CS541 Project 5

Synopsis

Implement Ray Casting with speed enhancement provided by a KD-tree built using the **incremental sweep** version of the **Surface Area Heuristic** algorithm.

Instructions

Download the new framework for this project.

- 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 $O(n \log^2 n)$ incremental sweep version of the algorithm so as to avoid the $O(n^2)$ behavior of the cheaper algorithm.
- For each pixel on the screen, generate a ray, intersect it with the scene, and draw the resulting pixel.
- 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 speed-up.

Notes

- Look in **scene.cpp** for the code that generates the scene. The scene consists of a 3x3 array of boxes (not 11x11 as stated in class).
- Look in **render.cpp** for some @TODO comments directing you where to place your code. Also find procedure **DrawSceneWithOpenGL** for an example of extracting the geometry and color information from the **Scene** data structure.
- 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 some of the Kd-tree efficiency points.
- The generation of a ray for each pixel on the screen requires transforming points via the inverse of the viewing and perspective transformations. You can achieve this with these calls:

  scene.projection.BackTransform(Vector4D)
  scene.viewing.BackTransform(Vector4D)

- When you draw a pixel on the screen, all of OpenGL's transformation machinery is active, 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 on the screen. 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**). 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 against **every** triangle, and draw the front most (expecting to see the correct picture drawn very slowly).
  3. Implement the KD-tree traversal with a dumbly build KD-tree (expecting to see the calculation speed somewhat).
  4. Finally implement the full incremental sweep algorithm to build it's idea of the “best” KD-tree (expecting to see a further speedup).

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 "CS541 Project 5" 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.