CS350 Project 3 Due: Oct 23

Synopsis

Implement Ray Casting with speed enhancement provided by a KD-tree built using the **Surface Area Heuristic** algorithm as described in class.

Instructions

Use the framework you have from the previous project (the BSPTree). As you extract the polygons from the scene, split each into triangles (by building triangles to vertices 0, 1,2, and 0,2,3, and 0,3,4, and so on). Place all the triangles in a KD tree built according to the Surface Area Heuristic algorithm described in class. Use the incremental sweep version of the algorithm so as to avoid the $O(N^2)$ behavior of the cheaper algorithm.

Calculate, and display on the screen, the percentage of triangle/ray intersections you evaluate compared to the total number that would have been calculated had there been no spatial data structure to enhance the calculation.

Notes

- The paper covered in class provides three algorithms for building the “best” Kd-tree. If you choose the $O(N^2)$ algorithm (the first), expect to lose most of the Kd-tree efficiency points.
- You can reverse map the pixels to world coordinates with the following OpenGL calls. See my web page for on-line OpenGL reference pages.
  glGetDoublev(GL_PROJECTION_MATRIX, ...);
  glGetDoublev(GL_MODELVIEW_MATRIX, ...);
  glGetIntegerv(GL_VIEWPORT, ...);
  gluUnProject(...);
  However, this is **too slow** to be used for each pixel. Instead, use it to convert a basis for the screen space to a basis for world coordinates. Then write each pixel as a linear combination of the screen basis, and calculate the world space pixel as the same linear combination of the world space basis. **Note:** If you don't do this, expect to lose most of the ray-tracing efficiency points.
- When you draw a pixel on the screen, all of OpenGL's transformation machinery is in your way, However, the world space point on the ray being traced is exactly the right point to draw through all that machinery to get the pixel drawn. So use "**glVertex3fv(p)**" to draw the pixel. Be sure to surround all your pixel drawing with "**glBegin(GL_POINTS**)" and "**glEnd()**" calls
- You can even use OpenGL to calculate the color of a pixel using all its usual Phong machinery. Just precede each **glVertex** call with calls to set the object's color and normal. (See the manual or examples in the framework for help in using **glNormal3fv**, and **glMaterialfv** and **glMateriali**). Make sure to **glEnable(GL_LIGHTING)** first.
- Suggested order of development: (1) Generate and draw each ray (expect to see a filled screen), (2) test every ray with **every** triangle, and draw the front most (expecting to see the correct picture drawn very slowly), and (3) finally implement the KD-tree (expecting to see the calculation speed up considerably.)

What to hand in

Use Moodle to submit a single zip file of your project.

- Place your project report in a text file named **report.txt** (or **report.doc** or **report.odt** if you (unnecessarily) insist on producing a formated document).
- Create a zip file containing only your report, and your source code ***.cpp, *.h** and ***.vcproj** files ***and nothing else.***
- Upload the zip file through Moodle using the “CS350 Project 3” link on the course web page.

Grading Basis

Grading will be on a 100 point basis, with the following distribution:

- **Kd-tree:** 50% based on accuracy and efficiency of Kd-tree building and use
- **Ray-tracing:** 30%, accuracy and efficiency of the Ray-tracing
- **Code:** 10%, based on examining the code
- **Project report:** 10%, based on succinctness and thoroughness.