

## CS535: Assignment #3 – Rendering using Spatial Subdivisions

**Out:** September 29, 2005

**Back/Due:** October 20, 2005

### 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 manipulate a higher computer graphics application. You will implement a spatial subdivision data structure and use it for various rendering operations. The data structure you create will be reusable for later work.

### Summary:

The assignment is to write a program which renders a solid, shaded 3D object bouncing within a box-shaped area. But this time, you have free reign of OpenGL, GLUT and GLUI to implement the program. The object now will be more complex (since you have hardware supported rendering) and sample objects will be provided. For each object, you must, upon program initialization, create an octree data structure (a spatial subdivision data structure explained in class). You will then use this octree data structure to perform view-frustum culling (e.g., only the subsets of the octree that are visible are to be sent to the graphics pipeline). Since the object is moving within the world box, the octree data structure must “move with” the object.

The user must be able to control the following: (1) the viewpoint and view direction (using either sliders in the user-interface or the GLUI-provided translation and rotation widgets), (2) enable/disable visualizing the octree boxes, (3) enabling disabling view-frustum culling, (3) changing the minimum octree box size.

The scene is being constantly redrawn as the object bounces within the world box. Changes to any of the above parameters should be effective in the next drawn frame.

### Specifics:

(0) Object Representation: each object will be in a format called “.sim”. This is a text format and is the following:

- |                             |  |
|-----------------------------|--|
| a. texture 0                | no. of textures; in this assignment always 0 |
| b. polygon <p>              | no. polygons (P=1, 1024, 45234, etc)         |
| c. <nv> -1 <flags>          | nv = num vertices, flags = data flags        |
| d. <x> <y> <z> <add'l data> | polygon vertex data                          |
| e. ...                      |  |
| f. ...                      |  |
| g. polygon <p>              |  |
| h. ...                      |  |
| i. ...                      |  |
| j. ...                      |  |

The data flags can be NORMALS, RGB, or both. If the flags includes normals, the first 3 numbers after the vertex (x,y,z) will be the normalized normal direction (nx, ny, nz). If the flags includes colors, the first or second (if normals are provided) will be the RGB colors (r, g, b) in the range [0,1] for each component. The number of polygons can be any number. The number of vertices per polygon can be any number but you can assume the polygon to be convex. An initial test object is in <http://www.cs.purdue.edu/~aliaga/tmp/smallpipes.sim>. More objects will appear there soon.

- (1) User Interface: a GLUT-based user interface must be used to allow the user to specify all the parameters listed above. Reasonable initial values should be given to all parameters at program initialization. The default viewing position should be (0,0,0) and default viewing direction should be (0,0,-1). A reasonable field of view is 60 degrees. You will have to compute the “scale of the world” by computing the bounding box of the input model and then choosing a reasonable near distance, far distance and world box size so that the object is about 10% the size of the world box and that manipulating the viewpoint within the world box does not cause the object to be clipped by the near and far plane. By default, the object should be placed in front of the camera (e.g, down the  $-z$  axis). The user interface (or the graphics window) should display a “frame-rate” (e.g., number of frames rendered per second).
- (2) Transformations: you should use OpenGL to scale, translate, projection, etc.
- (3) Clipping, Rasterization, and Shading: use OpenGL functionality; in particular, place at least one point-light source into the scene and illuminate the object using diffuse shading.
- (4) 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 one polygon is left in an octtree subbox 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.
- (5) 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.
- (6) Problematic polygons: when you create the octtree for an object, some polygons will not be completely contained within an octtree box. Thus, when you do view-frustum culling you might get the “wrong” answer unless you do proper processing. There are various options: (a) you split polygons so that they are perfectly contained in octtree boxes – 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 and (c) there is another option where you store such problematic polygons higher-up in the tree. You may implement (a) or (b).
- a. **Extra credit:** if you figure out (c), implement it, and provide a visualization (proof) of its implementation, you'll get good extra credit.
- (7) Collisions and Bouncing: similar to previously assignments, the object must bounce off the inside walls of the world environment but this time it is a 3D box and a 3D object. You must use a random (but reasonable) initial translational velocity vector and speed (such that each time you start the program, bouncing starts differently). Objects do not need to have an angular velocity. Collision response is thus simply a reflection of the translational velocity vector. You may simply use the object bounding box to do collision detection, but as with previous assignments, the object bounding box must "touch" the wall prior to bouncing along the reflected direction: remember incident angle equals reflected angle.

**GLUI: (mandatory in this assignment)**

GLUI is a platform independent user-interface builder. There are other similar libraries out there but I recommend this one for its simplicity yet completeness. You must install this library. More information about GLUI can be found at <http://www.cs.unc.edu/~rademach/glui>.

**Grading:**

Your program will be tested against the three object files you provide and additional object files we provide. 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, Denny Wong ([wang124@purdue.edu](mailto:wang124@purdue.edu)), your complete project (project files, source code and precompiled executable) by 4:30pm on the due date. It is your responsibility to make sure the email is delivered/dated before it is due. If you wish to receive confirmation of receipt, please ask the TA by email. If you email at 4:29:59pm on the due date and ask for confirmation and the confirmation is negative -- it will be considered late. **Hint: don't wait until the last moment to hand in the assignment.**

**IMPORTANT:** in your email message, please include a short message saying "This is Assignment #X" and attach your assignment inside a .ZIP file -- do not email a .EXE file as spam filters will probably filter-out the attachment.

If you have more questions, please see myself or the TA.