

CS 770/870

Assignment 6: Primitive Ray Tracer

November 12, 2008

Due: Monday, 11/24 by 23:59:59

Last allowed submission: 12/1 23:59:59

Your task is to write a basic ray tracing program. You may build this upon any of your existing assignments; assignments 3 and 5 seem to be the best candidates. The key requirement for your starting system is the ability to define multiple objects at multiple locations.

Program features

1. Object-oriented.
 - Each “shape” object should know how to intersect itself with a ray.
 - You should create a class to represent a ray; in addition to knowing the origin and direction of the ray, it is convenient for it to also maintain min and max t values of interest. Both values can be used by the initial ray to define the bounds of the viewing frustum. The max value can be used for other rays to represent the closest hit found so far. Perhaps you want to implement a *RayHit* class that also maintains a reference to the closest object hit so far.
 - You should implement a class to represent a coordinate frame transformation. This class would encapsulate the matrix transformation from an object coordinate system to the world coordinate system. It should be able to create the inverse transformation and provide methods to transform points and vectors from object coordinates to world coordinates and from world coordinates to object coordinates.
2. Support spheres and boxes. Your ray object intersection should be done by an object method and should be done in the object’s own local coordinate system. This requires the ability to map the ray specification from world coordinates to object coordinates. Note that the point in polygon test is drastically simplified when the polygons are rectangles and already lie in a major plane, which is how you should define your basic box class.
3. Support at least 1 light source, including shadows. I.e., determining by ray tracing if a point on the surface of an object is visible to the (each) light source.
4. Support ambient, specular, and diffuse lighting.
5. Support specular ray reflection and semi-transparent surfaces with the transmitted ray.

Point allocation (note that all 100 points are accounted for this time)

- | | |
|----|---|
| 20 | 1. Correct ray/sphere intersection |
| 20 | 2. Correct ray/box intersection |
| 20 | 3. Image ray implementation: visibility only, no reflection, refraction or shadows. |
| 10 | 4. Lighting model implementation for at least 1 light source. |
| 10 | 5. Light ray implementation: Shadows |
| 10 | 6. Reflected ray implementation |
| 10 | 7. Transmitted ray implementation |

Notes:

1. You must create a non-graphical driver program to demonstrate the basic correctness of your sphere and box intersection code. This program should create a single simple sphere and box at their “canonical” locations in their own coordinate system. It will generate some easily understood rays in that system, find the intersections and print the ray, the t value and the intersection. The generated rays should test all cases and your output should be clear enough that *we* will be able to see easily that the tests works.
2. Points are earned by correctly implementing features robustly. Points are deducted for bugs, incorrect implementation, poor style, poor design decisions, etc.