear complete. The triangles that were not rendered due to the view frustrum culling should not appear.

(5) Problematic polygons: when you create the octtree and/or second data structure for the object, some polygons will not be completely contained within a data structure node. Thus, when you do view-frustum culling you might get the “wrong” answer unless you do proper processing. There are various options: (a) you can split polygons so that they are perfectly contained in nodes – this unfortunately increases the number of polygons and causes “seams” between parts of the model, (b) you can render the visible boxes and the adjacent visible boxes (just in case), causing less efficient view-frustum culling or (c) you can store polygons not only in the leaf nodes but also in the higher-up nodes where they intersected more than one box. You may implement (a), (b), or (c) – you will have too, otherwise your rendering will not be correct.

Competition/Extra Credit:

In this assignment, you can gain up to 50% extra credit for supporting the efficient rendering of large models. We will test all programs on the same computer (a standard Windows PC with a reasonable graphics card) using the same window size, e.g., 1024x1024. The program with best rendering performance will be eligible for 50% extra credit. I say eligible because it must be able to render models considered “massive” – think 50 million, 100 million triangles. Special arrangements will be made for you to deliver you .sim file (probably generated with makepipe) – rendering the same model N times per frame does not count (because we can’t see the massive size and thus takes the fun out of it).

The rendering improvement is to be accomplished only by using a spatial hierarchy and alternative ways of rendering it. The octtree data structure must comply with assignment requirements. However you make arbitrary extensions to it (although must be the grading requirements indicated above) or even more so to the second data structure in order to achieve the rendering improvement. I suggest thinking about visibility culling, occlusion culling, and drastic simplifications (e.g., are these situations when you can just replace an octree subtree with a single-colored box?). Also recall that the screen only has 1 million pixels or so, so rendering more than 1 million triangles means that they must be on average less than 1 pixel in size. Hmmm…

Extra points are given to creativity and to performance. Have fun!

Important: to receive the extra credit, the usage of and a summary of your enhancements must be provided and clearly specified, for example via a short document.

Grading:
Your program will be tested against all the provided input files and some other ones too. We will use the interface to alter the aforementioned parameters and expect to see the correct behavior. If the interface does not allow a certain parameter to be changed, it will be considered not implemented at all. If the program does not compile, zero points will be given.

To give in the assignment, email the TA, as with the previous assignment, by the due date and time. It is your responsibility to make sure the email is delivered/dated before it is due. **Hint: don’t wait until the last moment to hand in the assignment**.