Geometric Intersection Unit Testing

CIS 460/560 Homework 4

Due Monday, November 14, 2013 at 11:59 am

1 Goal

To start your raytracer code with intersection testing. Since ray-primitive intersection tests are the backbone of the raytracer, it's important to get these right before continuing.

2 Bigger Picture

We're one assignment away from some breathtaking pictures. Raytracing involves shooting rays into the scene and testing for intersections with every primitive [or a smart set of primitives] in the scene. Before the brunt of the work that takes this from sounding straightforward to hair-pulling fun, we need to make sure we're testing the crucial step of intersections correctly. By passing these unit tests, you'll be able to know that whatever's wrong with your program in the next assignment, it's not the intersections.

3 Software Environment

We're taking a brief detour from the codebase thus far to do some unit tests of some intersection code. Then in the next assignment you'll have to return to your old code and possibly have to fix bugs or reimplement things that weren't done so well the first/second time. Your code must still compile in Moore 100 like the past assignments, but this will be straight C++.

4 Supplied Code

For this assignment, you have been provided with three pieces of base code that you will have to put into a Visual Studio project yourself:

1. GLM, the same as the linear algebra package we already gave you.
2. Intersection function stubs. Three empty functions have been provided, which you will implement to perform ray-primitive intersection tests. These

functions will take in information about a ray, a primitive (if that primitive needs extra information), and a transformation matrix. The function will return the t-value of the intersection (if any). Further documentation on these functions is provided in the header file. Not all unit tests have the answers; you must fill in the answers for those that do not, this is to make you do some intersection tests by hand for better understanding. Siggraph has some nice intersection test descriptions online, find them with your favorite search engine and convert the pseudocode to C++ for an easy A.

3. Intersection testing code. A header and source file for running the unit tests has been provided to you. You should call the main unit testing function RunTests() from your main() function. It will print its results to the console. The source file contains all the unit tests that have been provided, but you should seriously consider adding a few of your own. You will not be penalized for failing any tests you write at grade-time, but if you find a case you fail it's in your best interest to fix it anyway.

5 Requirements

5.1 Ray-Triangle Test (25 points)

Implement the intersection of a ray in 3D space and a triangle. This will be the basis for your ray intersection with meshes such as your surface revolutions and extrusions.

5.2 Ray-Cube Test (25 points)

Intersect a ray with a cube. You will use this with your furniture in the next assignment.

5.3 Ray-Sphere Test (25 points)

Like with the cube test, this will also be used with furniture.

5.4 Ray-Cylinder Test (25 points)

See above.

6 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!