PHOTON MAPPING

15-462: Computer Graphics

March 19, 2009

Recap

- Global illumination models can be basically partitioned into two sets:
- Point-sampling models
  - Ray tracing
- Integral-based models
  - Radiosity
- There are also hybrid models that are a mix of the two.

Questions to Think About

- What are some of the effects that we can't get using the rendering methods that we've gone over?
- More specifically, why can't we capture these effects with our current models?
- Furthermore, can we come up with a good way to do this efficiently?
- Is there a *perfect* model that can do all of this?

Point-Sampling Models

- Monte Carlo Ray Tracing
  - Path Tracing
  - Bidirectional Path Tracing
  - Metropolis Light Transport
- Photon Mapping

Monte Carlo Ray Tracing

- Many of the more complex ray tracing-based algorithms can be defined as "Monte Carlo algorithms," a class of algorithms that use repeated random sampling to compute results.
- Monte Carlo ray tracing is a much more complicated extension of distributed ray tracing, although the two terms tend to overlap.

Problem(s)

- The biggest problem that these Monte Carlo ray tracing algorithms run into is the amount of noise left behind.
  - This is particularly noticeable regarding caustics.
- We can remedy this by increasing the amount of rays we send out, but this is computationally more and more expensive.

Motivation

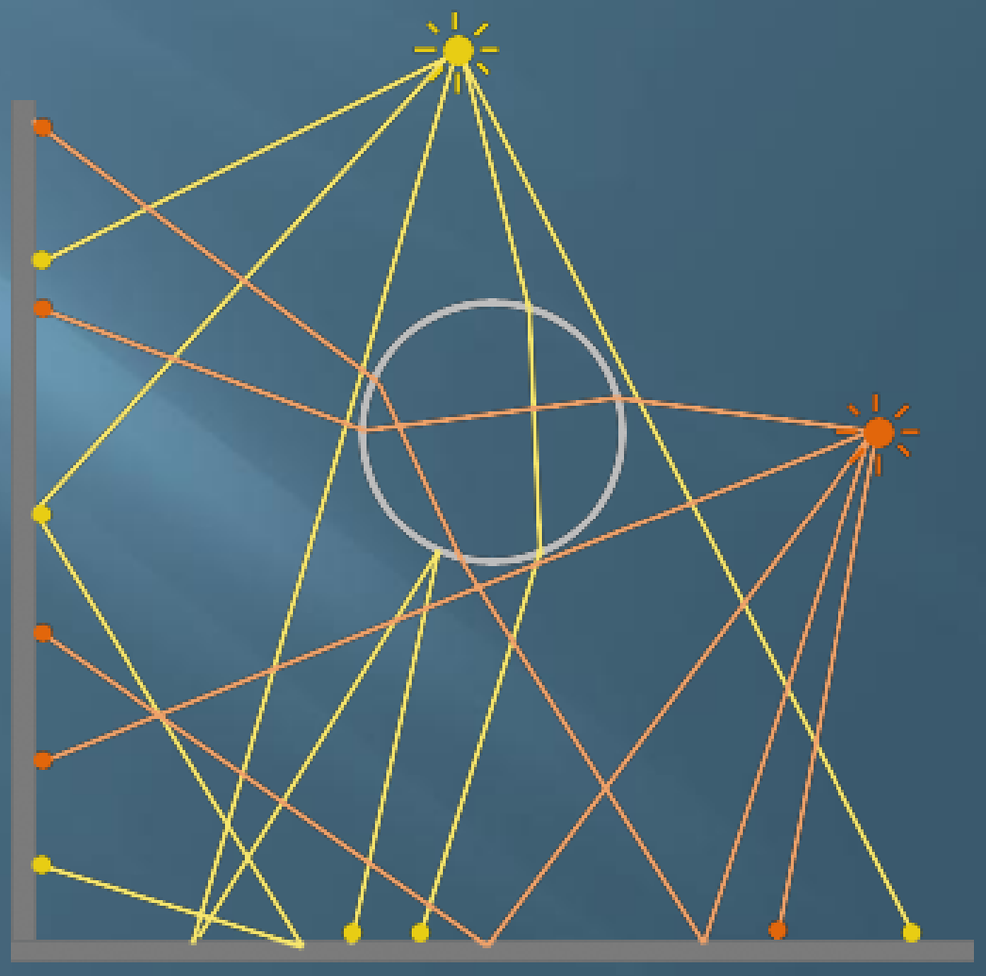
- We want a global illumination model that can do all of the following:
  - All global illumination effects can be simulated
  - Arbitrary geometry
  - Low memory consumption
  - Result is correct (ignoring the variance/noise)
- Photon mapping is an alternative to the rendering algorithms presented earlier.

The Basic Idea

- Photon mapping is a two-pass algorithm
- First pass: Photon tracing
  - Fire photons from all of the light sources in the scene
  - Build a “photon map” that stores their locations after being fired into the scene
- Second pass: Rendering
  - Render the scene by taking the information stored in the photon map into account

Photon Tracing

- The basic idea is simple: we fire a bunch of photons from all of the lights in the scene.
- A photon thought of in a similar way to a ray, except that it also has its own intensity, or power, represented by its color.



Photon Tracing

- We can handle a variety of light sources using this model:
  - Point lights
  - Spherical lights
  - Area lights
  - Complex lights
- Regardless of the shape, the basic idea is the same: for every light, we want to send out photons uniformly in all directions.
  - This can be optimized further by only firing photons in the direction of the geometry in the scene.

Photon Tracing

- Once a photon is fired, it bounces around the scene until it reaches some termination condition.
- In the most general case, a photon can either be reflected or absorbed by the object.
  - Absorption is the general termination condition upon which we then store it in the photon map.
- More accurately, the BRDF of the object with which the photon collides will determine what will happen to it.
  - That is, can it be specularly *and* diffusely reflected, refracted, etc?

Photon Tracing (Caustics)

- For caustics, things are slightly different.
  - We will specifically target specular objects in the scene such that we can capture caustics.
  - While we normally might allow diffuse reflection for some photons, after a photon has been specularly reflected or refracted for some time, then comes in contact with a diffuse object, we want to terminate it instantly and store it in the photon map.

Russian Roulette

- The technique used to determine what happens to a photon when colliding with an object is a probabilistic technique called Russian Roulette.
- The following is an example of how to use Russian roulette on an object with both diffuse and specular reflection (with probabilities p_d and p_s respectively).
- We generate some random number $\varepsilon \in [0,1]$
  - $\varepsilon \in [0, p_d]$ => diffuse reflection
  - $\varepsilon \in]p_d, p_s + p_d]$ => specular reflection
  - $\varepsilon \in]p_s + p_d, 1]$ => absorption

Russian Roulette

- Why does Russian Roulette work? And also, why do we use it?
- Recall that when light hits an object, based on the BRDF, fractions of it are reflected, absorbed, etc…
  - One way to deal with this is to spawn additional photons at each collision and reflect, refract, etc them with decreased intensity back into the scene.
  - However, this substantially increases the number of photons we fire.
  - Russian Roulette allows us to fire less photons at full intensity. We may fire less photons, but the expected value of the intensity remains the same.

The Photon Map

- We can use a spatial data structure to store our photon map. A good spatial data structure will give us the following:
  - Relatively fast insertions
  - Relatively fast lookups, but more importantly, a way of looking up elements “neighboring” a particular element.
- The structure that is most generally used for this is a balanced k-d tree. (This is what you will be asked to implement in project 4.)

(Balanced) K-D Trees

- Why a balanced k-d tree?
  - A 3-D grid is impractical (and subsequently, so is an octree) because our distribution of photons is not uniform.
  - A k-d tree is much more tailored to the distribution of the photons and balanced, it gives use $O(\log n)$ insertion and individual lookups. It also gives us a good representation of how to return the n nearest photons in relation to a given point. This is known more commonly as nearest-neighbor search.

The Photon Map (x2)

- One thing to note is that we actually want to keep two photon maps: one just for caustics and one for global illumination in general.
  - Caustics photon map: Concentrate on firing photons directly at specular objects in the scene.
  - Global photon map: Fire photons uniformly from all light sources into the scene.
- Why?
  - The global photon map will not contain enough photons to accurately represent caustics in our scene since not enough photons will be concentrated at points where caustics occur.

Radiance Estimate

- Once a photon has been fired and absorbed (or terminated), we can look it up again in the future. More specifically, we will need to look it up when calculating the radiance estimate of a point.
- The idea is simple: given a point, we want to find the *n* nearest photons and average over them to compute the radiance at that point.
- Getting the *n* nearest photons is easy: we run nearest neighbor search on our photon map.

Radiance Estimate

- Once we have these *n* photons, we want to evaluate the reflected radiance at our point.

$$L_r(x, \vec{w}) = \int_{\Omega} f_r(x, \vec{\omega}', \vec{\omega}) L_i(x, \vec{\omega})(\vec{n}_x \cdot \vec{w}') d\vec{\omega}'$$

- We can approximate this by drawing a sphere around our *n* photons with our point as the center. We then project all of our photons onto the circle defined by the intersection of the sphere and our object surface.
- Summing up all of the intensities of the photons and dividing them over the area of the circle will give us an accurate approximation for the radiance.

Rendering

- Photon tracing is a pre-rendering step that should be computed once before the second pass of the algorithm.
- In the rendering pass, we basically want to use the photon map(s) we built to assist us in computing the color of an object at a given point.

Color Computation

- We can now break down the color computation at a point in the following four components:
  - Direct Illumination
  - Specular Reflection
  - Caustics
  - Indirect Illumination
- Thus, solving the rendering equation boils down very simply to: $L = L_D + L_S + L_C + L_I$

Direct Illumination

- Direct illumination can be represented by the following integral:

$$\int_{\Omega_x} f_r(x, \vec{\omega}', \vec{\omega}) L_{i,l}(x, \vec{\omega})(\vec{\omega}' \cdot \vec{n}) d\vec{\omega}'$$

- There are a variety of ways to evaluate this term, but the simplest way is to take a standard ray tracing algorithm and compute the color at the point of intersection without taking into account the contributions of any specular reflection…

Specular Reflection

- Specular and glossy reflections (along with transmission through dielectrics) can be represented by the following integral:

$$\int_{\Omega_x} f_{r,s}(x, \vec{\omega}', \vec{\omega})(L_{i,c}(x, \vec{\omega}) + L_{i,d}(x, \vec{\omega}))(\vec{\omega}' \cdot \vec{n}) d\vec{\omega}'$$

- The best way to evaluate this term is to take advantage of a ray tracing algorithm and simply evaluate the color returned by reflecting or refracting a ray at the point of intersection.

Caustics

- Caustics can be represented using the following integral:

$$\int_{\Omega_x} f_{r,D}(x, \vec{\omega}', \vec{\omega}) L_{i,c}(x, \vec{\omega})(\vec{\omega}' \cdot \vec{n}) d\vec{\omega}'$$

- Calculating an accurate contribution from caustics can be achieved using our caustics photon map. We simply wish to take the radiance estimate from the map.
  - (Note: The global photon map will also give an approximate evaluation, but it is not as accurate)

Indirect Illumination

- Indirect illumination can be represented using the following integral:

$$\int_{\Omega_x} f_{r,D}(x, \vec{\omega}', \vec{\omega}) L_{i,d}(x, \vec{\omega})(\vec{\omega}' \cdot \vec{n}) d\vec{\omega}'$$

- Taking the radiance estimate from the global photon map will give us a good approximation of this term.
  - A more accurate evaluation can be achieved using Monte Carlo ray tracing, but this requires substantially more computation. The nice thing about our approximation is that caustics are separated out, which is the main cause for noise in Monte Carlo ray tracing.

Final Calculation

- Thus in summary, for each of the components of the color at a given point:
  - L_D best achieved using a general ray tracing algorithm (eye and shadow rays)
  - L_S: also best achieved by taking the contribution from specular and transmitted rays
  - L_C: taking the radiance estimate from our caustics photon map
  - L_I : taking the radiance estimate from our global photon map or using Monte Carlo ray tracing techniques

Other Effects

- Photon mapping is also suitable for picking up other effects (which we aren't actually going into):
  - Volume caustics
  - Subsurface scattering