



<http://en.wikipedia.org/wiki/Gonioreflectometer>

Stochastic Rendering Techniques

Scott Houde, John F. Hughes

Outline

- ▶ What is a stochastic rendering method?
 - ▶ And why would we want to use one?
- ▶ Quick Ray tracing review
- ▶ High Level math overview
 - ▶ Probability and Monte Carlo Integration
 - ▶ Russian roulette
- ▶ Stochastic Rendering methods
 - ▶ Path tracing
 - ▶ Bidirectional Path tracing
 - ▶ Photon Mapping
 - ▶ Metropolis Light Transport (MLT)

Stochastic Methods

- ▶ “A stochastic process is one whose behavior is non-deterministic, in that a system's subsequent state is determined both by the process's predictable actions and by a random element”

– <http://en.wikipedia.org/wiki/Stochastic>

- ▶ Will use random numbers and probability to solve the rendering equation



Review: Kajiya's Rendering Equation

$$L(i \rightarrow j) = L^e(i \rightarrow j) + \int_{k \in S} L(k \rightarrow i) f_r(k \rightarrow i \rightarrow j) G(k \rightarrow i) dk$$

- ▶ $L(i \rightarrow j)$ is the amount of light traveling along the ray from point i to point j .
- ▶ $L^e(i \rightarrow j)$ is the amount of light emitted by the surface (emittance)
- ▶ $f_r(k \rightarrow i \rightarrow j)$ is the *Bidirectional Reflectance Distribution Function* (BRDF) of the surface. Describes how much of the light incident on the surface at i from the direction of k leaves the surface in direction of j .
- ▶ $G(k \rightarrow i)$ is a geometry term which involves occlusion, distance, and the angle between each surface and the ray from k to i .
- ▶ Goal of this lecture: evaluate $L(i \rightarrow j)$ using stochastic techniques

¹J Kajiya. "The Rendering Equation." SIGGRAPH 1986, pp. 143-150

Why use stochastic techniques?

Good question? Why would we....

- ▶ One reason is that some* of these methods are considered unbiased
 - ▶ More importantly, the error they introduce can be estimated, so you can say “There is a high probability the result is within 5% of right”
 - ▶ **Important Note:** These methods are **NOT** real time!
- ▶ Another is that they are more “realistic”
 - ▶ Solving the rendering equation exactly would produce the best results however we simply can't do that
 - ▶ Instead we are going to use probability and “randomness” in order to get a result which should be close

*Pathtracing can be unbiased, photon mapping is biased

Recursive Ray Tracing (1/3)

- Our recursive lighting equation (Phong lighting + specular reflect + transmission):

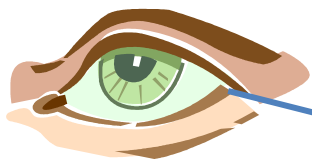
$$I_{\lambda} = \underbrace{L_{a\lambda} k_a O_{a\lambda}}_{\text{ambient}} + \sum_{\text{lights}} f_{att} L_{p\lambda} \left[\underbrace{k_d O_{d\lambda} (\vec{n} \cdot \vec{l})}_{\text{diffuse}} + \underbrace{k_s O_{s\lambda} (\vec{r} \cdot \vec{v})^n}_{\text{specular}} \right] + \underbrace{k_s O_{s\lambda} I_{r\lambda}}_{\text{reflected}} + \underbrace{k_t O_{t\lambda} I_{t\lambda}}_{\text{transmitted}}$$

recursive rays

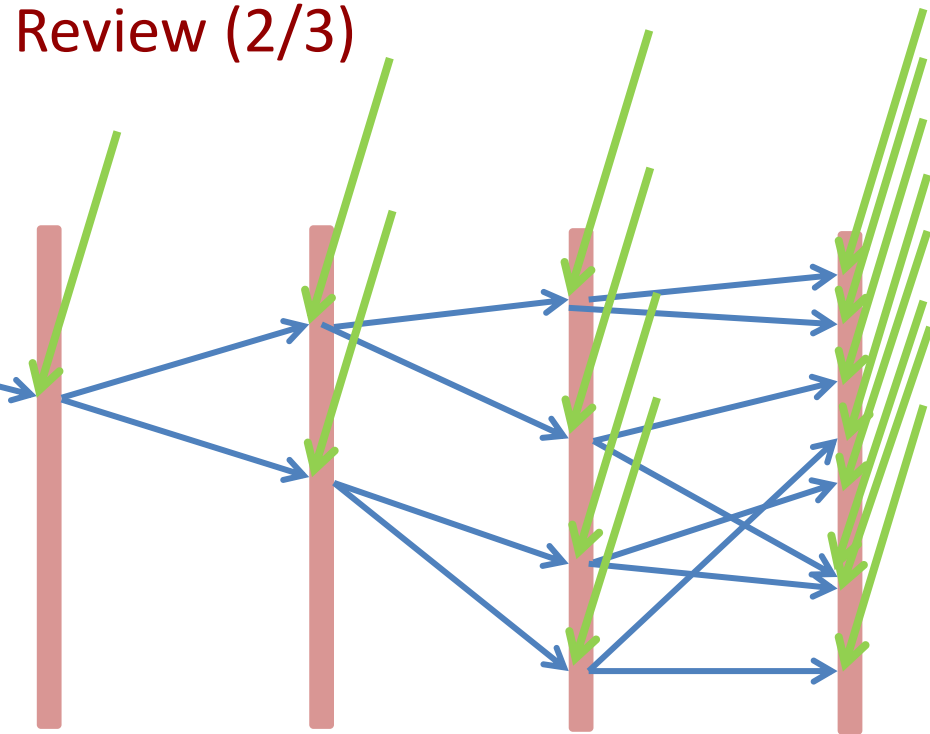
- I is the total color at a given point (lighting + specular reflection + transmission, λ subscript for each r,g,b)
 - Its presence in the transmitted and reflected terms implies recursion
- L is the light intensity; L_p is the intensity of a point light source
- k is the attenuation coefficient for the object material (ambient, diffuse, specular, etc.)
- O is the object color
- f_{att} is the attenuation function for distance
- \vec{n} is the normal vector at the object surface
- \vec{l} is the vector to the light
- \vec{r} is the reflected light vector
- \vec{v} is the vector from the eye point (view vector)
- n is the specular exponent
- note: intensity from recursive rays calculated with the same lighting equation at the intersection point
- light sources contribute specular and diffuse lighting
- Note: single rays of light do not attenuate with distance; purpose of f_{att} is to simulate diminishing intensity per unit area as function of distance for point lights (typically an inverse quadratic polynomial)

Recursive Ray Tracing Review (2/3)

A visual example



Most of your work is taken up by light bounced multiple times, which hardly matters since each bounce attenuates the light transport.



The red bars indicate the set of all points in the scene. The blue lines are the rays being traced, each branch indicates a scattering event. The green lines represent light arriving from light sources.

Recursive Ray Tracing Review (3/3)

- ▶ Given the definition from earlier do you think the recursive ray tracer we implemented is stochastic?
 - ▶ Not only no, but hell no!
- ▶ Why isn't it?
 - ▶ There is no random element!

No one told me there would be math

"I don't believe in mathematics." ~Albert Einstein

- ▶ In order to proceed we need to cover a couple of mathematical concepts
 - ▶ Probability
 - ▶ Monte Carlo theory
 - ▶ Russian roulette
- ▶ Purposefully going over them at a high level and with some hand waving
 - ▶ But I am going to try to keep the examples specific to what we are trying to accomplish

Probability from 50,000 feet

- ▶ Everything that could happen in a given situation has a probability of occurring
 - ▶ Also called a % chance
- ▶ The sum of the probabilities of all (disjoint) events always equals 1 (i.e. sum to 100%)
- ▶ For some methods (e.g. photon mapping) we are looking at the probability that a photon arrives at a particular position in the scene
- ▶ For other methods (e.g. path tracing) we are looking at the probability of tracing a ray in a certain direction



That was too far out, a little closer

- ▶ Random Variable
 - ▶ A variable that can take one of many values, each value having some probability of occurring
- ▶ $\Pr[X = 5]$ denotes the probability X produces the value 5 as its output
 - ▶ If X models a fair, six-sided die: $\Pr[X = 5] = \frac{1}{6}$
 - ▶ $\Pr[\dots]$ notation can be more abstract: $\Pr[X > 4] = \frac{1}{3}$

Probability notations and examples

- ▶ Expected Value (of any expression)
 - ▶ Formalizes the notion of an average
 - ▶ Also known as the mean (denoted μ)
 - ▶ Denoted $E[X]$ for random variable X
- ▶ Multiple interpretations for $E[X]$
 - ▶ Average of all values i that X produces, weighted by probability of i :

$$\sum \Pr[X = i] \times i \text{ (discrete)} \qquad \int PDF(i) \times i \, di \text{ (continuous)}$$

- ▶ The value you “expect” to get after averaging many, many **samples**:

$$\lim_{N \rightarrow \infty} \sum \frac{X_i}{N} \quad \text{where } X_1, \dots, X_i, \dots, X_N \text{ randomly chosen using same PDF}$$

Important Probability Concept for Rendering

The notion of Probability Density

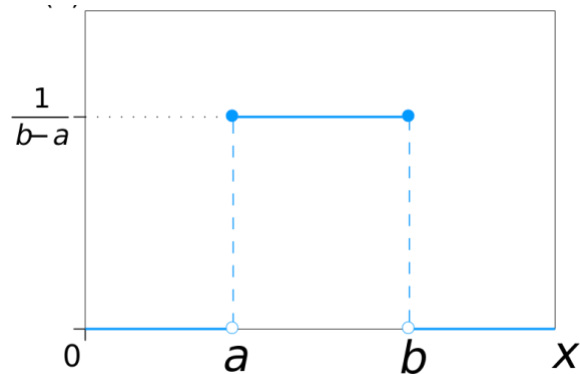
- ▶ When discussing probabilities where the domain is some interval in the reals we talk about the probability of generating numbers *within an interval* $[a,b]$ and not of generating a particular number
- ▶ More generally we posit the existence of a ***Probability Density Function (PDF)*** with the property that:

$$\Pr\{\text{a random number in the interval } [a, b] \text{ is generated}\} = \int_a^b p(x)dx$$

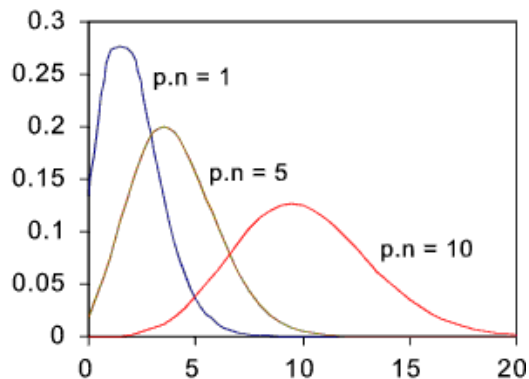
Thus integral over entire range = 1 (again, sum to 100%)

- ▶ If you are interested in more details about this see pages 810-812 in the book, for everyone else just trust me on this.

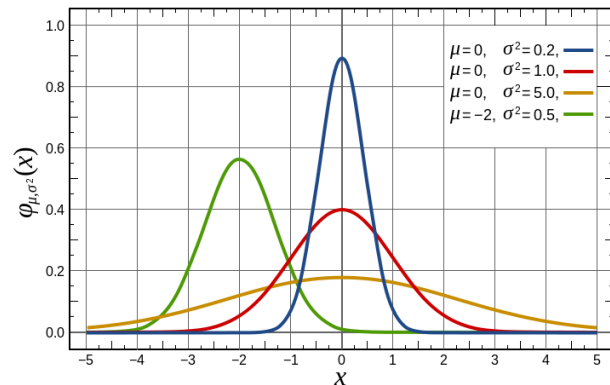
Common PDFs



Uniform, every point in interval is equally likely



Poisson, often used to describe the occurrence of rare events



Normal distribution

History: Monte Carlo Method

The Monte Carlo method was coined in the 1940s by John von Neumann, Stanislaw Ulam and Nicholas Metropolis, while they were working on nuclear weapon projects (Manhattan Project) in the Los Alamos National Laboratory.

It was named after the Monte Carlo Casino, a famous casino where Ulam's uncle often gambled away his money.

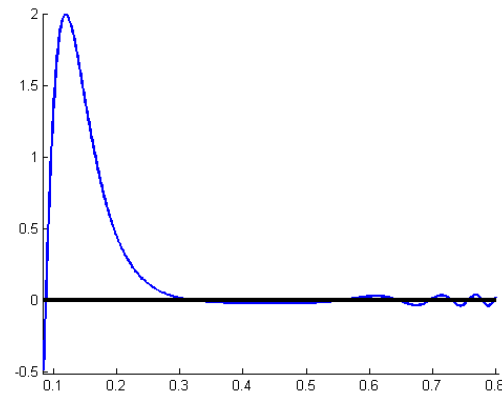
http://en.wikipedia.org/wiki/Monte_Carlo_method

Monte Carlo integration (1/4)

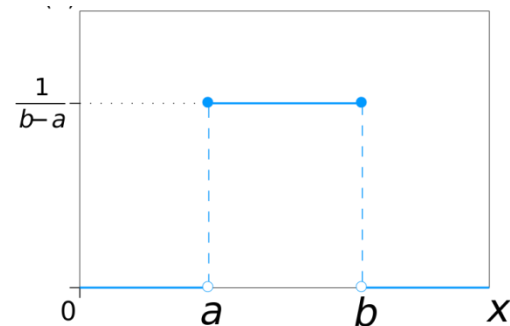
- ▶ Sometimes we need to integrate functions where computing antiderivatives is impossible or impractical
 - ▶ **Monte Carlo integration** is a probabilistic method for estimating complicated integrals
 - ▶ For example the integrand in the reflectance equation might not be described by an algebraic equation
- ▶ A Monte Carlo approach lets us *estimate* the integral of any function f over any interval $[a,b]$
 - ▶ Or integrals of more complicated functions over more complicated domains
- ▶ *Important note: There are several different Monte Carlo approaches which we aren't going to talk about today (MISER, VEGAS, etc..)*

Monte Carlo integration (2/4)

- ▶ We come to this estimate by
 - ▶ Taking randomly chosen points uniformly in the interval $[a,b]$
 - ▶ Evaluating the function at those points
 - ▶ Averaging the results
 - ▶ Dividing the average by the PDF of the uniform $\frac{1}{b-a}$ as a normalization step
- ▶ Of course every time we do this we get a different estimate, the variance in these estimates depends on the shape of the function



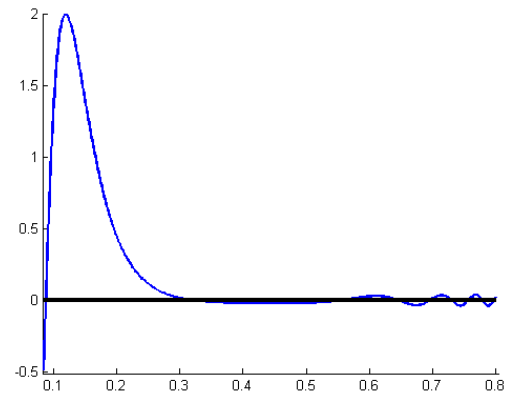
Function $f(x)$ we want to integrate



Uniform distribution

Monte Carlo integration (3/4)

- ▶ We can reduce this variance by generating the random points non-uniformly
- ▶ If we favor points where $f(x)$ is large we can drastically reduce the variance of the estimate
- ▶ This approach is called **importance sampling**
 - ▶ However we don't just alter the point selection strategy, that would just increase our estimate of the integral. We have to also alter the normalization.
 - ▶ The end result is a lower-variance estimate of the integral



Function $f(x)$ we want to integrate

Monte Carlo integration (4/4) -- why it works

- ▶ From before we know that:

$$\lim_{n \rightarrow \infty} \sum_{i=1}^n \frac{X_i}{n} = E[X] = \int_a^b p(x)x \, dx$$

- ▶ In Monte Carlo Integration (without normalization), we are doing:

$$\sum_{i=1}^n \frac{f(X_i)}{n} \approx E[f(X)] = \int_a^b p(x)f(x) \, dx$$

- ▶ But we want $\int_a^b f(x) \, dx$, so we normalize, therefore getting:

$$\sum_{i=1}^n \frac{f(X_i)}{np(x)} \approx E\left[\frac{f(X)}{p(X)}\right] = \int_a^b f(x) \, dx$$

Why would we want to do all this math?

- ▶ When light hits a surface it has a probability of reflecting anywhere within a hemisphere
 - ▶ Or if the surface is refractive and reflective anywhere within a sphere
- ▶ This is the kind of problem Monte Carlo integration can solve. Based on the probability of scattering function for the material and surface we can estimate how much light goes in a specific direction

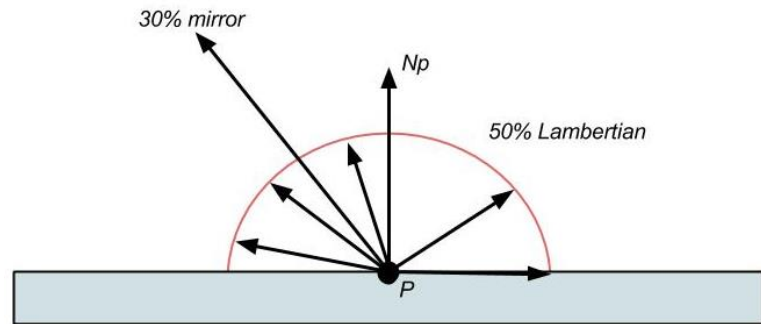


Figure 29.3: Scattering example: a 50% matte reflector that's also 30% mirror reflective, but not transmissive.

Russian Roulette (1/2)

- ▶ This method gives us an unbiased way to control the number of 'bounces' our path will take in the scene
 - ▶ Simply stopping after a certain number of bounces introduces bias!
- ▶ At each step
 - ▶ We generate a random floating point number between 0 and 1
 - ▶ Compare that to our Russian roulette level, which can be any number between 0 and 1, 0.2 is a good one to use
 - ▶ If our random number $>$ Russian roulette level, then keep going

Russian Roulette (2/2)

- ▶ Applying this method in our code
 - ▶ Let f' be the result of our function f after our Russian roulette method is applied
 - ▶ Before each sample
 - ▶ With some probability p , don't trace a ray
 - ▶ With probability $(1 - p)$, do trace a ray
- ▶ Problem: expected value of this method $\neq E[f(i)]$
 - ▶ With probability p : $f'(i) = 0$
 - ▶ With probability $(1 - p)$: $f'(i) = f(i)$
 - ▶ $E[f'(i)] = (p)0 + (1 - p)E[f(i)] = (1 - p)E[f(i)]$
- ▶ Solution: redefine f' to include a division by $(1 - p)$
 - ▶ $E[f'(i)] = (p)0 + (1 - p) \frac{E[f(i)]}{1-p} = E[f(i)]$



continental roulette

Let's go deeper

- ▶ Before I keep going, any questions?
- ▶ If I haven't lost everyone let's take a deeper look
- ▶ Lets go look at some stochastic rendering methods, with some pseudo code and a little more math

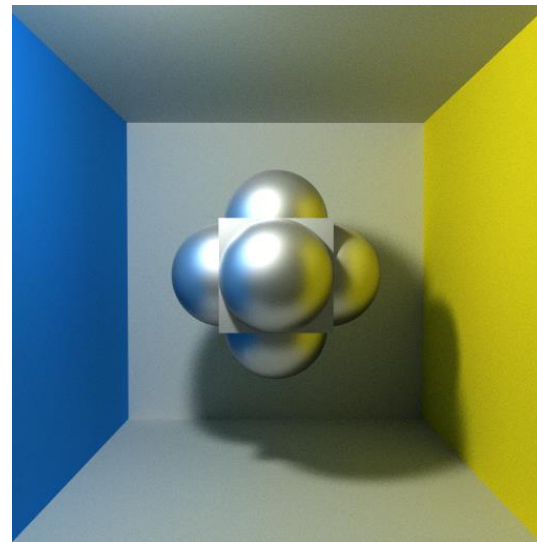


Convergence Characteristics

- ▶ An **unbiased** algorithm produces the exact solution to the rendering equation when infinity runs (with any fixed number of samples) are averaged together
- ▶ A **consistent** algorithm produces the exact solution when the sample size approaches infinity
- ▶ For example:
 - ▶ Photon mapping is biased but consistent
 - ▶ If the photon map contained a single photon, approximation of indirect lighting would be non-existent. Wrong result no matter how many passes.
 - ▶ If the photon map contained infinity photons, would match rendering equation. Intuitively, shooting infinity photons simulates nature.
 - ▶ Large photon maps give good results, converge faster than path tracers

Path Tracing

- ▶ By J. Kajiya, *[“The Rendering Equation”](#)* SIGGRAPH, 1986, Pages 143-150
- ▶ Uses Monte Carlo integration to approximate the light reaching each pixel
- ▶ Pros
 - ▶ Robust: handles many lighting phenomena
 - ▶ Correct: always converges to exact solution
- ▶ Cons
 - ▶ Takes many samples to converge to correct result
- ▶ Used to create ‘correct’ reference images, but faster techniques often used in practice



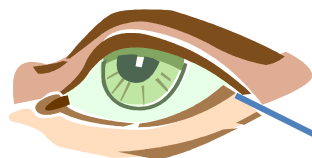
Screenshot of real-time path tracing demo by former TA Evan Wallace
<http://madebyevan.com/webgl-path-tracing/>

Path Tracing - In plain English

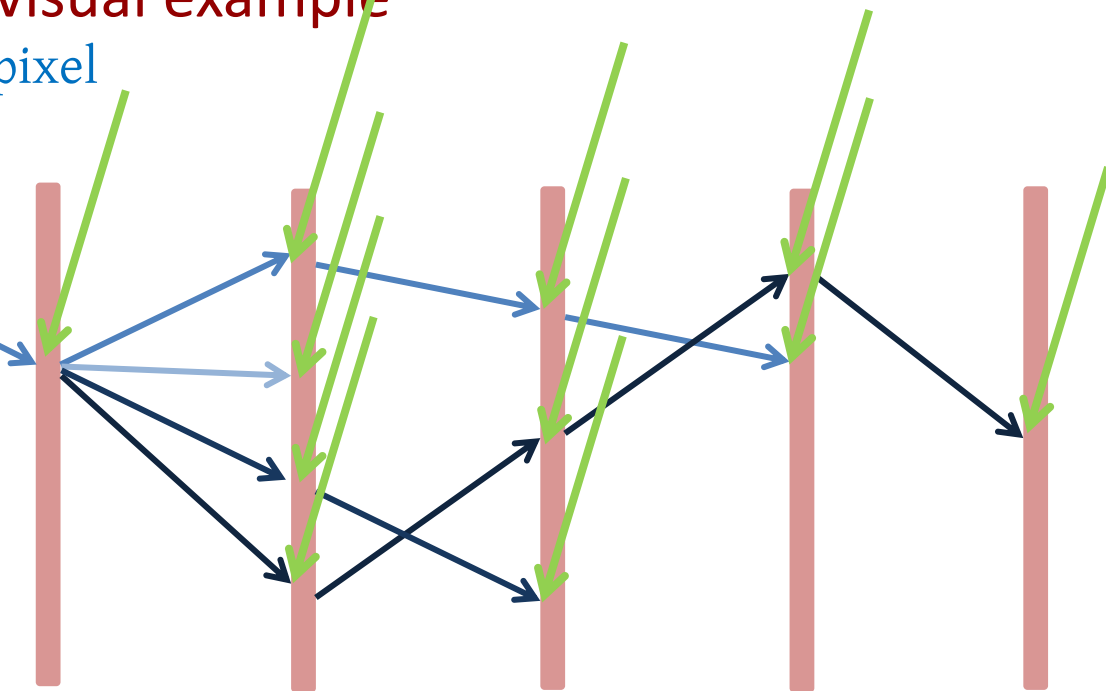
- ▶ Much like Ray tracing you start with a ray being traced from the eye into the scene
- ▶ If you intersect a surface
 - ▶ Get the direct and emitted light contribution to the intersection point, then check your Russian roulette probability,
 - ▶ If not killed:
 - Bounce in a random direction sampled from the hemisphere (or sphere)
 - If you hit a surface, repeat (get emitted and direct light, check Russian roulette, etc.)
 - ▶ If killed:
 - Return the sample color for the pixel weighted by the Russian roulette probability
- ▶ Repeat many, many times for each pixel

Path Tracing – a visual example

Tracing 4 paths per pixel



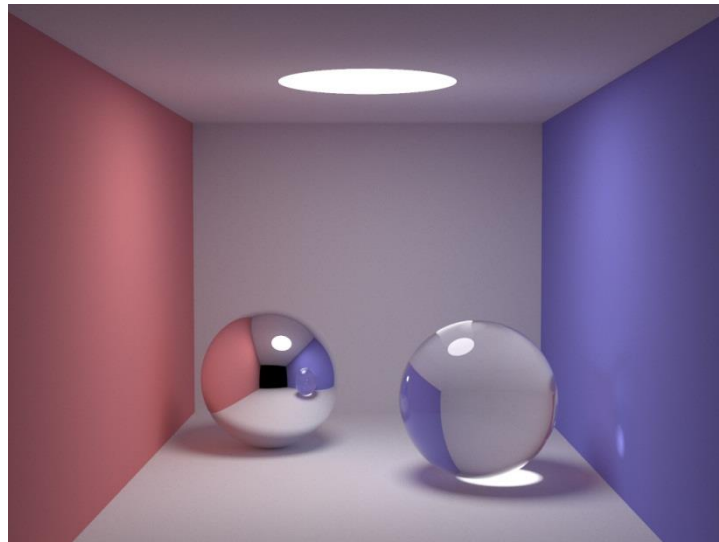
Most of your work here involves direct or once-scattered light, which is putting the work where it is most useful



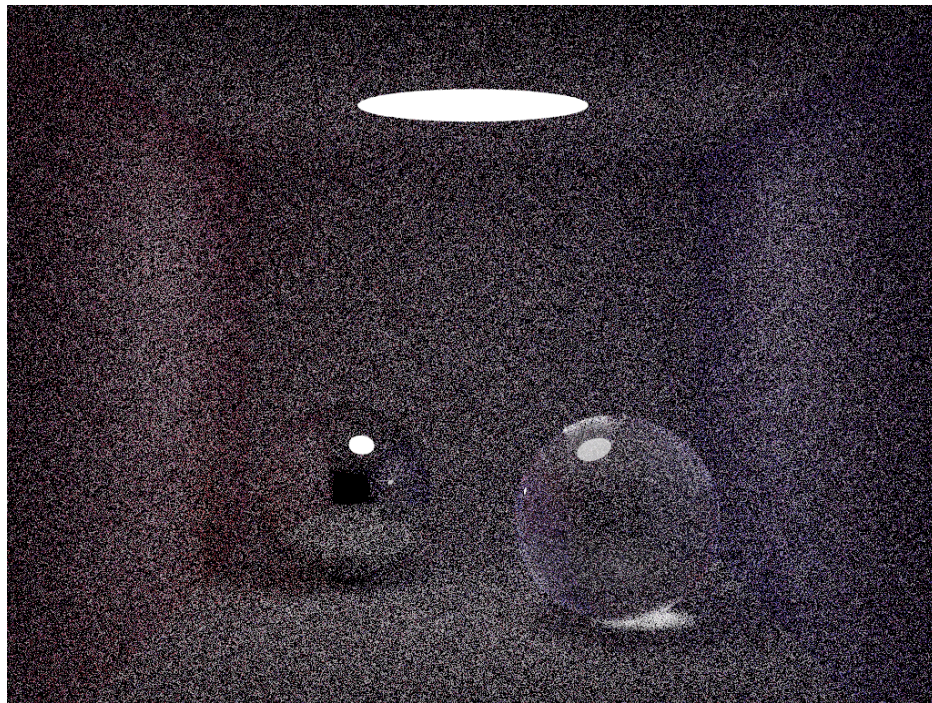
The red bars indicate the set of all points in the scene. The blue lines are the rays being traced, each branch indicates a scattering event. The green lines represent light arriving from light sources.

This gives us a Progressive Refinement

- ▶ Visualization of scene as more and more samples are averaged
- ▶ Screenshots from “smallpt” path tracer by Kevin Beason
 - ▶ (in 99 lines of C++!)
 - ▶ <http://www.kevinbeason.com/smallpt/>
- ▶ Final image:
- ▶ How does it look in progress?

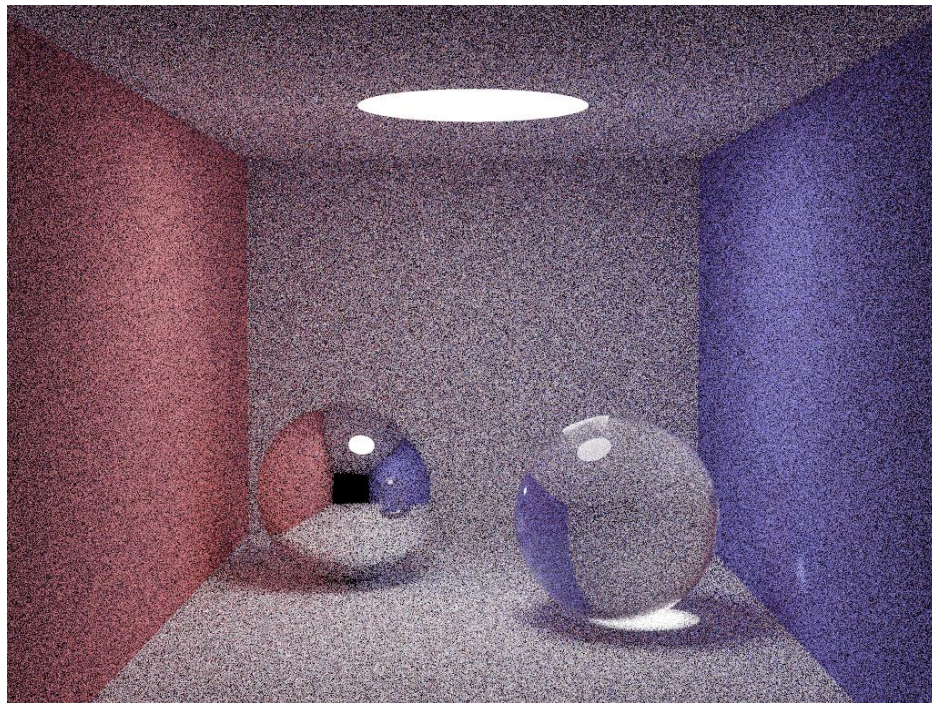


Progressive Refinement



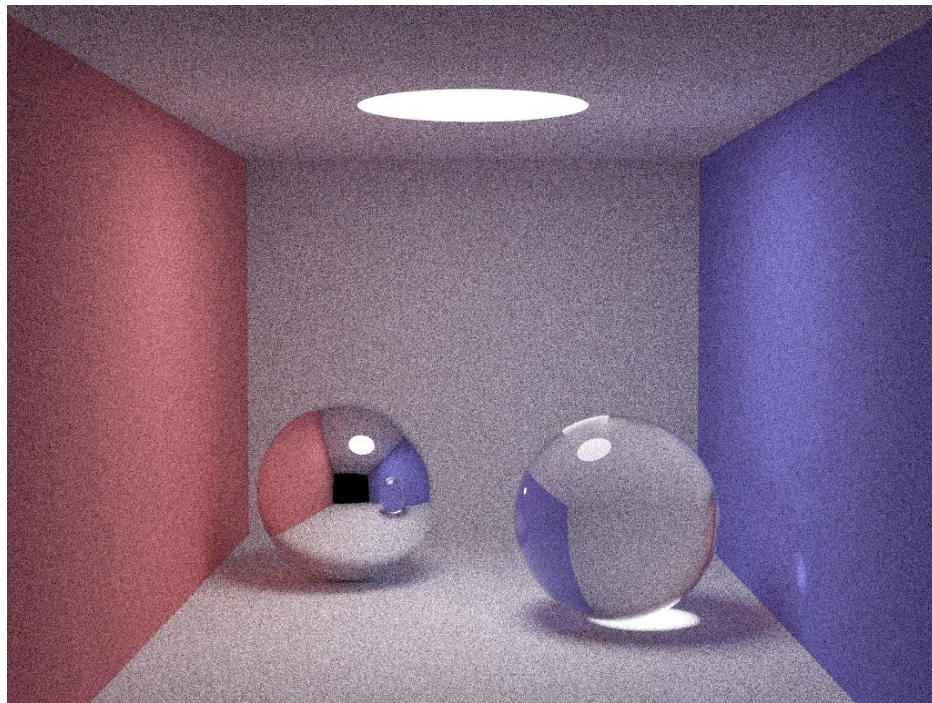
8 samples per pixel in 13 seconds

Progressive Refinement



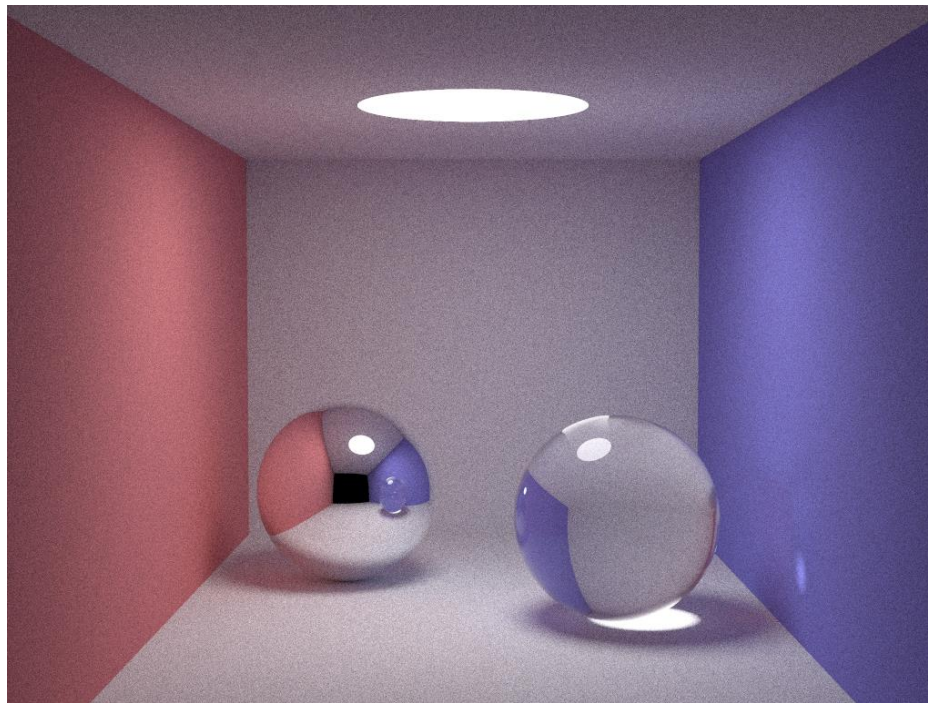
40 samples per pixel in 63 seconds

Progressive Refinement



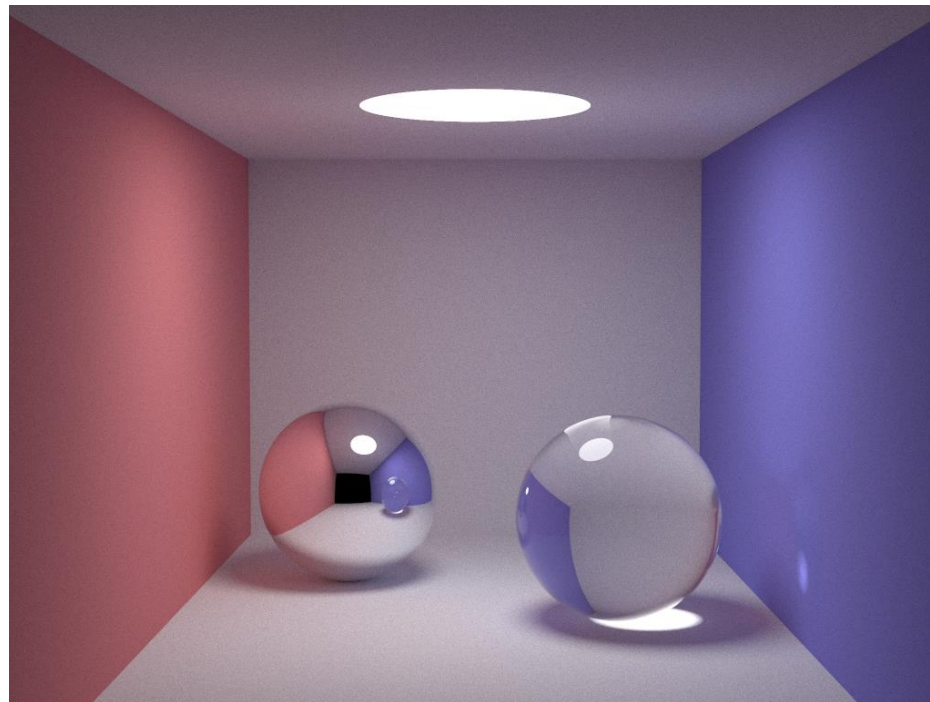
200 samples per pixel in 5 minutes

Progressive Refinement



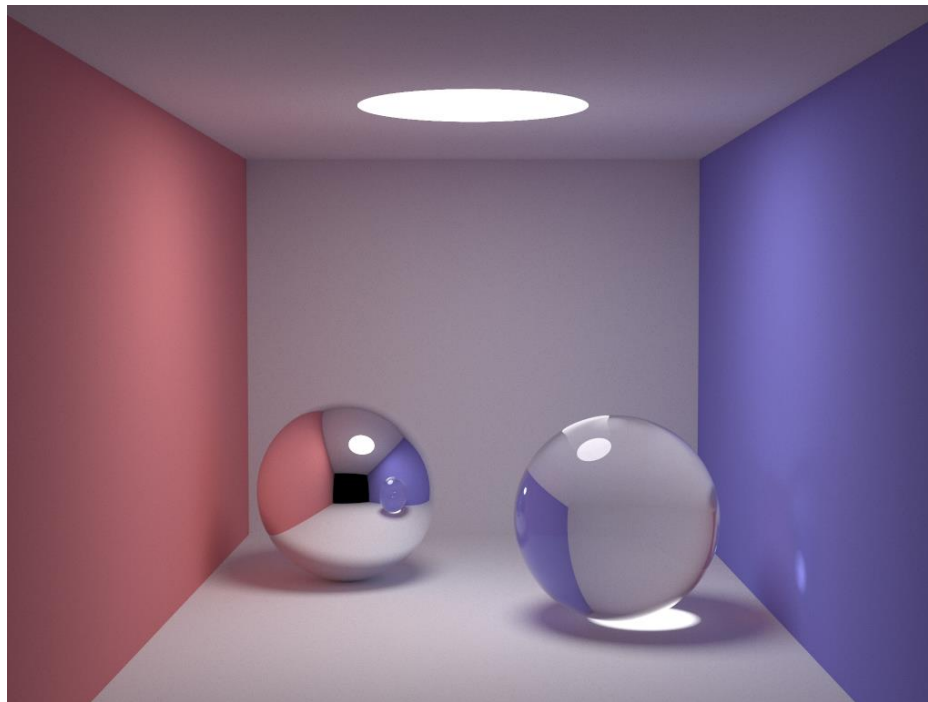
1000 samples per pixel in 25 minutes

Progressive Refinement



5000 samples per pixel in 124 minutes

Progressive Refinement

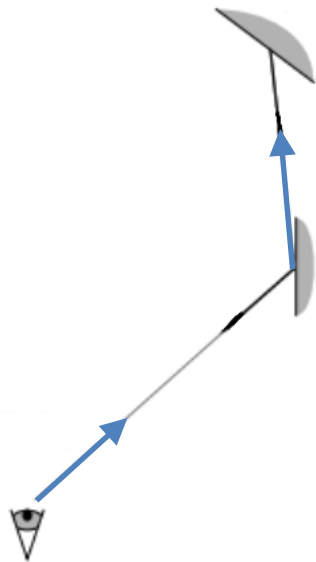


25,000 samples per pixel in 10.3 hours

A Naïve Path Tracer (1/3)

$$\int f(x) dx = E \left[\frac{f(i)}{\text{PDF}(i)} \right] \approx \frac{1}{N} \sum \frac{f(X_i)}{\text{PDF}(X_i)}$$

- ▶ Similar to recursive raytracing
- ▶ Instead of tracing light/reflection rays, bounce in a random direction
 - ▶ Uniformly sample hemisphere to select the ray direction
 - ▶ Corresponds to the X_i term in Monte Carlo integration
- ▶ Evaluate incoming light $L(k \rightarrow i)$ (using path tracing!)
- ▶ Multiply by BRDF and occlusion factors (f_r and G): $f(X_i)$
- ▶ Divide by probability ($\text{PDF}(X_i)$)
- ▶ Repeat and average ($\frac{1}{N} \sum \frac{f(X_i)}{\text{PDF}(X_i)}$)
- ▶ Ray tracing step is infinitely recursive.
How do we bottom out? (next slide)



A Naïve Path Tracer (2/3)

$$L(i \rightarrow j) = L^e(i \rightarrow j) + \int_{k \in S} L(k \rightarrow i) f_r(k \rightarrow i \rightarrow j) G(k \rightarrow i) dk$$

`L(i, j):`

`L_reflected := 0`

`p_RR := russian roulette probability constant`

`if random(0, 1) > p_RR:`

`k := sample direction uniformly on hemisphere and trace ray`

`L_reflected := ((L(k, i) * fr(k, i, j) * G(k, i))`
`/ (1 - p_RR)`
`/ PDF(k))`

`return L_e(i, j) + L_reflected`

Russian Roulette
Probability

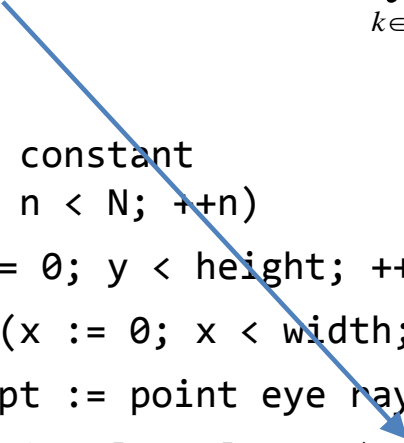
Monte Carlo
Probability

A Naïve Path Tracer (3/3)

$$L(i \rightarrow j) = L^e(i \rightarrow j) + \int_{k \in S} L(k \rightarrow i) f_r(k \rightarrow i \rightarrow j) G(k \rightarrow i) dk$$

Main:

```
N := a large constant
for (n := 0; n < N; ++n)
  for (y := 0; y < height; ++y)
    for (x := 0; x < width; ++x)
      pt := point eye ray intersects scene
      pixel[x, y] += L(pt, eye) / N
```



Is This Correct?

- ▶ A useful question to ask when you start out writing path tracers
- ▶ Our path tracer estimates the following:

$$E[L(i \rightarrow j)] = E \left[L^e(i \rightarrow j) + \frac{L(k \rightarrow i) f_r(k \rightarrow i \rightarrow j) G(k \rightarrow i)}{\text{PDF}(k)} \right] \quad \text{Russian roulette omitted for brevity}$$

$$= E[L^e(i \rightarrow j)] + E \left[\frac{L(k \rightarrow i) f_r(k \rightarrow i \rightarrow j) G(k \rightarrow i)}{\text{PDF}(k)} \right] \quad \begin{array}{l} L^e \text{ is a constant,} \\ \text{so } E[L^e] = L^e \end{array}$$

$$= L^e(i \rightarrow j) + E \left[\frac{L(k \rightarrow i) f_r(k \rightarrow i \rightarrow j) G(k \rightarrow i)}{\text{PDF}(k)} \right] \quad \begin{array}{l} E \left[\frac{f(k)}{\text{PDF}(k)} \right] \text{ is the definition of} \\ \text{Monte Carlo integration} \end{array}$$

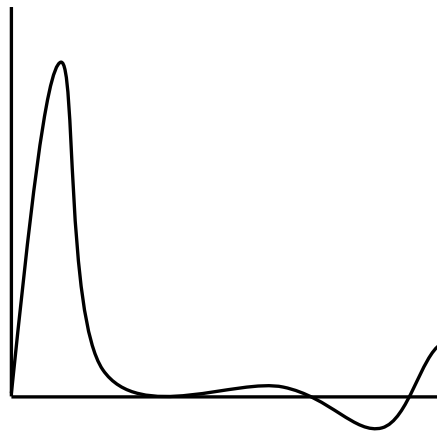
$$= L^e(i \rightarrow j) + \int_k L(k \rightarrow i) f_r(k \rightarrow i \rightarrow j) G(k \rightarrow i) dk \quad \text{Q.E.D.}$$

How Quickly Does This Converge?

- ▶ *Slowly*
 - ▶ Low chance of randomly shooting a ray towards a light, for example
 - ▶ Most rays gather little to no light
 - ▶ Need to average many samples to get good results
 - ▶ Need to wait a long time
-
- ▶ How can we modify the algorithm to converge more quickly?

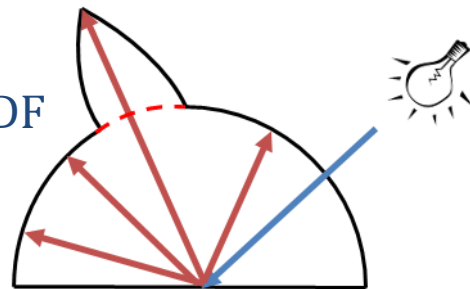
Importance Sampling (1/2)

- ▶ Consider a function that looks like this:
- ▶ We sample uniformly along the x axis
- ▶ But the spike contributes more to the integral than the rest of the function
- ▶ \therefore Most samples will give bad estimates of the area under the curve
- ▶ \therefore Will take a long time to converge to correct answer
- ▶ Lighting equation looks like this!
 - ▶ Spikes from lights and brighter objects, rest of the scene uniform, dimmer



Importance Sampling (2/2)

- ▶ Fix by sampling more often near the spikes in the function
 - ▶ Called **Importance Sampling**
- ▶ In terms of Monte Carlo rendering
 - ▶ Recursive ray direction chosen by sampling random point on hemisphere
 - ▶ Prefer rays that point towards light sources
 - ▶ Prefer rays that point along specular highlight spike in BRDF
 - ▶ Many other heuristics
 - ▶ Makes PDF (and, thus, $\text{PDF}(k)$ term) more complex



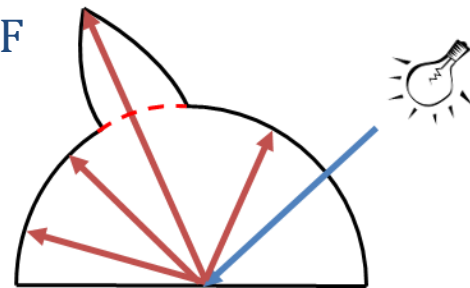
Total Scattering Distribution ($BRDF$)

Some Importance Sampling Strategies (1/2)

- ▶ Favor directly sampling light sources
 - ▶ (since lights represent a spike in the incoming light intensity)
 - ▶ ‘Allocate’ part of the probability space of X to sampling direct light sources
 - ▶ With some fixed probability p_{direct} ($= 0.1$ perhaps),
 - ▶ Trace a ray directly to a light
 - ▶ $PDF(k) = p_{direct} \times (\text{probability of choosing light}) \times (\text{probability of choosing point on light, for area lights})$
 - ▶ With probability $(1 - p_{direct})$,
 - ▶ Pick k like we did before
 - ▶ $PDF(k) = (1 - p_{direct}) \times (\text{probability of picking point on hemisphere, like before})$

Some Importance Sampling Strategies (2/2)

- ▶ Use the BRDF as the PDF
 - ▶ Favor selecting hemisphere points that correspond to BRDF spikes
 - ▶ More likely to trace a ray through the spike of the BRDF
 - ▶ Rays tend to lose less brightness due to BRDF
 - ▶ Math can be subtle
 - ▶ PDFs must integrate to 1
 - ▶ BRDFs don't necessarily integrate to 1
 - ▶ Need to divide BRDF by $\int \text{BRDF}$ to obtain a PDF weighted by BRDF



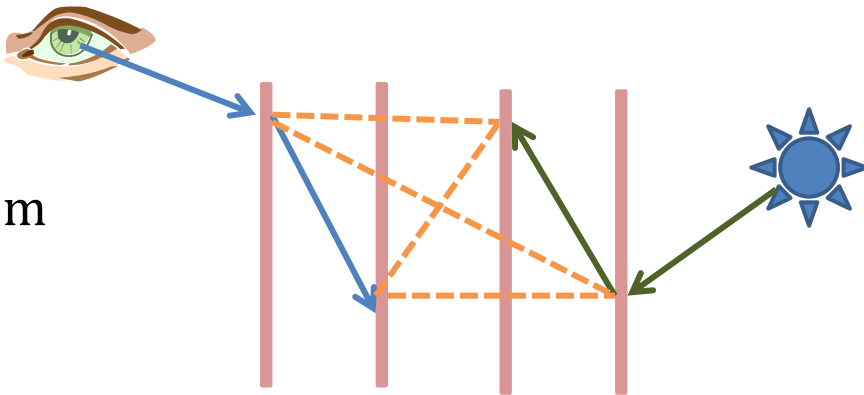
Total Scattering Distribution (*BRDF*)

Bidirectional Path Tracing (1/2)

- ▶ Much research into Monte Carlo techniques that converge quickly
- ▶ One example is Bidirectional Path Tracing (BDPT)
 - ▶ Extension of normal path tracing, by Lafortune and Willems¹
 - ▶ Like path tracing, converges to correct evaluation of rendering equation
 - ▶ Generally converges faster than conventional path tracing
- ▶ Core idea: forward map *and* back map (meet in the middle)
 - ▶ This isn't the last time we'll see this idea

Bidirectional Path Tracing (2/2)

- ▶ First, trace many random paths from a light source through the scene
- ▶ Second, trace many random paths from the eye through the scene
- ▶ Finally, send luminance from each hop in the light path to each hop in the eye path (being sure to check for occlusion)
- ▶ Pro: every hop gets direct lighting contribution
- ▶ Con: computing the probability to divide by is trickier



The red bars represent the set of points in a scene.

The green arrows represents a light path.

The blue arrows represent an eye path.

The orange lines represent all possible splices between the two sets.

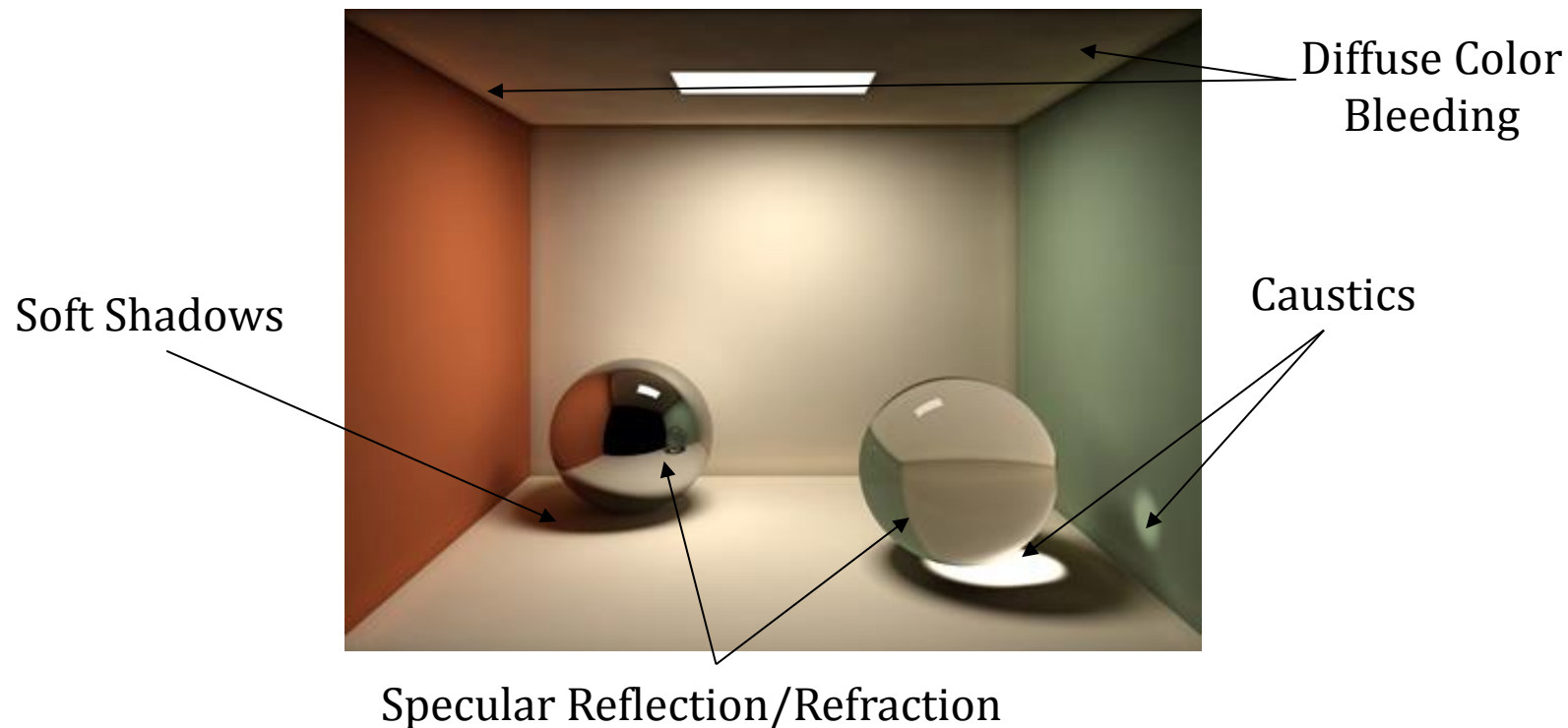
Photon Mapping

- ▶ Invented by Henrik Wann Jensen, 1995
- ▶ Extension of recursive raytracing
- ▶ Handles diffuse inter-object reflection and caustics
- ▶ Idea: store all indirect illumination in a three-dimensional **photon map**
 - ▶ A photon contains information about intensity, color, direction of a packet of light
 - ▶ A photon map contains photons organized in 3D space (usually a kd-tree)
- ▶ Photons are sampled during raytracing to collect inter-object reflection



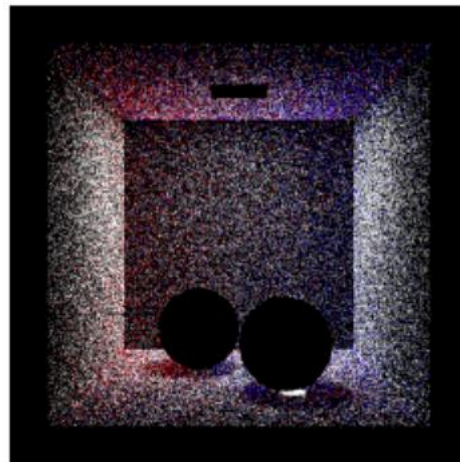
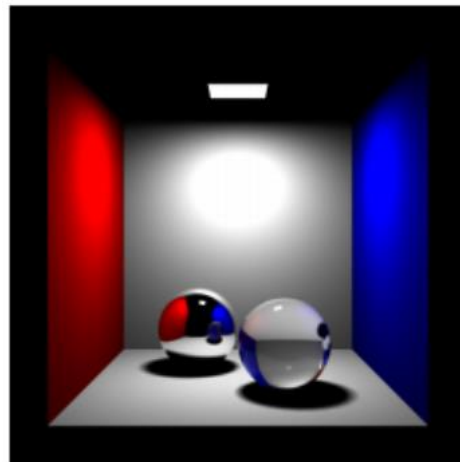
Image: Henrik Wann Jensen

Photon Mapping: Supported Lighting Effects



Generating the Photon Map

- ▶ Generate photon map by shooting photons into scene, randomly bouncing around scene (like path tracing)
- ▶ While [not enough photons in photon map]
 - ▶ Shoot a photon from a light into the scene
 - ▶ Where the photon collides
 - ▶ Store the photon in the photon map at that point (light intensities, direction, etc)
 - ▶ Determine whether photon is reflected or absorbed (Russian Roulette)
 - ▶ If reflected, attenuate photon energy according to BRDF and shoot photon back into scene
 - ▶ Repeat until photon is absorbed (or for a certain number of bounces)



Rendering with Photon Maps

- ▶ Raytrace normally
- ▶ To compute lighting at a point
 - ▶ Cast shadow ray, compute direct lighting, cast reflection ray, etc
 - ▶ Sample photon map
 - ▶ For some fixed k
 - ▶ Find the k photons in the photon map closest to the intersection point
 - ▶ Compute the illumination received from each of the k photons
 - ▶ Sum illumination from
 - ▶ Direct lighting
 - ▶ Perfect specular reflection (from raytracing)
 - ▶ Indirect lighting (from photon map)

Caustic Maps

- ▶ Since photons are shot into scene randomly, some areas may be under-represented
 - ▶ Especially problematic with caustics
- ▶ Solution
 - ▶ Split photon map into two photon maps
 - ▶ Store 'general' photons in one, caustic photons in the other
 - ▶ Caustic photon: any photon that passed through at least one specular reflection or refraction, before finally landing on a diffuse surface
 - ▶ Enforce a minimum size on each photon map
 - ▶ Sample both when rendering (sum contributions)



Metropolis Light Transport (MLT) (1/2)

- ▶ Another Monte Carlo rendering algorithm
- ▶ Able to render difficult scenes quickly
- ▶ Notorious for being difficult to implement correctly
- ▶ Uses Metropolis-Hastings Sampling
 - ▶ Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller, 1953
 - ▶ A form of importance sampling
 - ▶ Samples chosen so that distribution eventually proportional to the function being integrated



Scene entirely lit from a light *behind* the door.

Metropolis Light Transport (MLT) (2/2)

- ▶ Algorithm published by Eric Veach and Leonidas Guibas
- ▶ Conceptually: Final image = $\int_{\mathbf{P}}$ (Contribution of a single path)
 - ▶ Where \mathbf{P} is the space of all possible light paths through the scene
 - ▶ Integrate over all light paths from a light to the camera
 - ▶ Reformulates Kajiya rendering equation as a pure integral
- ▶ MLT amortizes costs of finding these paths
 - ▶ After algorithm finds a successful path,
high probability of perturbing that path locally
 - ▶ Perturbed path also likely to contribute to image

Interested in Learning More?

- ▶ Chapters 30-32 in the book discuss several of these techniques

- ▶ CS224 teaches them!

- ▶ Shameless plug!

- ▶ :D



Skipper John F. Hughes