CS184 Final Project — Global Illumination and Photon Mapping with Extended Ray Tracer

cs184-ad: Gabe Fierro, cs184-ae Emerson Knapp, cs184-au Harry Zhu

15 December 2011

Abstract

Our original goal for our final project was to implement additional features for the ray tracer we created for **Assignment 4**, and then implement photon mapping to augment the scenes we could then create with the modified ray tracer.

Contents

1 Features

We implemented the following features for the raytracer:

SCENE FILE INPUT

We created our own file format for detailing scenes for our program to render. SEE A FOLLOWING SECTION FOR A DETAIL OF THE FILE FORMAT.

ANTI-ALIASING

We implemented both uniform and jitter anti-aliasing techniques. Specifying the `-a N` option sends N^2 rays to each point on the screen (as opposed to the default 1), regularly spaced around the unit square represented at that pixel and averages the results to specify the color for that pixel. Specifying the `-ja N` option sends N^2 rays to random locations within the unit square and again averages the results. In practice, $N = 3$ is sufficient for the 800×800 images we generate, and uniform and jitter achieve essentially the same effect.

Images: Compare cornell2.t with no aa to the same scene, but with nine rays per pixel (run with `-ja 3`)

REFRACTION AND REFLECTION

Both rays and photons correctly refract and reflect when intersecting with objects in the scene according to the object's refractive index and reflectivity, respectively (details for photons below). Objects can be both reflective and refractive and we see the correct result.

Images: You can see an example of refraction here

EAR-CLIPPING

We specify vertices and faces in the manner of the OBJ file format: `v x y z` specifies a vertex at coordinates (x, y, z), and f n_1 n_2 n_3 creates a face using vertices n_1,n_2 and n_3. Using ear-clipping, we can specify arbitrary convex or concave polygons using the command f n_1 n_2 n_3 $\cdots$ n_k. The ear-clipping algorithm correctly identifies ears by differentiating between convex and concave vertices, and splits up the polygon into the appropriate triangles for rendering.

Images: This polygon contains both convex and concave vertices, and is specified by a single `f` line in the scene file.

PERSPECTIVE-CORRECT TEXTURE MAPPING

In the scene file, if a `.png` file is specified as a texture, then the objects in the scene use barymetric coordinates and linear interpolation (in the case of triangles) or uv-mapping using spherical coordinates (in the case of spheres) to use the texture as the surface of the object.

Images: Observe a black and white marble texture on a pair of triangles forming a rectangle, and a world map texture on a sphere.

AREA LIGHTS AND SOFT SHADOWS

TODO: EMERSON WRITE A DESCRIPTION

Images: doop doop doop

OBJ FILE FORMAT AND MULTIPLE SCENE SUPPORT

Just as in an OBJ file, in our scene files, one can specify vertices and n-sided polygons (as seen above), specify materials (using `usemtl FILENAME.mtl`, where FILENAME.mtl is a material file) as well as specify other scene or obj files to be included in the scene being rendered.

Images: Here is an example of rendering the free `angel.obj` model in a scene of our own construction. We've made the angel refractive to demonstrate how we can integrate external obj files in our scenes.

KD-TREE AND GENERAL OPTIMIZATION

We construct a KD-tree for the polygons in order to reduce the lookup time for calculating ray intersection. For photon mapping, we also create separate trees for global photons, shadow photons and caustic photons in order to facilitate k-nearest neighbors lookups for estimating radiance, generating shadow rays and visualizing caustics, respectively.
Additionally, we used OpenMP (`http://openmp.org/`) to parallelize the pixel interpolation, ray intersection and occlusion detection loops, resulting in a drastic speedup. For basic rendering (simple directional light, with no reflection or refraction) of a 100,000 polygon object such as the `angel.obj`, we were able to reduce rendering time from 90 minutes to 15 seconds.

Images: This image, consisting of almost 50,000 unique polygons, took about 15 seconds to render. We've titled it "Weeping Angel," after the alien race in the BBC show Doctor Who.
By the way, this is what happens if you incorrectly compute the bounding boxes....

2 Photon Mapping

2.1 Implementation Details

In order to implement photon mapping, we first determined how we were going to generate the photons that would propagate through the scene. For a point light, we generate random directions on a unit sphere

centered at the light's position and generate N photons in those directions (N is specified b the `-ph X` option, where $N = X \cdot 1000$). In order to make sure that we maintain the same amount of illumination for the scene regardless of how many photons we send out, we specify the power for the light in the scene file and weight the photons accordingly so the sum total of their flux is that of the light. [For directional lights, we follow a similar process].
In order to correctly model the photons "bouncing" around the scene, we use the Monte Carlo method known as **Russian roulette** to determine whether a given photon is

2.2 The Rendering Equation

For the purposes of photon mapping, we can consider the rendering equation as having four parts: direct illumination, specular reflection, caustics,

DIRECT ILLUMINATION

Direct illumination is the radiance for a location x given the occlusion of other objects in the scene and the light from a source that hits that location. We used normal ray tracing to estimate this integral, but for debugging purposes (using the `-r` option) we performed a k-nearest neighbors search at point x to gain an estimate for the direct illumination.

SPECULAR REFLECTION

Photon mapping can be used to estimate the specular reflection for at a point x, but we need a good deal more photons in order to produce that effect (see notes for caustics below). So, as with direct illumination, we use the normal ray tracing to calculate the specular reflection at a point x.

CAUSTICS

With the global photon map, because we get a relatively even distribution of photons, we get an estimate of the caustic for refractive objects. We send an additional number of photons toward each refractive object in order to increase the number of photons taken into account for estimating the caustic, which we estimate by doing the k-nearest neighbor operation and averaging the photon flux (which will be more accurate because there's a higher concentration of photons).

INDIRECT ILLUMINATION

At a given point x, we have to determine how much