State of the Art in Monte Carlo Global Illumination

SIGGRAPH 2004 Course 4 (Full day)

Organizers

Philip Dutré
Department of Computer Science
Katholieke Universiteit Leuven

Henrik Wann Jensen
Computer Science and Engineering
University of California, San Diego

Lecturers

Jim Arvo
Information and Computer Science
University of California, Irvine

Kavita Bala
Computer Science Department
Cornell University

Philippe Bekaert
Expertise Center for Digital Media
University of Limburg

Steve Marschner
Computer Science Department
Cornell University

Matt Pharr
NVIDIA Corporation

Course Description

Realistic image synthesis is increasingly important in areas such as entertainment (movies, special effects and games), design, architecture and more. A common trend in all these areas is the quest for more realistic images of increasingly complex models. Monte Carlo global illumination algorithms are the only methods that can handle this complexity. Recent advances in algorithms and compute power has made Monte Carlo algorithms very practical and a natural choice for most problems.

The purpose of this course is to impart upon the attendees a thorough understanding of the principles of Monte Carlo path tracing methods, as well as a detailed overview of the most recently developed methods.

Part 1 will cover the fundamentals of realistic image synthesis (radiometry, rendering equation). Part 2 focuses on the mathematical tools needed to compute integrals using Monte Carlo sampling. Parts 3 and 4 cover specific algorithms that have proven very useful in global illumination rendering.

Prerequisites

A basic understanding of classic photo-realistic rendering algorithms, such as basic ray tracing and radiosity is assumed. Knowledge of probability theory will be very helpful. Some familiarity with transport equations and radiometry will be useful, but is not necessary.

Course Syllabus

The course is subdivided in 4 main parts, which can be attended seperately, if attendees wish to do so. The first part, "Fundamentals", explains the physical concepts describing the global illumination problem. The second part covers an important tool for global illumination algorithms, Monte Carlo integration. Direct illumination techniques are used to illustrate these principles. The third and fourth parts highlight specific algorithms that are commonly used in photo-realistic rendering.

8:30: Part 1: "Fundamentals"

Introduction (10 minutes): *Philip Dutré*

Radiometry (40 minutes): *Steve Marschner*

- Nature of light, and how to model light in computer graphics
- Radiometric quantities (radiance, irradiance, flux)
- Bidirectional reflectance distribution functions models used in global illumination rendering (Phong model, Lafortune model, Cook-Torrance model)
- Transport equations and rendering equation; particle model of light

The Rendering Equation (55 minutes): *Philip Dutré*

- Various forms of the rendering equation
- How to think about the rendering equation?
- Path integral formulation
- General path generation strategies
- Design strategies for global illumination algorithms

10:15: Break

10:30: Part 2 "Monte Carlo and Sampling"

Fundamentals of Monte Carlo Integration (60 minutes): *Jim Arvo*

- How does Monte Carlo integration work?
- Basic sampling techniques
- Importance sampling
- Stratified sampling
- Good sample distributions
- Efficient sampling of high-dimensional problems
- Quasi-Monte Carlo vs. Monte Carlo
- Sampling of geometry and reflection models
- Russian Roulette

Direct Illumination (45 minutes): *Kavita Bala*

- Sampling of light sources
- Special types of light sources
- Efficient sampling of many light sources

12:15: Lunch

1:45: Part 3: "Global Illumination Algorithms 1" (14:00 - 15:45)

Metropolis Sampling (45 minutes): *Matt Pharr*

- What is Metropolis sampling?
- Sampling a BRDF
- Sampling of motion blur
- Metropolis light transport

Biased Monte Carlo Ray Tracing (60 minutes): *Henrik Jensen*

- Filtering techniques
- Irradiance caching
- Photon mapping

3:30: Break

3:45: Part 4: "Global Illumination Algorithms 2"

Stochastic Radiosity (60 minutes): *Philippe Bekaert*

- Radiosity equations - how to derive them from the rendering equation
- Stochastic relaxation methods for radiosity
- Random walk methods (particle tracing, density estimation)
- Variance reduction techniques and hierarchical refinement

Interactive Techniques (40 minutes): *Kavita Bala*

- Recent advances in interactive global illumination algorithms

Summary Statement (5 minutes): *Philip Dutré*

Presenters

Jim Arvo

Information and Computer Science
408D Computer Science Building
University of California, Irvine
Irvine, CA 92697-3425
USA
arvo@uci.edu
URL: http://www.ics.uci.edu/~arvo/
Phone +1 (949) 824-9236

James Arvo is an Associate Professor at the University of California, Irvine. Before, he has been an Associate Professor of Computer Science at Caltech since 1995. From 1990 to 1995 he was a senior researcher in the Program of Computer Graphics at Cornell University. He received a B.S. in Mathematics from Michigan Technological University, an M.S. in Mathematics from Michigan State University, and a Ph.D. in Computer Science from Yale University in 1995.

Kavita Bala

Computer Science Department
Cornell University
5142 Upson Hall
Ithaca NY 14853
USA
kb@cs.cornell.edu
URL : http://www.cs.cornell.edu/~kb/
Phone: +1 (607) 255 1383

Kavita Bala is an assistant professor in the Computer Science Department and Program of Computer Graphics at Cornell University. Before Cornell, Bala received her doctorate degree from MIT for her research on 4D radiance interpolants. Her research interests include interactive global illumination, shadow algorithms, point-based rendering, display representations, and image-based rendering and texturing. She is co-author of the book "Advanced Global Illumination" (A K Peters 2003).

Philippe Bekaert

Expertise Center for Digital Media
University of Limburg
Wetenschapspark 2
B-3590 Diepenbeek
BELGIUM
Philippe.Bekaert@luc.ac.be
Phone: +32 (11) 26 84 11

Philippe Bekaert is an associate professor at the Expertise Center for Digital Media at the University of Limburg, Belgium. Before, he was a post-doctoral research fellow at the Max Planck Institut fuer Informatik in Saarbruecken, Germany. He received a Ph.D. in Computer Science, titled "Hierarchical and Stochastic Algorithms for Radiosity" (K.U.Leuven, 1999). He is author or co-author of numerous papers on stochastic radiosity. His current research interests include interactive global illumination and the Monte Carlo method. He is co-author of the book "Advanced Global Illumination" (A K Peters, 2003).

Philip Dutré

Department of Computer Science - Katholieke Universiteit Leuven
Celestijnenlaan 200A
B-3001 Leuven
BELGIUM
phil@cs.kuleuven.ac.be
URL: http://www.cs.kuleuven.ac.be /~phil/
Phone: +32 (16) 32 76 67

Philip Dutré is professor at the Department of Computer Science at the Katholieke Universiteit Leuven, Belgium. Previously, he was a post-doctoral research associate in the Program of Computer Graphics at Cornell University. He received a Ph.D. in Computer Science titled "Mathematical Frame-works and Monte Carlo Algorithms for Global Illumination in Computer Graphics" (K.U.Leuven, 1996). His current research interests include real-time global illumination, image-based relighting and goal-based rendering. He is involved in teaching undergraduate and graduate courses covering computer graphics and rendering algorithms. He is co-author of the book "Advanced Global Illumination" (A K Peters, 2003).

Henrik Jensen

Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114
USA
henrik@graphics.ucsd.edu
URL: http://graphics.ucsd.edu/~henrik
Phone: +1 (858) 822 0415
Fax: +1 (650) 723 0033

Dr. Henrik Wann Jensen is an assistant professor at the University of California at San Diego, where he is establishing a computer graphics lab with a research focus on realistic image synthesis, global illumination, rendering of natural phenomena, and appearance modeling. He is the author of "Realistic Image Synthesis using Photon Mapping," AK Peters 2001. Prior to coming to UCSD, he was a Postdoc at Stanford University and Massachusetts Institute of Technology. He received his M.Sc. and Ph.D. in Computer Science from the Technical University of Denmark in 1996 for developing the photon mapping method.

Steve Marschner

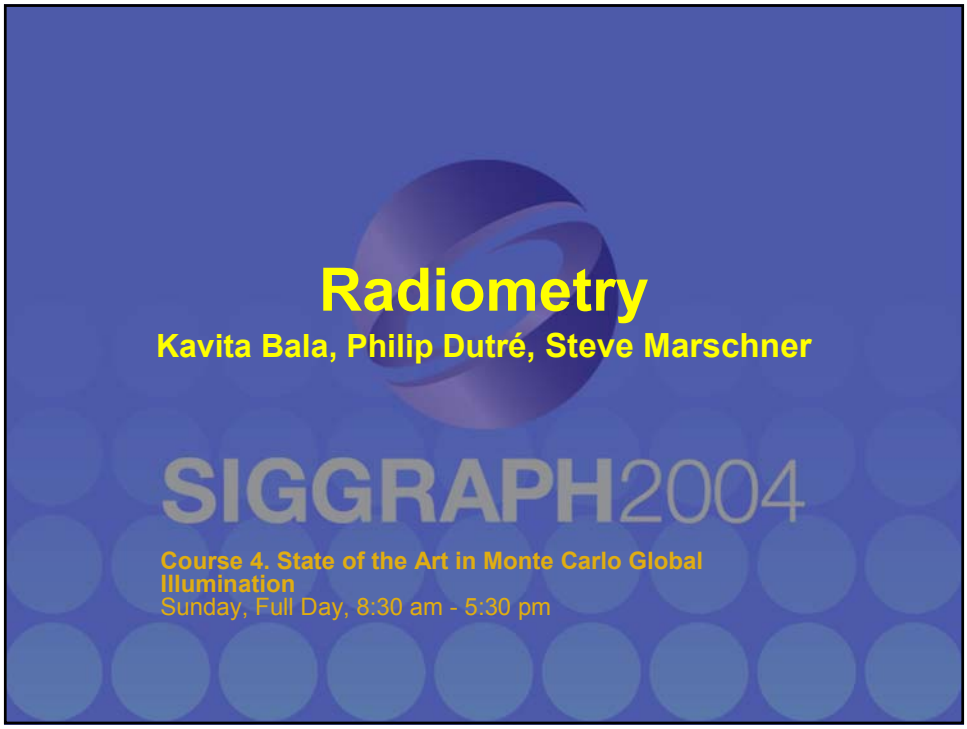
Computer Science Department
Cornell University
5159 Upson Hall
Ithaca, NY 14853
USA
srm@cs.cornell.edu
URL: http://www.cs.cornell.edu/~srm/
Phone +1 (607) 255 8367
Fax: +1 (607) 255 4428

Steve Marschner is Assistant Professor of Computer Science at Cornell University, where he is conducting research into light interaction with materials. He obtained his Sc.B. from Brown University in 1993 and his Ph.D. from Cornell in 1998. He held research positions at Hewlett-Packard Labs, Microsoft Research, and Stanford University before joining Cornell in 2002. He has delivered numerous presentations, including papers at IEEE Visualization '94, the Eurographics Rendering Workshop in 1999 and 2000, and SIGGRAPH 2003; and SIGGRAPH courses in 2000 ("Image-based surface details"), 2001 (measuring surface reflection), 2002 (inverse rendering), and 2003 (Monte Carlo rendering; as a substitute).

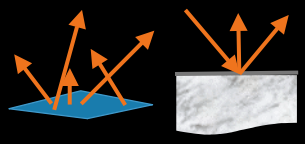
Matt Pharr

NVIDIA
664 Douglass St
San Francisco, CA 94114
USA
matt@pharr.org
URL: http://graphics.stanford.edu/~mmp/

Matt Pharr is a member of the technical staff at NVIDIA, where he works on high-quality interactive graphics, programmable shading, and Cg language features in the Software Architecture group. Previously, he was a co-founder of Exluna, which developed off-line rendering software, and was investigating advanced shading algorithms for graphics hardware. He was a PhD student in the Stanford Graphics Lab, where he researched systems issues for rendering, theoretical foundations of rendering, and published a series of SIGGRAPH papers on these topics. With Greg Humphreys, he is the author of the book Physically Based Rendering: From Theory to Implementation.


Radiometry
Kavita Bala, Philip Dutré, Steve Marschner
SIGGRAPH2004
Course 4. State of the Art in Monte Carlo Global Illumination
Sunday, Full Day, 8:30 am - 5:30 pm

Outline



- Light Model
- Radiometry
- Materials: Interaction with light

- Rendering equation

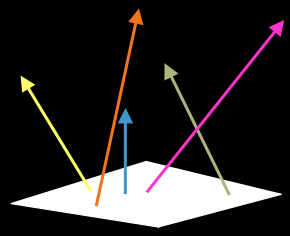
Light Models

- Geometric Optics
  - Emission, Reflection / Refraction, Absorption
- Wave Model

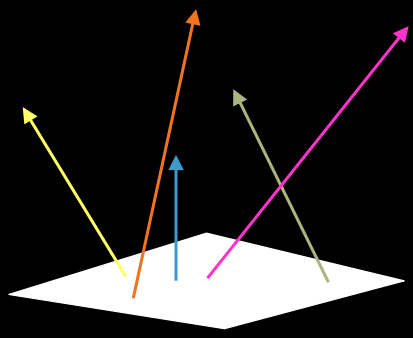
  - Maxwell's Equations
  - Object size comparable to wavelength
  - Diffraction & Interference, Polarization
- Quantum Model
  - Fluorescence, Phosphorescence

Geometric Optics: Properties

- Light travels in straight lines
- Rays do not interact with each other
- Rays have color(wavelength), intensity



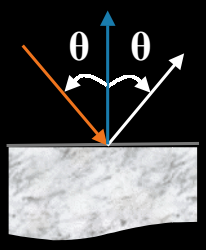
Emission



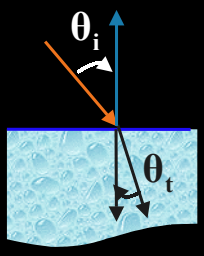
Reflections/Refractions

- **Interface between 2 materials**

reflection



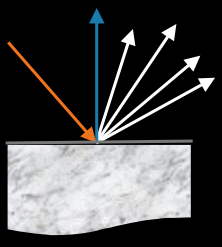
refraction



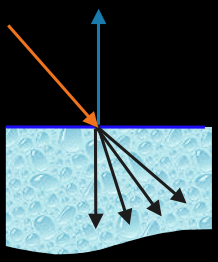
- **Specular reflections and refractions**
  - One direction

Realistic Reflections/Refractions

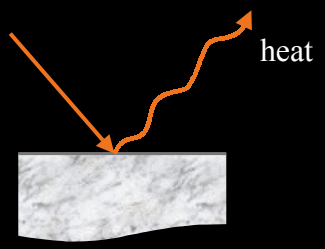
reflection



refraction



Absorption



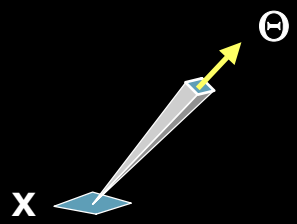
Geometric Optics: other effects

- Participating Media

- Varying index of refraction

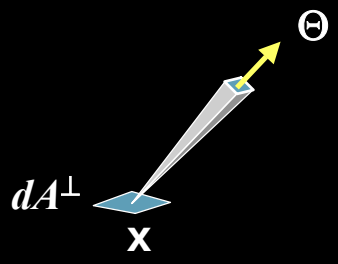
Radiometry: radiance

- **Radiometry: Measurement of light energy**
- **Radiance: *radiant energy density***
  - *x is position, Θ is direction*
  - Varies with position and direction: 5D function



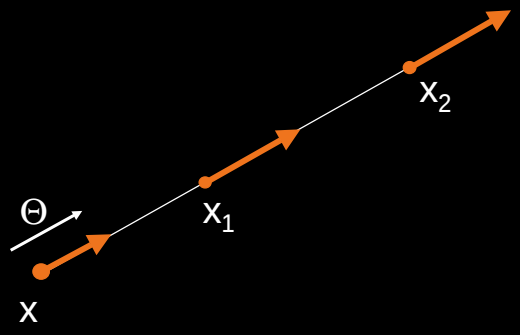
Radiance

- **Radiance $L(x \rightarrow \Theta)$ is the power**
  - **Per unit projected surface area**
  - **Per unit solid angle**
  - $$L(x \rightarrow \Theta) = \frac{d^2 P}{dA^{\perp} d\omega_{\Theta}}$$
  - units: Watt / m^2.sr
  - Wavelength dependence
- **Important quantity**



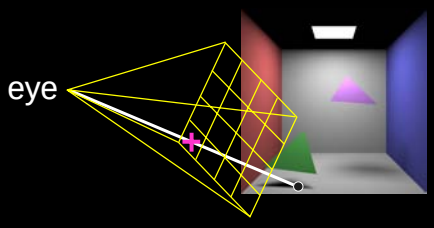
Why is radiance important?

- Invariant along a straight line (in vacuum)



Why is radiance important?

- Response of a sensor (camera, human eye) is proportional to radiance

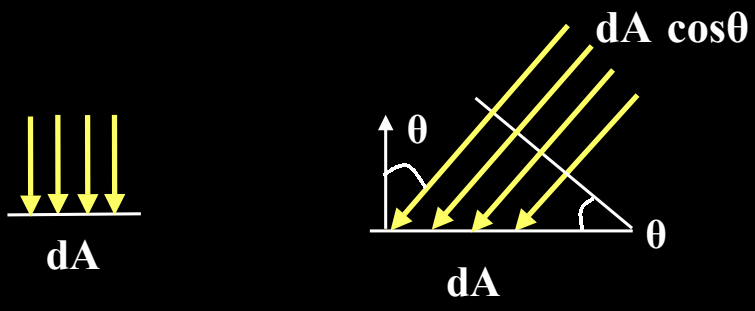


- Pixel values in image are proportional to radiance received from that direction

Radiance: Projected area

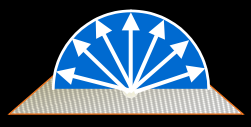
- $L(x \to \Theta) = \dfrac{d^2 P}{dA^{\perp} d\omega_{\Theta}}$

- Why per unit projected surface area?



Example: Diffuse emitter

- Diffuse emitter: light source with equal radiance everywhere



$$L(x \to \Theta) = \frac{d^2 P}{dA^{\perp} d\omega_{\Theta}}$$

$$P = \int_{Area} \int_{Solid\ Angle} L(x \to \Theta) \cdot \cos\theta \cdot d\omega_{\Theta} \cdot dA$$

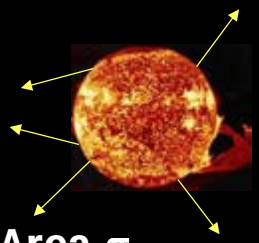
$$= L \int_{Area} dA \int_{Solid\ Angle} \cos\theta \cdot d\omega_{\Theta}$$

$$= L \cdot Area \cdot \pi$$

Sun Example: radiance

Power: 3.91×10^{26} W

Surface Area: 6.07×10^{18} m^2

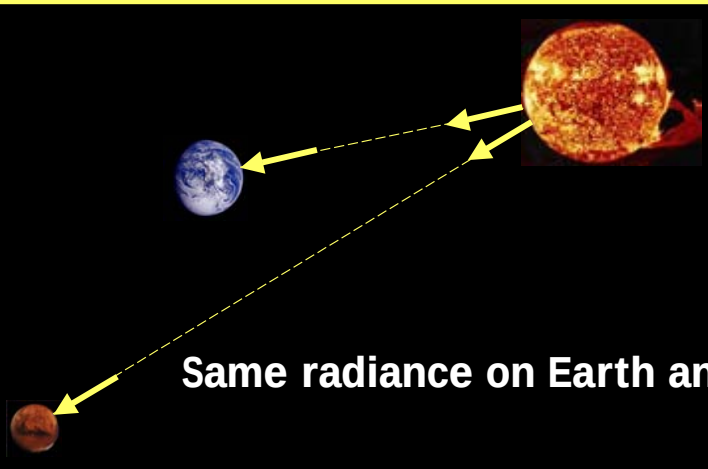


Power = Radiance.Surface Area.π

Radiance = Power/(Surface Area.π)

Radiance = 2.05×10^7 W/ m^2.sr

Sun Example

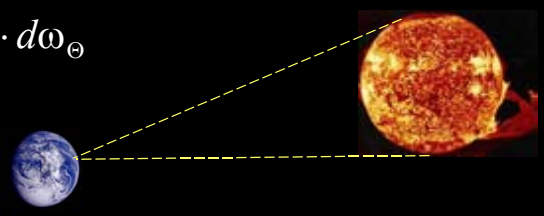


Same radiance on Earth and Mars?

Sun Example: Power on Earth

Power reaching earth on a 1m² square:

$$P = L \int_{Area} dA \int_{Solid\ Angle} \cos\theta \cdot d\omega_{\Theta}$$

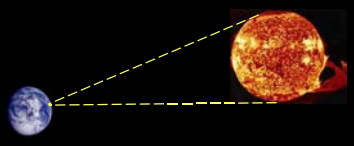


Assume $\cos\theta$ = 1 (sun in zenith)

$$P = L \int_{Area} dA \int_{Solid\ Angle} d\omega_{\Theta}$$

Sun Example: Power on Earth

Power = Radiance.Area.Solid Angle

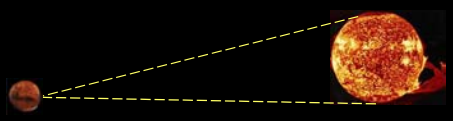


Solid Angle = Projected $Area_{Sun}$/$(distance_{earth\_sun})^2$
= 6.7 10^{-5} sr

P = (2.05 x 10^7 W/ m^2.sr) x (1 m^2) x (6.7 10^{-5} sr)
= 1373.5 Watt

Sun Example: Power on Mars

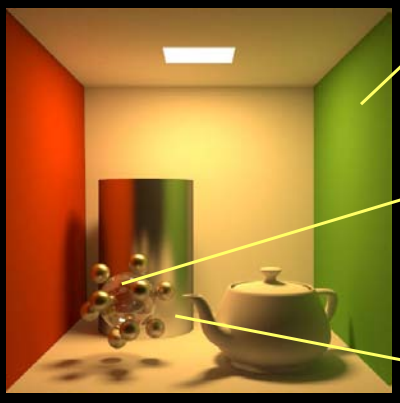
Power = Radiance.Area.Solid Angle

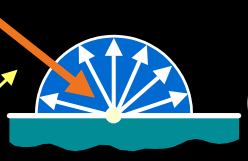


Solid Angle = Projected $Area_{Sun}$ / $(distance_{mars\_sun})^2$
= 2.92 10^{-5} sr

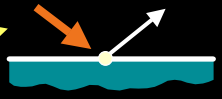
P = (2.05 x 10^7 W/ m^2.sr) x (1 m^2) x (2.92 10^{-5} sr)
= 598.6 Watt

Materials - Three Forms





Ideal diffuse (Lambertian)

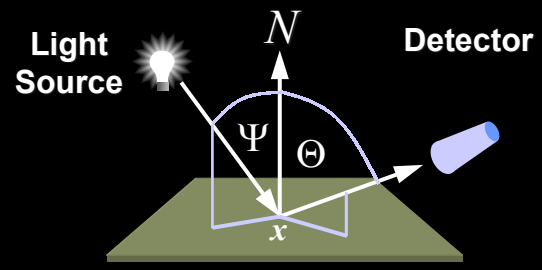


Ideal specular

Directional diffuse

BRDF

- **Bidirectional Reflectance Distribution Function**



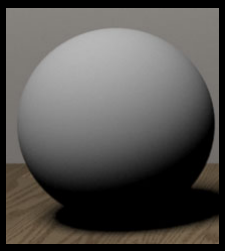
$$f_r(x, \Psi \rightarrow \Theta) = \frac{dL(x \rightarrow \Theta)}{dE(x \leftarrow \Psi)} = \frac{dL(x \rightarrow \Theta)}{L(x \leftarrow \Psi)\cos(N_x, \Psi)d\omega_\Psi}$$

BRDF special case: ideal diffuse

Pure Lambertian

$$f_r(x, \Psi \rightarrow \Theta) = \frac{\rho_d}{\pi}$$

$$0 \leq \rho_d \leq 1$$



Properties of the BRDF

- **Reciprocity:**

$$f_r(x, \Psi \rightarrow \Theta) = f_r(x, \Theta \rightarrow \Psi)$$

- **Therefore, notation:** $f_r(x, \Psi \leftrightarrow \Theta)$

Properties of the BRDF

- **Bounds:**

$$0 \leq f_r(x, \Psi \leftrightarrow \Theta) \leq \infty$$

- **Energy conservation:**

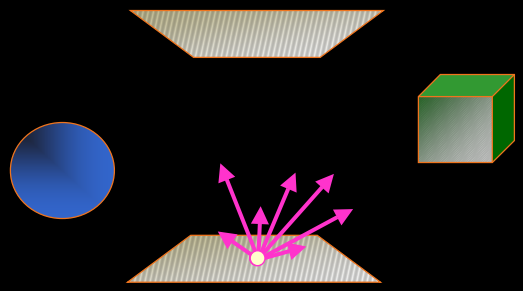
$$\forall \Psi \quad \int_{\Theta} f_r(x, \Psi \leftrightarrow \Theta) \cos(N_x, \Theta) d\omega_{\Theta} \leq 1$$

Light Transport

- Goal:
  - Describe radiance distribution in the scene
- Assumptions:
  - Geometric Optics
  - Achieve steady state

Radiance represents equilibrium

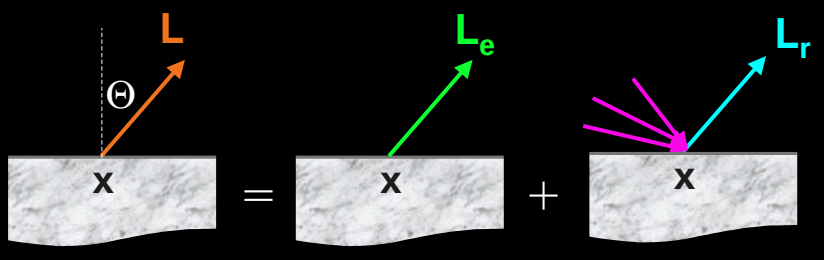
- Radiance values at all points in the scene and in all directions expresses the equilibrium
- 4D function: only on surfaces



Rendering Equation (RE)

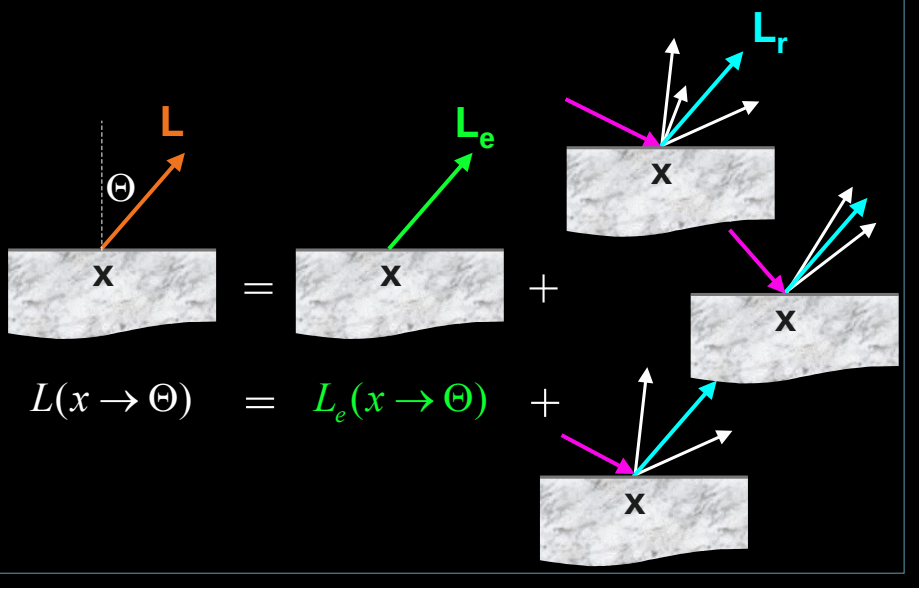
- RE describes energy transport in a scene
- Input:
  - light sources
  - geometry of surfaces
  - reflectance characteristics of surfaces
- Output: value of radiance at all surface points and in all directions

Rendering Equation



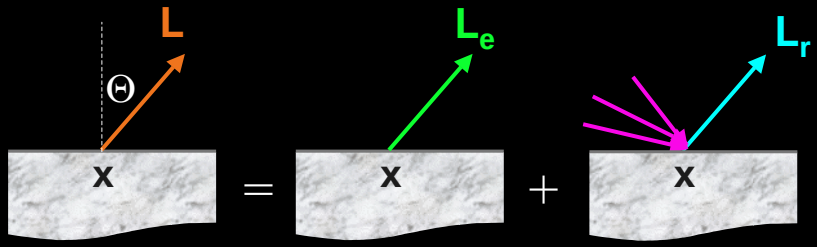
$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + L_r(x \rightarrow \Theta)$$

Rendering Equation



$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) +$$

Rendering Equation



$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{hemisphere} L(x \leftarrow \Psi) \cdots$$

Rendering Equation

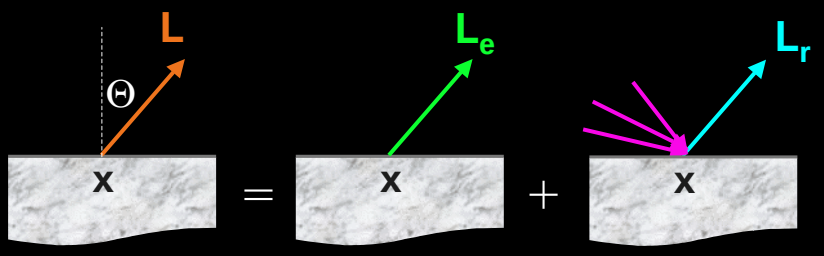
$$f_r(x, \Psi \leftrightarrow \Theta) = \frac{dL(x \rightarrow \Theta)}{dE(x \leftarrow \Psi)}$$

$$dL(x \rightarrow \Theta) = f_r(x, \Psi \leftrightarrow \Theta) dE(x \leftarrow \Psi)$$

$$dL(x \rightarrow \Theta) = f_r(x, \Psi \leftrightarrow \Theta) L(x \leftarrow \Psi) \cos(N_x, \Psi) d\omega_\Psi$$

$$L_r(x \rightarrow \Theta) = \int_{hemisphere} f_r(x, \Psi \leftrightarrow \Theta) L(x \leftarrow \Psi) \cos(N_x, \Psi) d\omega_\Psi$$

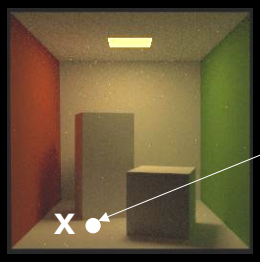
Rendering Equation



$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{hemisphere} L(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) \cos(\mathbf{N}_x, \Psi) d\omega_\Psi$$

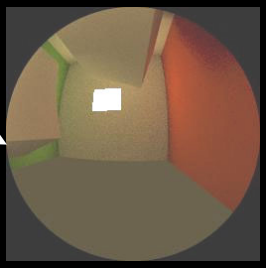
- **Applicable for each wavelength**

Rendering Equation



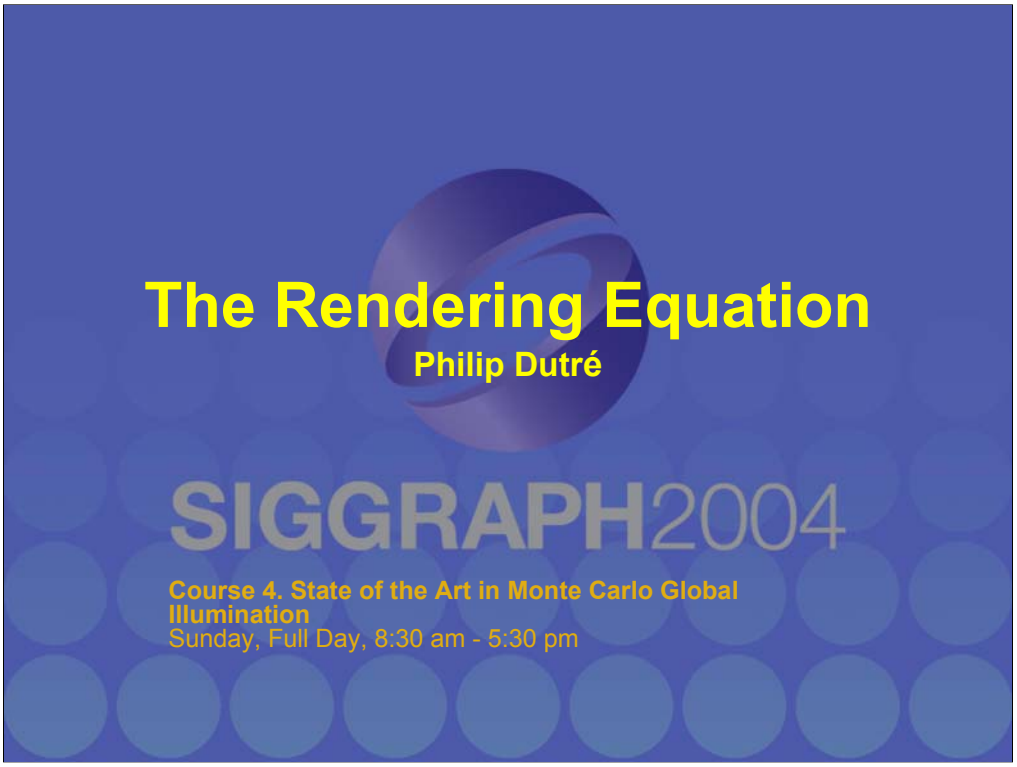
$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{hemisphere} L(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) \cos(\mathbf{N}_x, \Psi) d\omega_\Psi$$

incoming radiance



Summary

- **Geometric Optics**
- **Goal:**
  - to compute steady-state radiance values in scene
- **Rendering equation:**
  - mathematical formulation of problem


The Rendering Equation
Philip Dutré
SIGGRAPH2004
Course 4. State of the Art in Monte Carlo Global Illumination
Sunday, Full Day, 8:30 am - 5:30 pm

Overview

- Rendering Equation
- Path tracing
- Path Formulation
- Various path tracing algorithms

This part of the course will cover in detail the rendering equation and how to reason about it.

We will start by repeating a few concepts seen before, namely the definition of the BRDF.

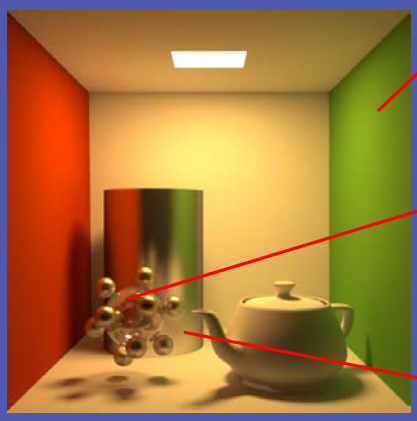
Light Transport

- Goal:
  - Describe radiance distribution in the scene
- Assumptions:
  - Geometric Optics
  - Achieve steady state (equilibrium)

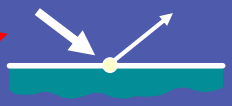
Rendering Equation (RE)

- RE describes energy transport in a scene
- Input:
  - **light sources**
  - **geometry of surfaces**
  - **reflectance characteristics of surfaces**
- Output: value of radiance at all surface points and in all directions

Materials



Ideal diffuse
(Lambertian)

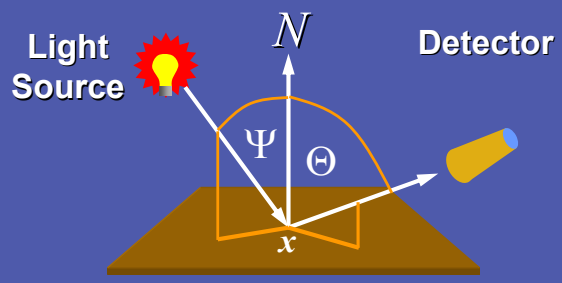


Ideal specular

Directional
Diffuse(“glossy”)

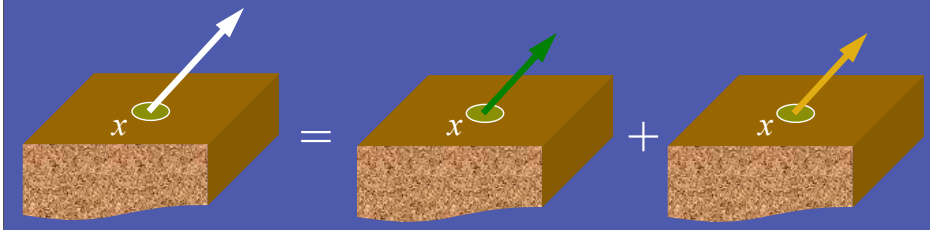
BRDF

- Bidirectional Reflectance Distribution Function



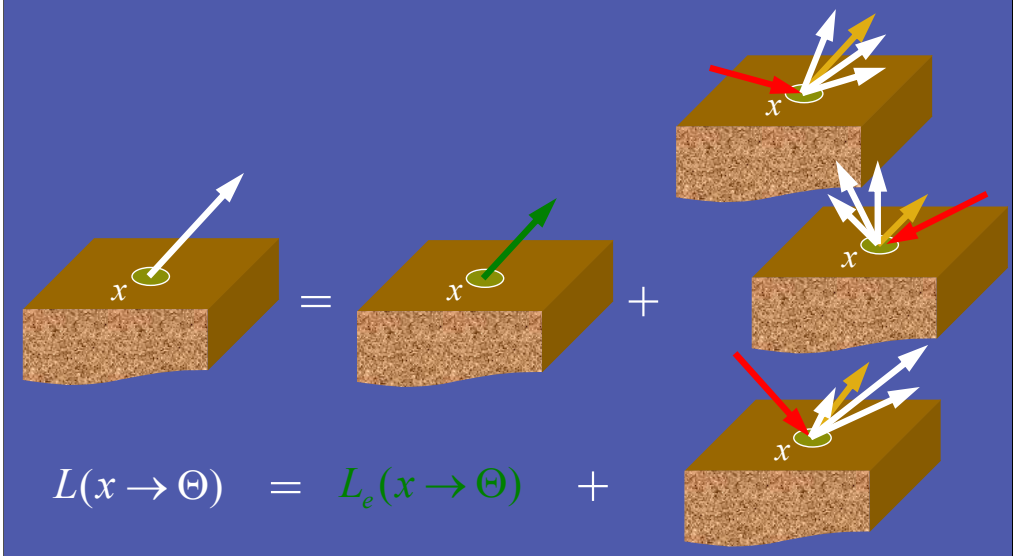
$$f_r(x, \Psi \rightarrow \Theta) = \frac{dL(x \rightarrow \Theta)}{dE(x \leftarrow \Psi)} = \frac{dL(x \rightarrow \Theta)}{L(x \leftarrow \Psi)\cos(N_x, \Psi)d\omega_\Psi}$$

Rendering Equation



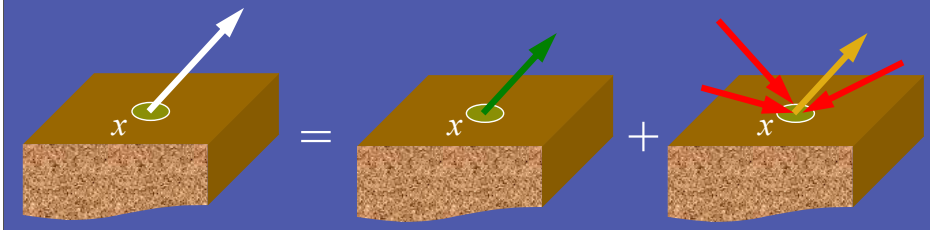
$$L(x \to \Theta) = L_e(x \to \Theta) + L_r(x \to \Theta)$$

Rendering Equation



$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) +$$

Rendering Equation



$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{hemisphere} L(x \leftarrow \Psi) \cdots$$

Rendering Equation

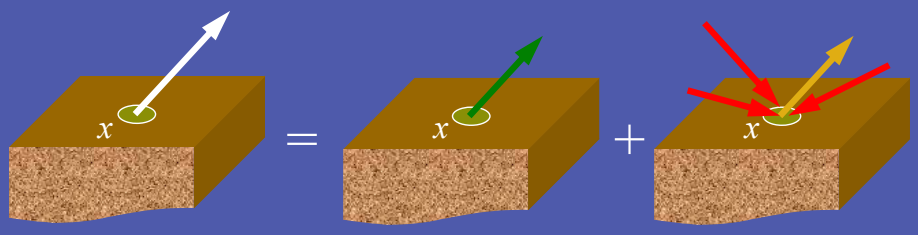
$$f_r(x, \Psi \leftrightarrow \Theta) = \frac{dL(x \rightarrow \Theta)}{dE(x \leftarrow \Psi)}$$

$$dL(x \rightarrow \Theta) = f_r(x, \Psi \leftrightarrow \Theta) dE(x \leftarrow \Psi)$$

$$dL(x \rightarrow \Theta) = f_r(x, \Psi \leftrightarrow \Theta) L(x \leftarrow \Psi) \cos(N_x, \Psi) d\omega_\Psi$$

$$L_r(x \rightarrow \Theta) = \int_{hemisphere} f_r(x, \Psi \leftrightarrow \Theta) L(x \leftarrow \Psi) \cos(N_x, \Psi) d\omega_\Psi$$

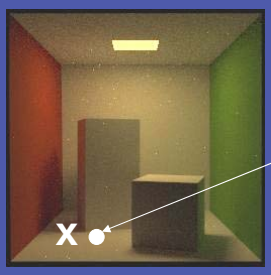
Rendering Equation



$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{hemisphere} L(x \leftarrow \Psi)\, f_r(x, \Psi \leftrightarrow \Theta) \cos(\mathbf{N}_x, \Psi)\, d\omega_\Psi$$

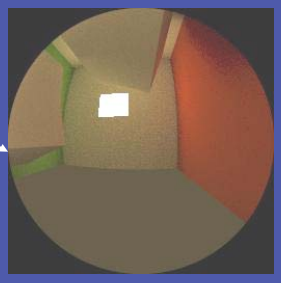
- **Applicable for each wavelength**

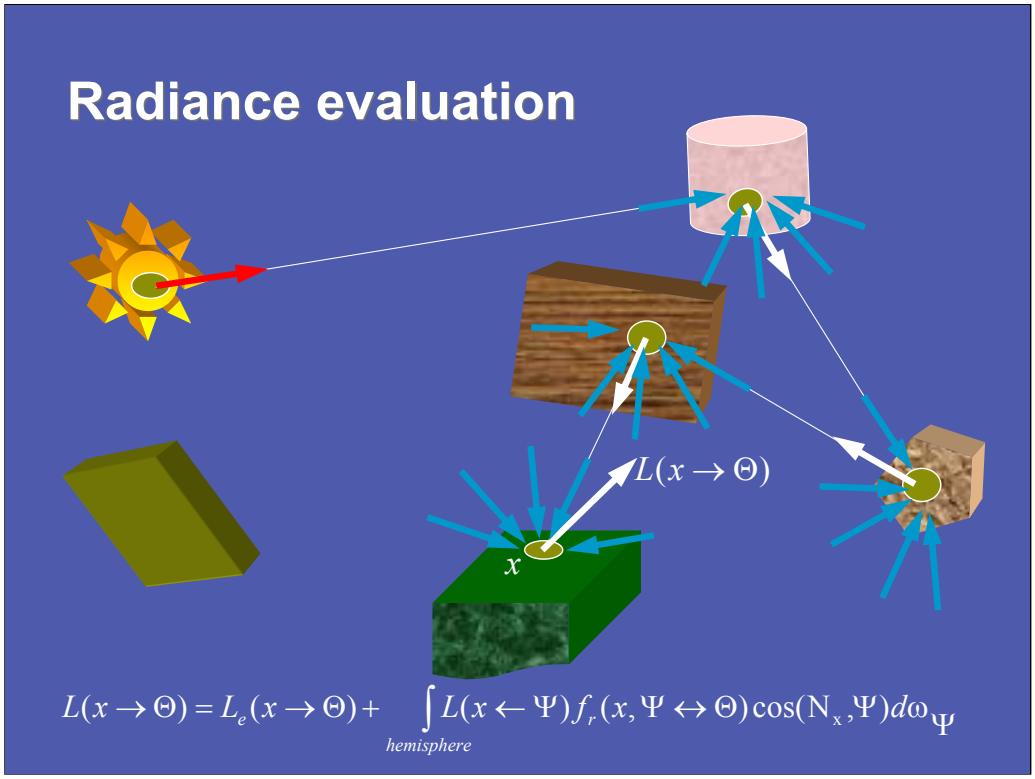
Rendering Equation



$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{hemisphere} L(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) \cos(\mathbf{N}_x, \Psi) d\omega_\Psi$$

incoming radiance



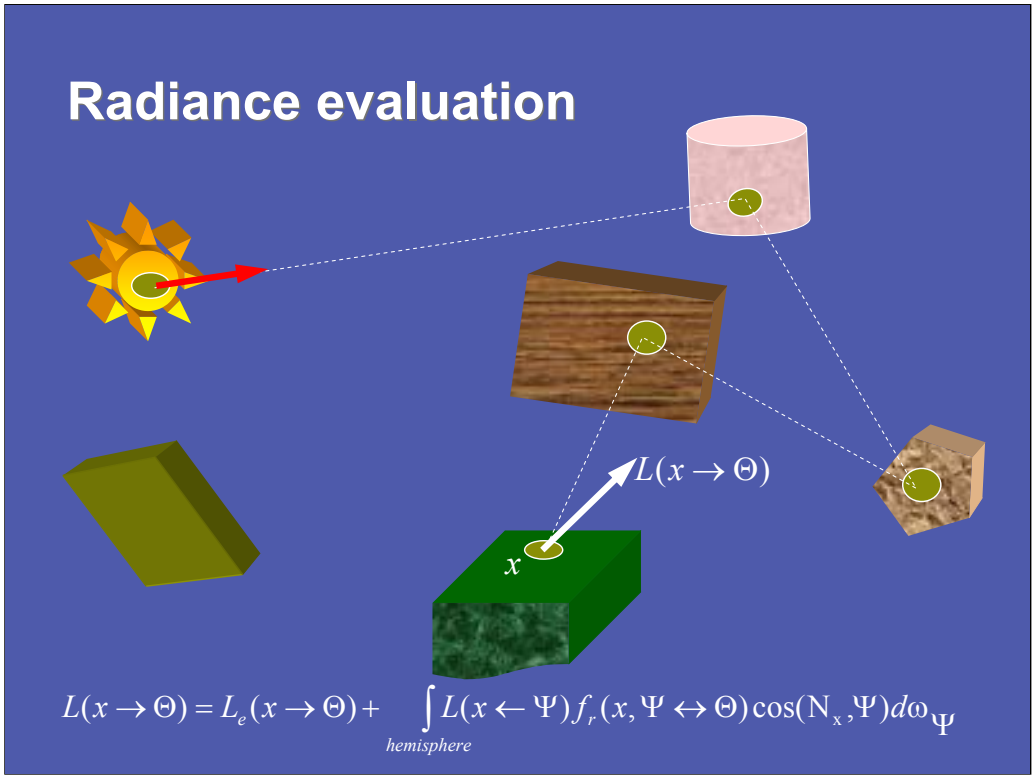


This is an illustration of the recursive nature of the rendering equation.

All radiance values incident at surface point x are themselves outgoing radiance values. We have to trace back the path they arrive from. This means tracing a ray from x in the direction of the incoming radiance. This results in some surface point, and the incoming radiance along the original direction now simply equals the outgoing radiance at this new point.

The problem is then stated recursively, since this new radiance value is also described exactly by the rendering equation.

This process will continue until the paths are traced back to the light source. Then we can pick up the self-emitted radiance, take into account all possible cosine factors and possible BRDF values along the path, perform the necessary integration at each surface point, to finally arrive at the original radiance value we are interested in.

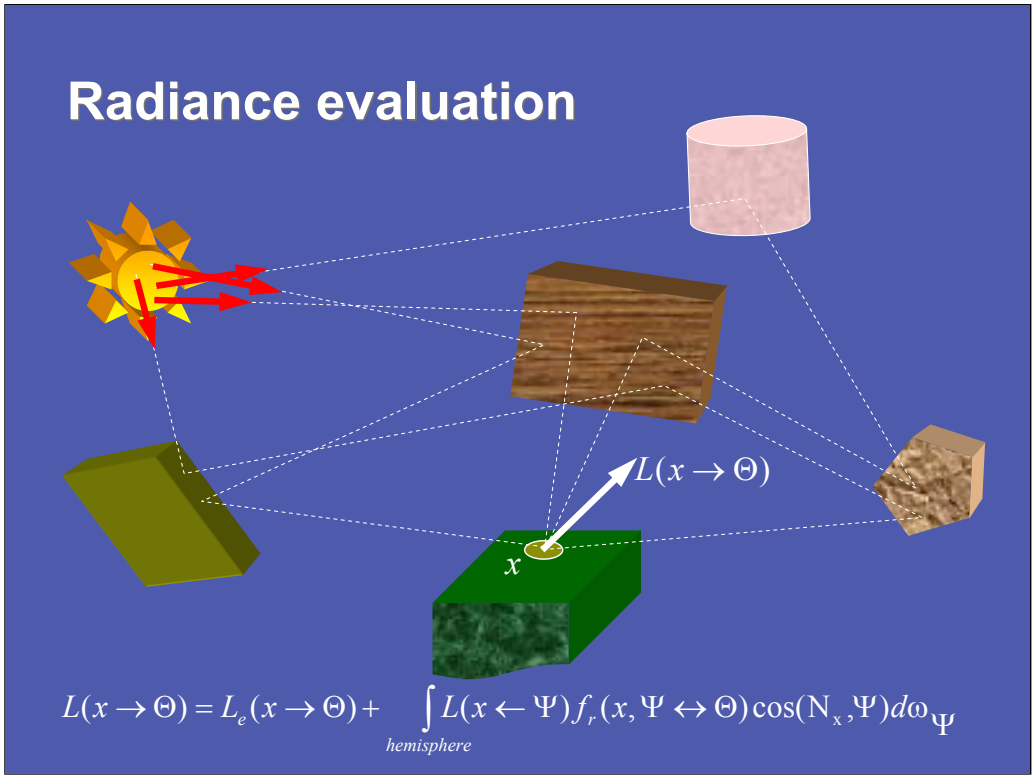


This is an illustration of the recursive nature of the rendering equation.

All radiance values incident at surface point x are themselves outgoing radiance values. We have to trace back the path they arrive from. This means tracing a ray from x in the direction of the incoming radiance. This results in some surface point, and the incoming radiance along the original direction now simply equals the outgoing radiance at this new point.

The problem is then stated recursively, since this new radiance value is also described exactly by the rendering equation.

This process will continue until the paths are traced back to the light source. Then we can pick up the self-emitted radiance, take into account all possible cosine factors and possible BRDF values along the path, perform the necessary integration at each surface point, to finally arrive at the original radiance value we are interested in.



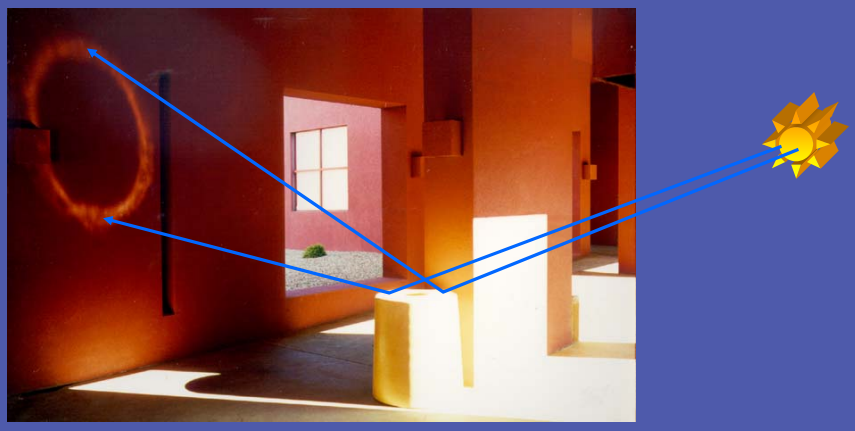
This is an illustration of the recursive nature of the rendering equation.

All radiance values incident at surface point x are themselves outgoing radiance values. We have to trace back the path they arrive from. This means tracing a ray from x in the direction of the incoming radiance. This results in some surface point, and the incoming radiance along the original direction now simply equals the outgoing radiance at this new point.

The problem is then stated recursively, since this new radiance value is also described exactly by the rendering equation.

This process will continue until the paths are traced back to the light source. Then we can pick up the self-emitted radiance, take into account all possible cosine factors and possible BRDF values along the path, perform the necessary integration at each surface point, to finally arrive at the original radiance value we are interested in.

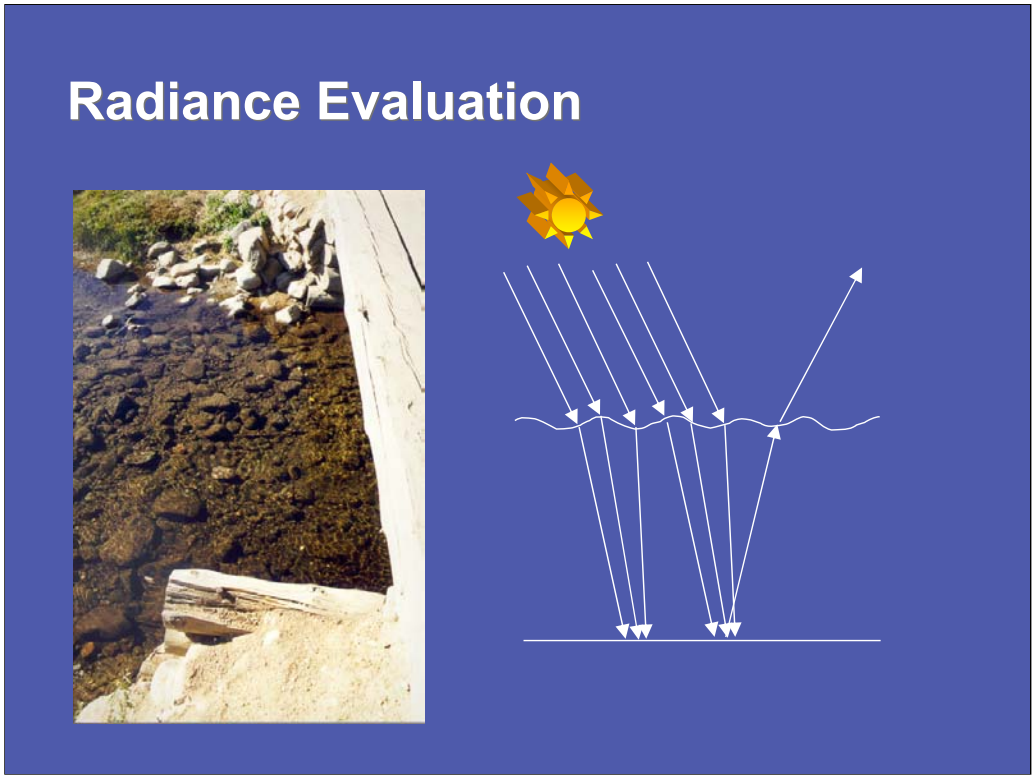
Radiance Evaluation



Reconstructing all possible paths between the light sources and the point for which one wants to compute a radiance value is the core business of all global illumination algorithms.

This photograph was taken on a sunny day in New Mexico. It is shown here just to illustrate some of the unexpected light paths one might have to reconstruct when computing global illumination solutions.

The circular figure on the left wall is the reflection of the lid on the trash can. The corresponding light paths (traced from the sun), hit the lid, then hit the wall, and finally end up in our eye. For a virtual scene, these same light paths need to be followed to reconstruct the reflection.



This photograph shows a similar effect.

We see shimmering waves on the bottom of the river (a similar effect is noticable in swimming pools). Light rays from the sun hit the transparent and wavy surface of the water, then are reflected on the bottom of the river, are refracted again by the water, the they hit our eye.

The complex pattern of light rays hitting the bottom, together with the changing nature of the surface of the water, causes these shimmering waves.

This effect is known as a caustic: light rays are reflected or refracted in different patterns and form images of the light source: the circular figure in the previous photograph, or the shimmering waves in this one.

(www.renderpark.be)

RE → paths

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{hemisphere} L(x \leftarrow \Psi) f_r(\ldots) \cos(\ldots) d\omega_\Psi$$

Paths of length 1:

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{hemisphere} L_e(x \leftarrow \Psi_x) f_r(\ldots) \cos(\ldots) d\omega_{\Psi_x}$$

Paths of length 2:

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{h\_x} \left(\int_{h\_y} L_e(y \leftarrow \Psi_y) f_r(\ldots) \cos(\ldots) d\omega_{\Psi_y} \right) f_r(\ldots) \cos(\ldots) d\omega_{\Psi_x}$$

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{h\_x} \int_{h\_y} L_e(y \leftarrow \Psi_y) f_r(\ldots) \cos(\ldots) f_r(\ldots) \cos(\ldots) d\omega_{\Psi_y} d\omega_{\Psi_x}$$

Paths of length 3:

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{h\_x} \int_{h\_y} \int_{h\_z} L_e(z \leftarrow \Psi_z) f_r(\ldots) \cos(\ldots) f_r(\ldots) \cos(\ldots) f_r(\ldots) \cos(\ldots) d\omega_{\Psi_z} d\omega_{\Psi_y} d\omega_{\Psi_x}$$

Path formulation

$$L(x \to \Theta) = \int_{paths} L_e(y \to \Psi)\, T((y,\Psi) \Rightarrow (x,\Theta))\, d\mu_{path}$$

$$T((y,\Psi) \Rightarrow (x,\Theta))$$

- Transfer function
  - All possible paths
  - … of any length
  - … any material reflections
  - … no light transport neglected

Path formulation

- Many different light paths contribute to single radiance value
  - many paths are unimportant
- Tools we need:
  - generate the light paths
  - sum all contributions of all light paths
  - clever techniques to select important paths

So, many different light paths, all originating at the light sources, will contribute to the value of the radiance at a single surface point.

Many of these light paths will be unimportant. Imagine a light switched on on the 1st floor of a building. You can imagine that some photons will travel all the way up to the 4th floor, but it is very unlikely that this will contribute significantly to the illumination on the 4th floor. However, we cannot exclude these light paths from consideration, since it might happen that the contribution is significant after all.

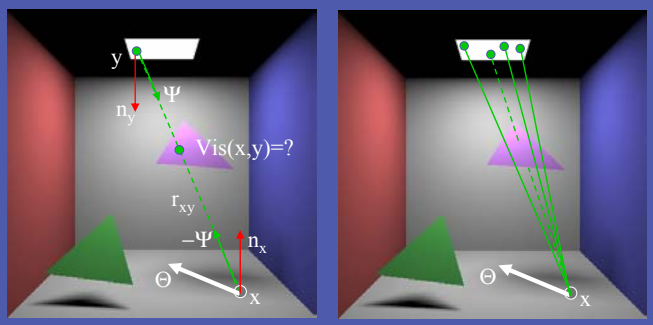
So, one of the mechanisms that a good global illumination algorithm needs is how to select the important paths from amongst many different possibilities, or at least how to try to put more computational effort into the ones that are likely to give the best contributions.

This is of course a chicken and egg problem. If we would know what the importance of each path was, we would have solved the global illumination problem. So the best we can do is to make clever guesses.

Direct Illumination

$$L(x \to \Theta) = \int_{A_{source}} f_r(x, -\Psi \leftrightarrow \Theta) \cdot L(y \to \Psi) \cdot G(x,y) \cdot dA_y$$

$$G(x,y) = \frac{\cos(n_x, \Theta)\cos(n_y, \Psi) Vis(x,y)}{r_{xy}^2}$$



area integration

One can do better by reformulating the rendering equation for direct illumination. Instead of integrating over a hemisphere, we will integrate over the surface area of the light source. This is valid, since we are only interested in the contribution due to the light source.

To transform the hemispherical coordinates to area coordinates over the hemisphere, we need to transform a differential solid angle to a differential surface. This introduces an extra cosine term and an inverse distance squared factor.

Additionally, the visibility factor, which was hidden in the hemispherical formulation since we ‘traced’ the ray to the closest intersection point, now needs to be mentioned explicitly.

Generating direct paths

- Parameters
  - How many paths (“shadow-rays”)?
    - total?
    - per light source? (~intensity, importance, …)
  - How to distribute paths within light source?
    - distance from point x
    - uniform

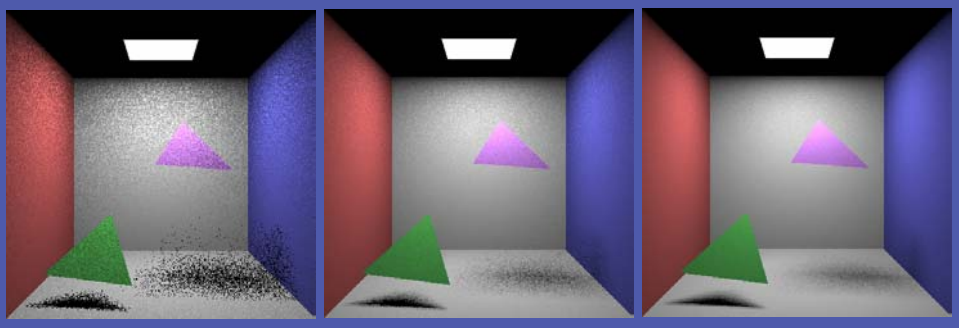
To compute the direct illumination using Monte Carlo integration, the following parameters can now be chosen:

- How many paths will be generated total for each radiance value to be computed? More paths result in a more accurate estimator, but the computational cost increases.

- How many of these paths will be send to each light source? It is intuitively obvious that one wants to send more paths to bright light sources, closer light sources, visible light sources.

- How to distribute the paths within each light source? When dealing with large light sources, points closer to the point to be shaded are more important than farther- away points.

Generating direct paths



1 path / source | 9 paths / source | 36 paths / source

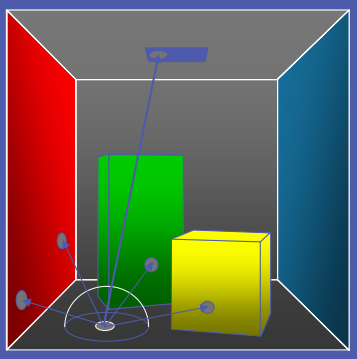
Here are a few examples of the results when generating a different number of paths per light source.

This simple scene has only one light source, and respectively 1, 9 and 36 paths are generated. The radiance values are computed more accurately in the latter case, and thus visible noise is less objectionable.

Although the first image is unbiased, its stochastic error is much higher compared to the last picture.

Alternative direct paths

– shoot paths at random over hemisphere; check if they hit light source



paths not used efficiently
noise in image

might work if light source occupies large portion on hemisphere

The algorithm in which the area of the light source is sampled is the most widely used way of computing direct illumination.

However, many more ways are possible, all based on a Monte Carlo evaluation of the rendering equation.

This slide shows an algorithm we have shown before: directions are sampled over the hemisphere, and they are traced to see whether they hit the light source and contribute to the radiance value we are interested in.

In this approach, many samples are wasted since their contribution is 0.

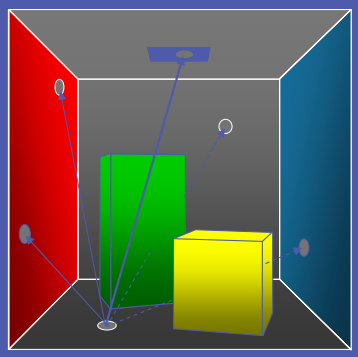
Alternative direct paths

1 paths / point 16 paths / point 256 paths / point

These images show the result of hemispherical sampling. As can be expected, many pixels are black when using only 1 sample, since we will only have a non-black pixel if the generated direction points to the light source.

Alternative direct paths

– pick random point on random surface; check if on light source and visible to target point



paths not used efficiently

noise in image

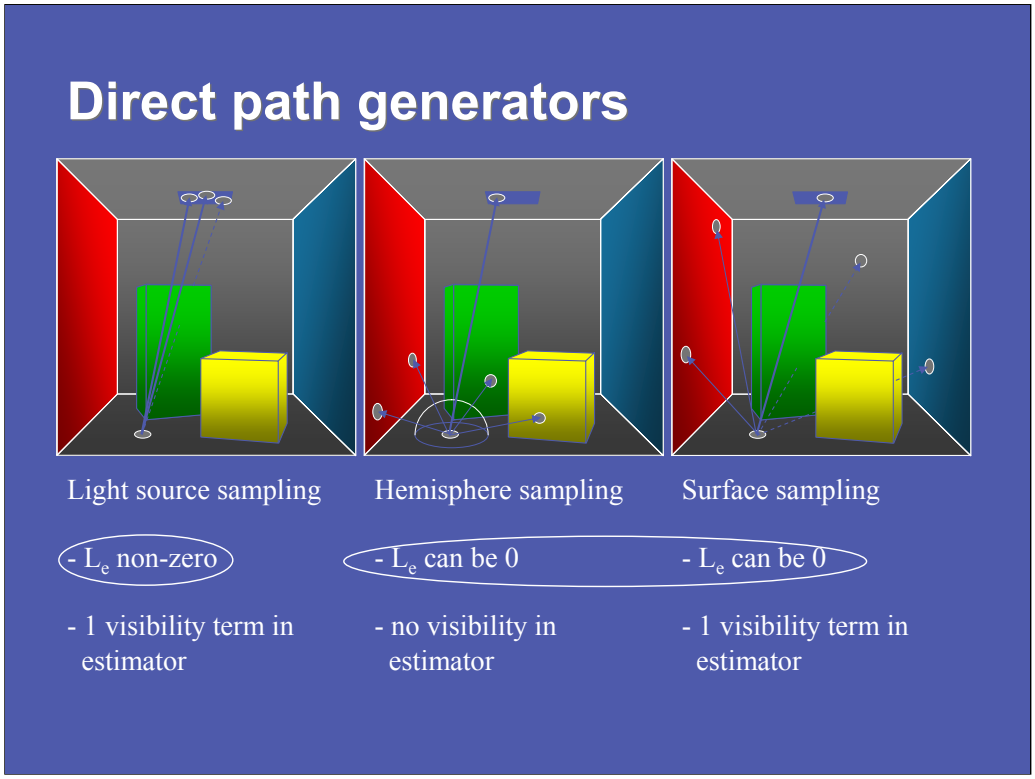
might work for large surface light sources

This is another algorithm for direct illumination:

We can write the rendering equation for as an integral over ALL surfaces in the scene, not just the light sources. Of course, the direct illumination contribution of most of these surfaces will be 0.

A Monte Carlo procedure will then sample a random surface point. For each of these surface points, we need to evaluate the self-emitted radiance (only different from 0 when a light source), the visibility between the sampled point and the target point, the geometry factor, and the BRDF.

Since both the self-emitted radiance and the visibility term might produce a 0 value in many cases, many of the samples will be wasted.



Here we see the 3 different approaches next to each other.

The noise resulting from each of these algorithms has different causes.

When sampling the area of the light source, most of the noise will come from failed visibility tests, and a little noise from a varying geometry factor.

When sampling the hemisphere, most noise comes from the self-emitted radiance being 0 on the visible point, but the visibility itself does not cause noise. However, each sample is more costly to evaluate, since the visibility is now folded into the ray tracing procedure.

When sampling all surfaces in the scene, noise comes failed visibility checks AND self-emitted radiance being 0. So this is obviously the worst case for computing direct illumination.

Although all these algorithms produce unbiased images when using enough samples, the efficiency of the algorithms is obviously different.

Direct paths

- Different path generators produce different estimators and different error characteristics
- Direct illumination general algorithm:

```
compute_radiance (point, direction)
      est_rad = 0;

      for (i=0; i<n; i++)
            p = generate_path;
            est_rad += energy_transfer(p) / probability(p);

      est_rad = est_rad / n;
      return(est_rad);
```

A general MC algorithm for computing direct illumination then generates a number of paths, evaluates for each path the necessary energy transfer along the path (radiance * BRDF * geometry), and computes the weighted average.

The differences in different algorithms lie in how efficient the paths are w.r.t. energy transfer.

Indirect Illumination

- Paths of length > 1

- Many different path generators possible
- Efficiency dependent on:
  - type of BRDFs along the path
  - Visibility function
  - ...

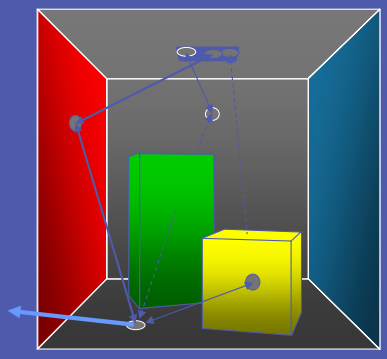
What about indirect illumination?

The principle remains exactly the same: we want to generate paths between a light source and a target point. The only difference is that the path will be of length greater than 1.

Again, the efficiency of the algorithm will depend on how clever the most useful paths can be generated.

Indirect paths - surface sampling

– Simple generator (path length = 2):
  - select point on light source
  - select random point on surfaces



per path:
2 visibility checks

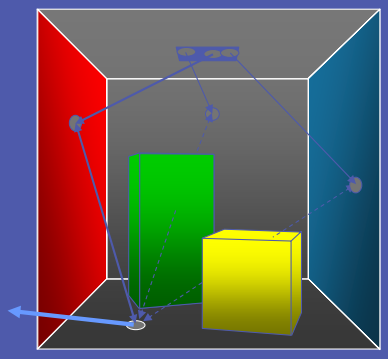
An added complexity is that we now have to deal with recursive evaluations. Although we show in these slides only the final paths between the light source and the target point, in an actual algorithm these paths will be generated recursively.

A simple algorithm involves samples all surface points in the scene. To generate paths of length 2, one can generate a random point on the surfaces, and a random point on a light source (direct illumination for the intermediate point). The necessary energy transfer is computed along the paths, and a weighted average using the correct pdf's is computed.

Indirect paths - source shooting

– “shoot” ray from light source, find hit location

– connect hit point to receiver



per path:

1 ray intersection

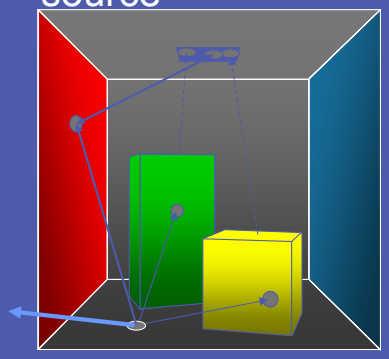
1 visibility check

This algorithm might generate the intermediate point in a slightly different way: a random direction is sampled over the hemisphere around a random point on the light source, this ray is traced in the environment, and the closest intersection point found.

Then this visible point is connected to the target point.

Indirect paths - receiver shooting

- "shoot" ray from receiver point, find hit location
- connect hit point to random point on light source



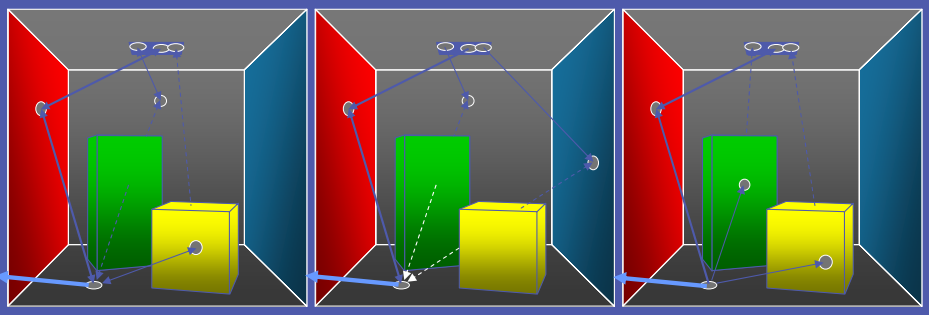
per path:
1 ray intersection
1 visibility check

Another algorithm might generate the intermediate point in a slightly different way: a random direction is sampled over the hemisphere around the target point, this ray is traced in the environment, and the closest intersection point found.

Then this visible point is connected to a random surface point generated on the light source.

This is the usual way of generating indirect paths in stochastic ray tracing.

Indirect paths



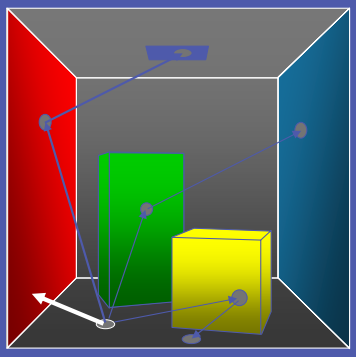
Surface sampling	Source shooting	Receiver shooting
- 2 visibility terms; can be 0	- 1 visibility term - 1 ray intersection	- 1 visibility term - 1 ray intersection

Here are all the different approaches compared.

All three of these algorithms will produce an unbiased image when generating enough samples, but the efficiency will be very different.

More variants ...

- "shoot" ray from receiver point, find hit location
- "shoot" ray from hit point, check if on light source



per path:
2 ray intersections
L_e might be zero

Even more variants can be thought of, as shown on this slide.

This is just to illustrate the general principle, that any path generator will do, as long as the correct energy transfer and correct probabilities for all the paths are computed.

Indirect paths

- Same principles apply to paths of length > 2
  - generate multiple surface points
  - generate multiple bounces from light sources and connect to receiver
  - generate multiple bounces from receiver and connect to light sources
  - …
- Estimator and noise characteristics change with path generator

For paths of length greater than 2, one can also come up with a lot of different path generators.

Usually these are implemented recursively.

Indirect paths

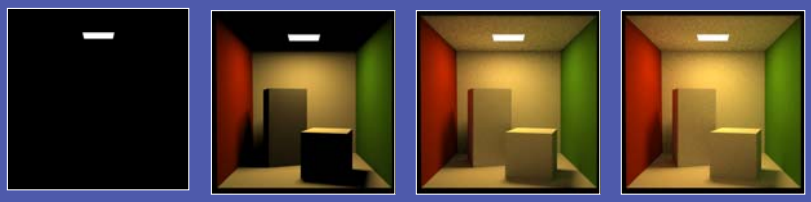
– General algorithm:

```
compute_radiance (point, direction)
      est_rad = 0;

      for (i=0; i<n; i++)
            p = generate_indirect_path;
            est_rad += energy_transfer(p) / probability(p);

      est_rad = est_rad / n;
      return(est_rad);
```

Indirect paths
How to end recursion?



- Contributions of further light bounces become less significant
- If we just ignore them, estimators will be incorrect!

An important issue when writing a recursive path generator is how to stop the recursion.

Our goal is still to produce unbiased images, that is, images which will be correct if enough samples are being generated.

As such, we cannot ignore deeper recursions, although we would like to spend less time on them, since the light transport along these longer paths is will probably be less significant.

Russian Roulette

Integral

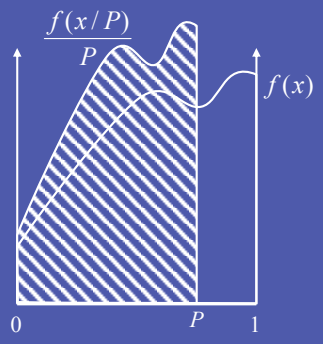
$$I = \int_0^1 f(x)dx = \int_0^P f(\frac{x}{P})Pdx$$

Estimator

$$\langle I_{roulette} \rangle = \begin{cases} f(\frac{x_i}{P}) / P & \text{if } x_i \le P, \\ 0 & \text{if } x_i > P. \end{cases}$$

Variance

$$\sigma_{roulette} > \sigma$$



Russian Roulette is a technique that can be used to stop the recursion.

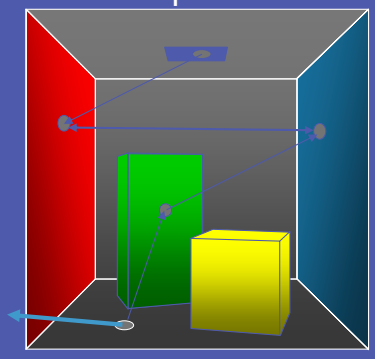
Mathematically, it means that we will cosnider part of integration domain to have a function value of 0. If a sample is generated in this part of the domain, it is ‘absorbed’. Of course, this means that the samples which are not absorbed will need to get a greater weight, since they have to compensate for the fact that we still want an unbiased estimator for the original integral.

Russian Roulette

- In practice: pick some 'absorption probability' α
  - probability $1-\alpha$ that ray will bounce
  - estimated radiance becomes $L/(1-\alpha)$
- E.g. $\alpha = 0.9$
  - only 1 chance in 10 that ray is reflected
  - estimated radiance of that ray is multiplied by 10
- Intuition
  - instead of shooting 10 rays, we shoot only 1, but count the contribution of this one 10 times

Complex path generators

- Bidirectional ray tracing
  - shoot a path from light source
  - shoot a path from receiver
  - connect end points



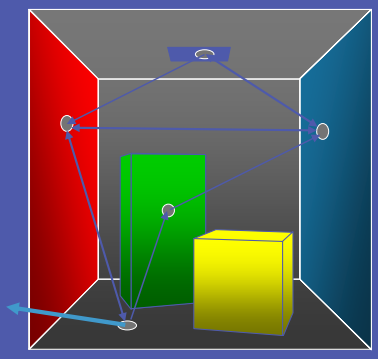
More complex path generators are also possible.

Bidirectional ray tracing is an algorithm that generates paths with variable length, both from the light source and the eye, and connects the end points.

Again, this is path generator, and results in an unbiased images if all relevant pdf's are taken into account.

Complex path generators

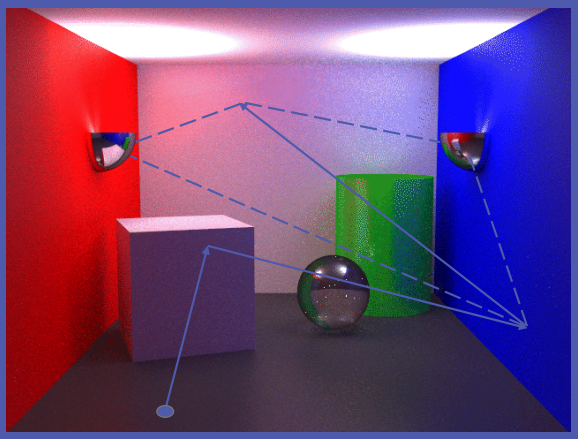
Combine all different paths and weight them correctly



Bidirectional ray tracing

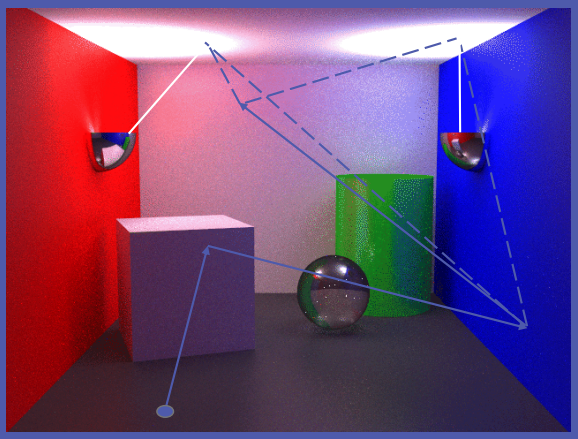
- Parameters
  - eye path length = 0: shooting from source
  - light path length = 0: shooting from receiver
- When useful?
  - Light sources difficult to reach
  - Specific brdf evaluations (e.g., caustics)

Bidirectional ray tracing



(E. Lafortune, 1996)

Bidirectional ray tracing



(E. Lafortune, 1996)

Classic ray tracing?

- Classic ray tracing:
  - shoot shadow-rays (direct illumination)
  - shoot perfect specular rays only for indirect
- Ignores many paths
  - does not solve the rendering equation

How does classic ray tracing compare to the physically correct path genertors described so far?

Classic ray tracing only generates a subset of all possible paths: shadow rays, and the perfect specular and refractive paths. As such, classic ray tracing ignores many of the other paths along which energy is transported from the light sources to the receiving surfaces.

Even more advanced …

- Store paths and re-use
  - Photon-mapping (Jensen)
  - Radiance cache (Walter)
  - Irradiance gradients (Ward)
- Mutate existing paths
  - Metropolis (Veach)

General global illumination algorithm

- Design path generators
- Path generators determine efficiency of global illumination algorithm
- Future:
  - Different path generators for different areas of the image
  - Adaptively
  - Re-use of paths as much as possible

The Rendering Equation and Path Tracing

This chapter gives various formulations of the rendering equation, and outlines several strategies for computing radiance values in a scene.

8.1 Formulations of the rendering equation

The global illumination problem is in essence a transport problem. Energy is emitted by light sources and transported through the scene by means of reflections (and refractions) at surfaces. One is interested in the energy equilibrium of the illumination in the environment.

The transport equation that describes global illumination transport is called the rendering equation. It is the integral equation formulation of the definition of the BRDF, and adds the self-emittance of surface points at light sources as an initialization function. The self-emitted energy of light sources is necessary to provide the environment with some starting energy. The radiance leaving some point x, in direction Θ, can be expressed as an integral over all hemispherical directions incident on the point x (figure 8.1):

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{\Omega_x} f_r(x, \Psi \leftrightarrow \Theta) L(x \leftarrow \Psi) cos(N_x, \Psi) d\omega_\Psi$$

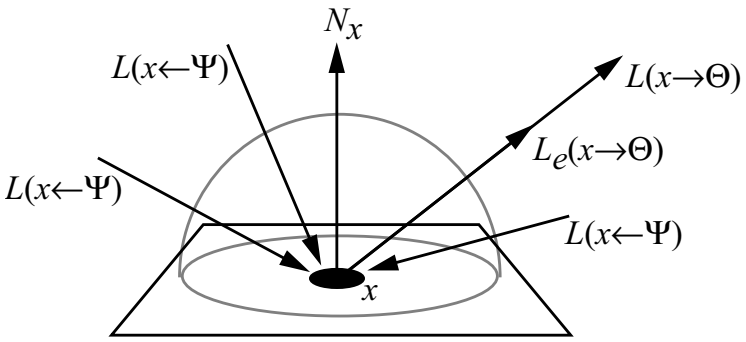


Figure 8.1: Rendering equation

One can transform the rendering equation from an integral over the hemisphere to an integral over all surfaces in the scene. Also, radiance remains unchanged along straight paths, so exitant radiance can be transformed to incident radiance and vice-versa, thus obtaining new versions of the rendering equation. By combining both options with a hemispheric or surface integration, four different formulations of the rendering equation are obtained. All these formulations are mathematically equivalent.

Exitant radiance, integration over the hemisphere

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{\Omega_x} f_r(x, \Psi \leftrightarrow \Theta) L(y \rightarrow -\Psi) \cos(N_x, \Psi) d\omega_\Psi$$

with

$$y = r(x, \Theta)$$

When designing an algorithm based on this formulation, integration over the hemisphere is needed, and as part of the function evaluation for each point in the integration domain, a ray has to be cast and the nearest intersection point located.

Exitant radiance, integration over surfaces

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_A f_r(x, \Psi \leftrightarrow \Theta) L(y \rightarrow \overrightarrow{yx}) V(x, y) G(x, y) dA_y$$

with

$$G(x, y) = \frac{cos(N_x, \Psi) cos(N_y, \Psi)}{r_{xy}^2}$$

Algorithms based on this formulation need to evaluate the visibility $V(x, y)$ between two points x and y, which is a different operation than casting a ray from x in a direction Θ.

Incident radiance, integration over the hemisphere

$$L(x \leftarrow \Theta) = L_e(x \leftarrow \Theta) + \int_{\Omega_y} f_r(y, \Psi \leftrightarrow -\Theta) L(y \leftarrow \Psi) \cos(N_y, \Psi) d\omega_\Psi$$

with

$$y = r(x, \Theta)$$

Incident radiance, integration over surfaces

$$L(x \leftarrow \Theta) = L_e(x \leftarrow \Theta) + \int_A f_r(y, \Psi \leftrightarrow \overrightarrow{yz}) L(y \leftarrow \overrightarrow{yz}) V(y, z) G(y, z) dA_z$$

with

$$y = r(x, \Theta)$$

8.2 Importance function

In order to compute the average radiance value over the area of a pixel, one needs to know the radiant flux over that pixel (and associated solid angle incident w.r.t. the aperture of the camera). Radiant flux is expressed by integrating the radiance distribution over all possible surface points and directions. Let $S = A_p \times \Omega_p$ denote all surface points A_p and directions Ω_p visible through the pixel. The flux $\Phi(S)$ is written as:

$$\Phi(S) = \int_{A_p} \int_{\Omega_p} L(x \rightarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x$$

When designing algorithms, it is often useful to express the flux as an integral over all possible points and directions in the scene. This can be achieved by introducing the initial importance function $W_e(x \leftarrow \Theta)$:

$$\Phi(S) = \int_A \int_\Omega L(x \rightarrow \Theta) W_e(x \leftarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x$$

$W_e(x \leftarrow \Theta)$ is appropriately defined by:

$$W_e(x \leftarrow \Theta) = \begin{cases} 1 & \text{if } (x,\Theta) \in S \\ 0 & \text{if } (x,\Theta) \notin S \end{cases}$$

The average radiance value is then given by:

$$L_{average} = \frac{\int_A \int_\Omega L(x \rightarrow \Theta) W_e(x \leftarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x}{\int_A \int_\Omega W_e(x \leftarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x}$$

We now want to develop the notion of importance further, by considering the possible influence of some energy value at each pair (x, Θ) on the value $\Phi(S)$. Or: if a single radiance value $L(x \rightarrow \Theta)$ is placed at (x, Θ), and if there are no other sources of illumination present, how large would the resulting value of $\Phi(S)$ be? This influence value attributed to $L(x \rightarrow \Theta)$ is called the importance of (x, Θ) w.r.t. S, is written as $W(x \leftarrow \Theta)$, and depends only on the geometry and reflective properties of the objects in the scene.

The equation expressing $W(x \leftarrow \Theta)$ can be derived by taking into account two mechanisms in which $L(x \rightarrow \Theta)$ can contribute to $\Phi(S)$:

Self-contribution If $(x, \Theta) \in S$, then $L(x \rightarrow \Theta)$ fully contributes to $\Phi(S)$. This is called the self-importance of the set S, and corresponds to the above definition of $W_e(x \leftarrow \Theta)$.

Indirect contributions It is possible that some part of $L(x \rightarrow \Theta)$ contributes to $\Phi(S)$ through one or more reflections at several surfaces. The radiance $L(x \rightarrow \Theta)$ travels along a straight path and reaches a surface point $r(x, \Theta)$. Energy is reflected at this surface point according to the BRDF. Thus, there is a hemisphere of directions at $r(x, \Theta)$, each emitting a differential radiance value as a result of the reflection of the radiance $L(r(x, \Theta) \leftarrow -\Theta)$. By integrating the importance values for all these new directions, we have a new term for $W(x \leftarrow \Theta)$.

Both terms combined produces the following equation:

$$W(x \leftarrow \Theta) = W_e(x \leftarrow \Theta) + \int_{\Omega_z} f_r(z, \Psi \leftrightarrow -\Theta) W(z \leftarrow \Psi) \cos(N_r(x, \Theta), \Psi) d\omega_\Psi$$

with

$$z = r(x, \Theta)$$

Mathematically, this equation is identical to the transport equation of incident radiance, and thus, the notion *incidence* can be attributed to importance. The source function $W_e = 1$ if x is visible through the pixel and Θ is a direction pointing through the pixel to the aperture of the virtual camera.

To enhance the analogy with radiance as a transport quantity, exitant importance can be defined as:

$$W(x \rightarrow \Theta) = W((r, \Theta) \leftarrow -\Theta)$$

and also:

$$W(x \rightarrow \Theta) = W_e(x \rightarrow \Theta) + \int_{\Omega_x} f_r(x, \Psi \leftrightarrow \Theta) W(x \leftarrow \Psi) cos(N_x, \Psi) d\omega_\Psi$$

An expression for the flux of through every pixel, based on the importance function, can now be written. Only the importance of the light sources needs to be considered when computing the flux:

$$\Phi(S) = \int_A \int_{\Omega_x} L_e(x \rightarrow \Theta) W(x \leftarrow \Theta) cos(N_x, \Theta) d\omega_\Theta dA_x$$

It is also possible to write $\Phi(S)$ in the following form:

$$\Phi(S) = \int_A \int_{\Omega_x} L_e(x \leftarrow \Theta) W(x \rightarrow \Theta) cos(N_x, \Theta) d\omega_\Theta dA_x$$

and also:

$$\begin{aligned} \Phi(S) &= \int_A \int_{\Omega_x} L(x \rightarrow \Theta) W_e(x \leftarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x \\ \Phi(S) &= \int_A \int_{\Omega_x} L(x \leftarrow \Theta) W_e(x \rightarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x \end{aligned}$$

There are two approaches to solve the global illumination problem: The first approach starts from the pixel, and the radiance values are computed by solving one of the transport equations describing radiance. A second approach computes the flux starting from the light sources, and computes for each light source the corresponding importance value. If one looks at various algorithms in some more detail:

- Stochastic ray tracing propagates importance, the surface area visible through each pixel being the source of importance. In a typical implementation, the importance is never explicitly computes, but is implicitly done by tracing rays through the scene and picking up illumination values from the light sources.

- Light tracing is the dual algorithm of ray tracing. It propagates radiance from the light sources, and computes the flux values at the surfaces visible through each pixel.

- Bidirectional ray tracing propagates both transport quantities at the same time, and in an advanced form, computes a weighted average of all possible inner products at all possible interactions.

8.3 Path formulation

The above description of global illumination transport algorithms is based on the notion of radiance and importance. One can also express global transport by considering path-space, and computing a transport measure over each individual path. Path-space encompasses all possible paths of any length. Integrating a transport measure in path-space then involves generating the correct paths (e.g. random paths can be generated using an appropriate Monte Carlo sampling procedure), and evaluating the throughput of energy over each generated path. This view was developed by Spanier and Gelbard and introduced into rendering by Veach.

$$\Phi(S) = \int_{\Omega^*} f(\overline{x}) d\mu(\overline{x})$$

in which Ω^* is the path-space, $\overline{x}$ is a path of any length and $d\mu(\overline{x})$ is a measure in path space . $f(\overline{x})$ describes the throughput of energy and is a succession of $G(x, y)$, $V(x, y)$ and BRDF evaluations, together with a L_e and W_e evaluation at the beginning and end of the path.

An advantage of the path formulation is that paths are now considered to be the sample points for any integration procedure. Algorithms such as Metropolis light transport or bidirectional ray tracing are often better described using the path formulation.

8.4 Simple stochastic ray tracing

In any pixel-driven rendering algorithm we need to use the rendering equation to evaluate the appropriate radiance values. The most simple algorithm to compute this radiance value is to apply a basic and straightforward MC integration scheme to the standard form of the rendering equation:

$$\begin{aligned} L(x \rightarrow \Theta) &= L_e(x \rightarrow \Theta) + L_r(x \rightarrow \Theta) \\ &= L_e(x \rightarrow \Theta) + \int_{\Omega_x} L(x \leftarrow \Psi) f_r(x, \Theta \leftrightarrow \Psi) \cos(\Psi, N_x) d\omega_\Psi \end{aligned}$$

The integral is evaluated using MC integration, by generating N random directions Ψ_i over the hemisphere Ω_x, according to some pdf $p(\Psi)$. The estimator for $L_r(x \rightarrow \Theta)$ is given by:

$$\langle L_r(x \rightarrow \Theta) \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{L(x \leftarrow \Psi_i) f_r(x, \Theta \leftrightarrow \Psi_i) \cos(\Psi_i, N_x)}{p(\Psi_i)}$$

$L(x \leftarrow \Psi_i)$, the incident radiance at x, is unknown. It is now necessary to trace the ray leaving x in direction Ψ_i through the scene to find the closest intersection point $r(x, \Psi)$. Here, another radiance evaluation is needed. The result is a recursive procedure to evaluate $L(x \leftarrow \Psi_i)$, and as a consequence, a path, or a tree of paths if $N > 1$, is generated in the scene.

These radiance evaluations will only yield a non-zero value, if the path hits a surface for which L_e has a value different from 0. In other words, in order to compute a contribution to the illumination of a pixel, the recursive path needs to reach at least one of the light sources in the scene. If the light sources are small, the resulting image will therefore mostly be black. This is expected, because the algorithm generates paths, starting at a point visible through a pixel, and slowly working towards the light sources in a very uncoordinated manner.

8.5 Russian Roulette

The recursive path generator described above needs a stopping condition to prevent the paths being of infinite length. We want to cut off the generation of paths, but at the same time, we have to be very careful about not introducing any bias into the

image generations process. Russian Roulette addresses the problem of keeping the lengths of the paths manageable, but at the same time leaves room for exploring all possible paths of any length. Thus, an unbiased image can still be produced.

The idea of Russian Roulette can best be explained by a simple example: suppose one wants to compute a value V. The computation of V might be computationally very expensive, so we introduce a random variable r, which is uniformly distributed over the interval $[0, 1]$. If r is larger than some threshold value $\alpha \in [0, 1]$, we proceed with computing V. However, if $r \leq \alpha$, we do not compute V, and assume $V = 0$. Thus, we have a random experiment, with an expected value of $(1 - \alpha)V$. By dividing this expected value by $(1 - \alpha)$, an unbiased estimator for V is maintained.

If V requires recursive evaluations, one can use this mechanism to stop the recursion. α is called the absorption probability. If α is small, the recursion will continue many times, and the final computed value will be more accurate. If α is large, the recursion will stop sooner, and the estimator will have a higher variance. In the context of our path tracing algorithm, this means that either accurate paths of a long length are generated, or very short paths which provide a less accurate estimate.

In principle any value for α can be picked, thus controlling the recursive depth and execution time of the algorithm. $1 - \alpha$ is often set to be equal to the hemispherical reflectance of the material of the surface. Thus, dark surfaces will absorb the path more easily, while lighter surfaces have a higher chance of reflecting the path.

8.6 Indirect Illumination

In most path tracing algorithms, direct illumination is explicitly computed separately from all other forms of illumination (see previous chapter on direct illumination). This section outlines some strategies for computing the indirect illumination in a scene. Computing the indirect illumination is usually a harder problem, since one does not know where most important contributions are located. Indirect illumination consists of the light reaching a target point x after at least one reflection at an intermediate surface between the light sources and x.

8.6.1 Hemisphere sampling

The rendering equation can be split in a direct and indirect illumination term. The indirect illumination (i.e. not including any direct contributions from light sources to the point x) contribution to $L(x \rightarrow \Theta)$ is written as:

$$L_{indirect}(x \rightarrow \Theta) = \int_{\Omega_x} L_r(r(x, \Psi) \rightarrow -\Psi) f_r(x, \Theta \leftrightarrow \Psi) \cos(\Psi, N_x) d\omega_\Psi$$

The integrand contains the reflected terms L_r from other points in the scene, which are themselves composed of a direct and indirect illumination part. In a closed environment, $L_r(r(x, \Psi) \rightarrow -\Psi)$ usually has a non-zero value for all (x, Ψ) pairs. As a consequence, the entire hemisphere around x needs to be considered as the integration domain.

The most general MC procedure to evaluate indirect illumination, is to use any hemispherical pdf $p(\Psi)$, and generating N random directions Ψ_i. This produces the following estimator:

$$\langle L_{indirect}(x \rightarrow \Theta) \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{L_r(r(x, \Psi_i) \rightarrow -\Psi_i) f_r(x, \Theta \leftrightarrow \Psi_i) \cos(\Psi_i, N_x)}{p(\Psi_i)}$$

In order to evaluate this estimator, for each generated direction Ψ_i, the BRDF and the cosine term are to be evaluated, a ray from x in the direction of Ψ_i needs to be traced, and the reflected radiance $L_r(r(x, \Psi_i) \rightarrow -\Psi_i)$ at the closest intersection point $r(x, \Psi_i)$ has to be evaluated. This last evaluation shows the recursive nature of indirect illumination, since this reflected radiance at $r(x, \Psi_i)$ can be split again in a direct and indirect contribution.

The simplest choice for $p(\Psi)$ is $p(\Psi) = 1/2\pi$, such that directions are sampled proportional to solid angle. Noise in the resulting picture will be caused by variations in the BRDF and cosine evaluations, and variations in the reflected radiance L_r at the distant points.

The recursive evaluation can again be stopped using Russian Roulette, in the same way as was done for simple stochastic ray tracing. Generally, the local hemispherical reflectance is used as an appropriate absorption probability. This choice can be explained intuitively: One only wants to spend work (i.e. tracing rays and evaluating $L_{indirect}(x)$) proportional to the amount of energy present in different parts of the scene.

8.6.2 Importance sampling

Uniform sampling over the hemisphere does not use any knowledge about the integrand in the indirect illumination integral. However, this is necessary to reduce noise in the final image, and thus, some form of importance sampling is needed. Hemispherical pdf's proportional (or approximately proportional) to any of the following factors can be constructed:

Cosine sampling

Sampling directions proportional to the cosine lobe around the normal N_x prevents directions to be sampled near the horizon of the hemisphere where $\cos(\Psi, N_x)$ yields a very low value, and thus possibly insignificant contributions to the computed radiance value.

BRDF sampling

BRDF sampling is a good noise-reducing technique when a glossy or highly specular BRDFs is present. It diminishes the probability that directions are sampled where the BRDF has a low value or zero value. Only for a few selected BRDF models, however, is it possible to sample exactly proportional to the BRDF. Even better would be trying to sample proportional to the product of the BRDF and the cosine term. Analytically, this is even more difficult to do, except in a few rare cases where the BRDF model has been chosen carefully.

Incident radiance field sampling

A last technique that can be used to reduce variance when computing the indirect illumination is to sample a direction Ψ according to the incident radiance values $L_r(x \leftarrow \Psi)$. Since this incident radiance is generally unknown, an adaptive technique needs to be used, where an approximation of $L_r(x \leftarrow \Psi)$ is constructed during the execution of the rendering algorithm.

8.6.3 Overview

It is now possible to build a full global illumination renderer using stochastic path tracing. The efficiency, accuracy and overall performance of the complete algorithm will be determined by the choice of all of the following parameters. As is usual in MC evaluations, the more samples or rays are generated, the less noisy the final image will be.

Number of viewing rays per pixel The amount of viewing rays through each pixel is responsible for effects such as aliasing at visible boundaries of objects or shadows.

Direct Illumination:

- The total number of shadow rays generated at each surface point x;
- The selection of a single light source for each shadow ray;
- The distribution of the shadow ray over the area of the selected light source.

Indirect Illumination (hemisphere sampling):

- Number of indirect illumination rays;
- Exact distribution of these rays over the hemisphere (uniform, cosine, ...);
- Absorption probabilities for Russian Roulette.

The better one makes use of importance sampling, the better the final image and the less noise there will be. An interesting question is, given a maximum amount of rays one can use per pixel, how should these rays best be distributed to reach the highest possible accuracy for the full global illumination solution? This is still an open problem. There are generally accepted 'default' choices, but there are no hard and fast choices. It generally is accepted that branching out equally at all levels of the tree is less efficient. For indirect illumination, a branching factor of 1 is often used after the first level. Many implementations even limit the indirect rays to one per surface point, and compensate by generating more viewing rays.

Monte Carlo Sampling Techniques for Image Synthesis

James Arvo
Department of Information and Computer Science, and
Department of Electrical Engineering and Computer Science
University of California, Irvine

Abstract

In this portion of the notes we will look at some practical methods for constructing sampling algorithms that are both unbiased *and* efficient*; that is, algorithms that allow us to converge to the* correct *answer using* fewer *samples. We shall first look at the problem of generating uniform parametrizations for arbitrary regions of the plane or portions of surfaces. Such parametrizations allow us to employ a very common form of* variance reduction *known as* stratification*, which generally increases the efficiency of any sampling method used in image synthesis. We then look at the problem of combining several sampling techniques into a single unbiased strategy that retains the advantages of the individual strategies.*

1 Introduction

Monte Carlo techniques arise in image synthesis primarily as a means to solve integration problems. Integration over domains of two or higher dimensions is ubiquitous in image synthesis; indirect global illumination is expressed as an integral over all paths of light, which entails numerous direct illumination problems such as the computation of form factors, visibility, reflected radiance, subsurface scattering, and irradiance due to complex or partially occluded luminaires, all of which involve integration.

The Monte Carlo method stems from a very natural and immediate connection between integration and *expectation*. Every integral, in both deterministic and probabilistic contexts, can be viewed as the expected value (mean) of a random variable; by averaging over many samples of the random

variable, we may thereby approximate the integral. However, there are infinitely many random variables that can be associated with any given integral; of course, some are better than others.

One attribute that makes some random variables better than others for the purpose of integration is the ease with which they can be sampled. In general, we tend to construct Monte Carlo methods using only those random variables with convenient and efficient sampling procedures. But there is also a competing attribute. One of the maxims of Monte Carlo integration is that the probability density function of the random variable should mimic the integrand as closely as possible. The closer the match, the smaller the variance of the random variable, and the more reliable (and efficient) the estimator. In the limit, when the samples are generated with a density that is exactly proportional to the (positive) integrand, the variance of the estimator is identically zero [9]. That is, a single sample delivers the exact answer with probability one.

Perhaps the most common form of integral arising in image synthesis is that expressing either irradiance or reflected radiance at a surface. In both cases, we must evaluate (or approximate) an integral over solid angle, which is of the form

$$\int_S f(\vec{\omega})(\vec{\omega} \cdot \vec{n})\, d\omega, \tag{1}$$

where S is subset of the unit sphere, $\vec{\omega}$ is a unit direction vector, and $\vec{n}$ is the surface normal vector, or over surface area, which is of the form

$$\int_A f(x) \frac{(x \cdot \vec{n})(x \cdot \vec{n}')}{||x||^2}\, dx, \tag{2}$$

where A is a surface and x is a point in $\mathbb{R}^3$. Technically, these integrals differ only by a change of variable that results from the pullback of surface differentials to solid angle differentials [1]. The function f may represent the radiance or emissive power of a luminaire (in the case of irradiance) or it may include a BRDF (in the case of reflected radiance). In all cases, visibility may be included in the function f, which can make the integrand arbitrarily discontinuous, thereby drastically reducing the effectiveness of standard numerical quadrature methods for computing the integral.

To apply Monte Carlo integration most effectively in image synthesis, we seek sampling algorithms that match the geometries and known light distributions found in a simulated scene to the extent possible. Consequently, a wide assortment of sampling algorithms have been developed for sampling

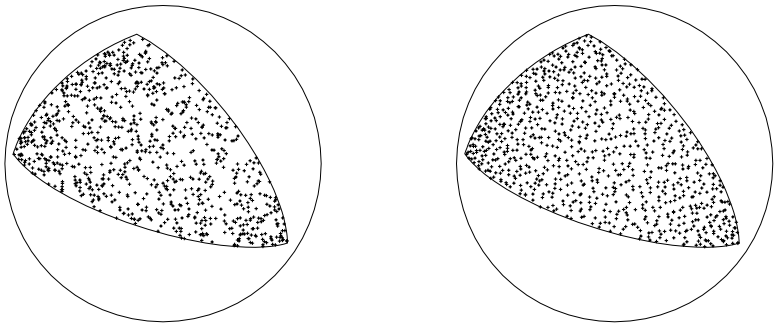


Figure 1: *A spherical triangle with uniform samples (left) and stratified samples (right). Both sets of samples were generated using a uniform parametrization for spherical triangles, which we derive below.*

both the surfaces of and the solid angles subtended by various scene geometries [10] so that both forms of the integral above can be accommodated. In this chapter we will see how to construct random variables for specific geometries: that is, random variables whose range coincides with some bounded region of the plane or some bounded surface in $\mathbb{R}^3$, and whose probability density function is constant. For instance, we will see how to generate uniformly distributed samples over both planar and spherical triangles, and projected spherical polygons.

All of the sampling algorithms that we construct are based on mappings from the unit square, $[0,1] \times [0,1]$, to the regions or surfaces in question that preserve uniform sampling. That is, uniformly distributed samples in the unit square are mapped to uniformly distributed samples in the range. Such mappings also preserve *stratification*, also known as *jitter sampling* [5], which means that uniform partitionings of the unit square map to uniform partitionings of the range. The ability to apply stratified sampling over various domains is a great advantage, as it is often a very effective variance reduction technique. Figure 1 shows the result of applying such a mapping to a spherical triangle, both with and without stratified (jittered) sampling. Figure 2 shows the result of applying such a mapping to a projected spherical polygon, so that the samples on the original spherical polygon are "cosine-distributed" rather than uniformly distributed.

All of the resulting algorithms depend upon a source of uniformly distributed random numbers in the interval $[0,1]$, which we shall assume is available from some unspecified source: perhaps "drand48," or some other pseudo-random number generator.

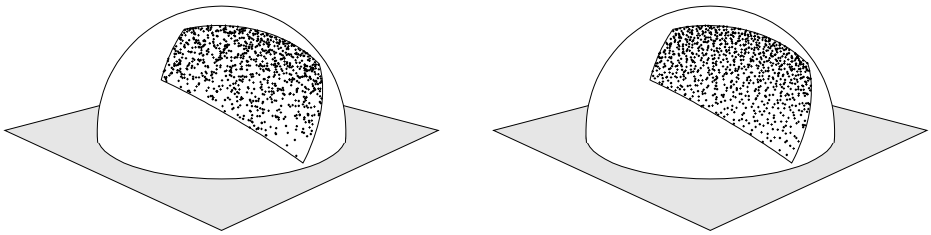


Figure 2: *A projected spherical polygon with uniform samples (left) and stratified samples (right). Projection onto the plane results in more samples near the north pole of the sphere than near the equator. Both sets of samples were generated using a uniform parametrization for spherical polygons, which we derive below.*

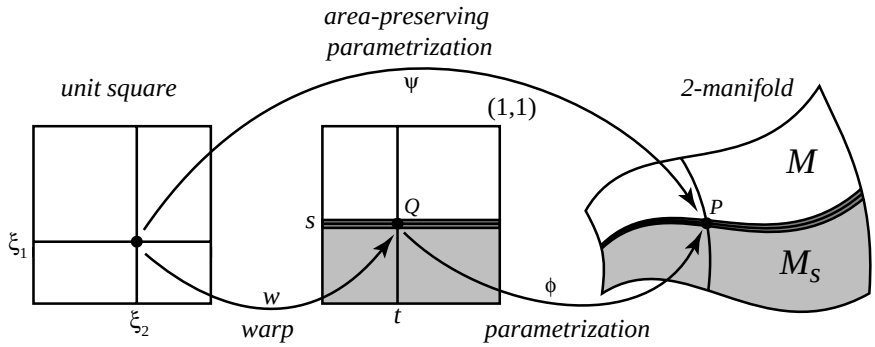


Figure 3: *An arbitrary parametrization ϕ for a 2-manifold $\mathcal{M}$ can be converted into a uniform parametrization, which is useful for uniform and stratified sampling, by composing it with a warping function. The warping function can be derived directly from ϕ by following a precise procedure.*

2 Sampling Algorithms for 2-Manifolds

Although there is a vast and mature literature an Monte Carlo methods, with many texts describing how to derive sampling algorithms for various geometries and density functions (see, for example, Kalos and Whitlock [6], Spanier and Gelbard [11], or Rubinstein [9]), these treatments do not provide step-by-step instructions for deriving the types of algorithms that we frequently require in computer graphics. In this section we present a detailed "recipe" for how to convert an arbitrary parametrization $\phi : [0,1]^2 \to \mathcal{M}$, from the unit square to a 2-manifold, into an *area-preserving*[1], or *uniform*,

[1]Unfortunately, there is no univerally accepted terminology for this type of mapping. The term "area-preserving" is technically a misnomer, as the mapping could uniformly magnify or shrink areas. We therefore opt for the term "uniform" here.

parametrization, $\psi : [0,1]^2 \rightarrow \mathcal{M}$. That is, a mapping ψ from $[0,1] \times [0,1]$ to the manifold $\mathcal{M}$ with the following property:

$$\textbf{area}(A) = \textbf{area}(B) \implies \textbf{area}(\psi[A]) = \textbf{area}(\psi[B]), \tag{3}$$

for all $A, B \in [0,1] \times [0,1]$. Or, equivalently,

$$\textbf{area}(A) \; = \; c \cdot \textbf{area}(\psi[A]) \tag{4}$$

for all $A \subseteq [0,1] \times [0,1]$, where $c > 0$ is a fixed constant.

Such mappings are used routinely in image synthesis to sample surfaces of luminaires and reflectors. Note that ψ may in fact shrink or magnify areas, but that all areas undergo exactly the same scaling; hence, it is area-preserving in the strictest sense only when $\textbf{area}(\mathcal{M}) = 1$. A parametrization with this property will allow us to generate uniformly distributed and/or stratified samples over $\mathcal{M}$ by generating samples with the desired properties on the unit square (which is trivial) and then mapping them onto $\mathcal{M}$. We shall henceforth consider uniform parametrizations to be synonymous with *sampling algorithms*.

Let $\mathcal{M}$ represent a shape that we wish to generate uniformly distributed samples on; in particular, $\mathcal{M}$ may be any 2-manifold with boundary in $\mathbb{R}^n$, where n is typically 2 or 3. The steps for deriving a sampling algorithm for $\mathcal{M}$ are summarized in Figure 4. These steps apply for all dimensions $n \geq 2$; that is, $\mathcal{M}$ may be a 2-manifold in any space.

Step 1 requires that we select a smooth bijection ϕ from $[0,1] \times [0,1]$ to the 2-manifold $\mathcal{M}$. Such a function is referred to as a *parametrization* and its inverse is called a *coordinate chart*, as it associates unique 2D coordinates with (almost) all points of $\mathcal{M}$. In reality, we only require ϕ to be a bijection *almost everywhere*; that is, on all but a set of measure zero, such as the boundaries of $\mathcal{M}$ or $[0,1] \times [0,1]$. In theory, any smooth bijection will suffice, although it may be impractical or impossible to perform some of the subsequent steps in Figure 4 symbolically (particularly step 4) for all but the simplest functions.

Step 2 defines a function $\sigma : [0,1]^2 \rightarrow \mathbb{R}$ that links the parametrization ϕ to the notion of surface area on $\mathcal{M}$. Specifically, when the relative density function[2], ρ, is constant, then for any region $A \subseteq [0,1]^2$, the function σ satisfies

$$\int_A \sigma \; = \; \textbf{area}(\phi[A]). \tag{11}$$

[2]While the emphasis in these notes in on generating *uniform* parametrizations, there are cases in which we desire something else. The method described here can handle those cases as well, via the *relative density function*. An example of this will be shown later.

1. Select a *parametrization* ϕ for the 2-manifold $\mathcal{M} \subset \mathbb{R}^n$. That is, select a smooth bijection (diffeomorphism) $\phi : [0,1]^2 \to \mathcal{M}$, and a *relative density* function $\rho : [0,1]^2 \to \mathbb{R}^+$.

2. Define the function $\sigma : [0,1]^2 \to \mathbb{R}$ by

$$\sigma \equiv \rho \cdot \sqrt{(\phi_s \cdot \phi_s)(\phi_t \cdot \phi_t) - (\phi_s \cdot \phi_t)^2} \tag{5}$$

   where $\phi_s = \left(\frac{\partial \phi_1}{\partial s}, \ldots, \frac{\partial \phi_n}{\partial s}\right)$ is the vector of partial derivatives.

3. Define two cumulative distribution functions on $[0,1]$ by

$$F(s) \equiv \frac{\int_0^1 \int_0^s \sigma(u,v)\, du\, dv}{\int_0^1 \int_0^1 \sigma(u,v)\, du\, dv} \tag{6}$$

$$G_s(t) \equiv \frac{\int_0^t \sigma(s,v)\, dv}{\int_0^1 \sigma(s,v)\, dv} \tag{7}$$

4. Invert the two cumulative distribution functions

$$f(z) \equiv F^{-1}(z) \tag{8}$$
$$g(z_1, z_2) \equiv G_{z_1}^{-1}(z_2) \tag{9}$$

5. Define the new parametrization $\psi : [0,1]^2 \to \mathcal{M}$ by

$$\psi(z_1, z_2) \equiv \phi(f(z_1), g(f(z_1), z_2)). \tag{10}$$

   Thus, the mapping $(z_1, z_2) \mapsto (f(z_1), g(f(z_1), z_2))$ defines the warping function that converts the original parametrization ϕ into the uniform parametrization ψ.

Figure 4: *Five steps for deriving a uniform parametrization ψ from $[0,1] \times [0,1]$ to any bounded 2-manifold $\mathcal{M} \subset \mathbb{R}^n$, beginning from an arbitrary parametrization ϕ for $\mathcal{M}$. The function ψ is suitable for stratified sampling of $\mathcal{M}$. Step 2 simplifies in the common two- and three- dimensional cases. Step 4 is often the only impediment to finding a closed-form expression for the uniform parametrization.*

That is, the integral of σ over any region A in the 2D parameter space is the surface area of the corresponding subset of $\mathcal{M}$ under the mapping ϕ. Equation (5) holds for all $n \geq 2$. For the typical cases of $n = 2$ and $n = 3$, however, the function σ can be expressed more simply. For example, if $\mathcal{M}$ is a subset of $\mathbb{R}^2$ then

$$\sigma \;\equiv\; \rho \cdot \det(D\phi)\,, \tag{12}$$

where $D\phi$ is the *total derivative* of ϕ, which in this case is the 2×2 Jacobian matrix of ϕ. On the other hand, if $\mathcal{M}$ is a subset of $\mathbb{R}^3$, then

$$\sigma \;\equiv\; \rho \cdot ||\, \phi_s \times \phi_t \,||\,, \tag{13}$$

which is a convenient abbreviation for equation (5) that holds only when $n = 3$, as the two partial derivatives of ϕ are vectors in $\mathbb{R}^3$ in this case. Non-uniform sampling results from allowing the relative density function ρ to be non-constant. In most of what follows, however, we will simply drop ρ, under the assumption that it is constant.

Step 3 can often be carried out without the aid of an explicit expression for σ. For example, the cumulative distributions can often be found by reasoning directly about the geometry imposed by the parametrization rather than applying formulas (5), (6) and (7), which can be tedious. Let $\mathcal{M}_s$ denote the family of sub-manifolds of $\mathcal{M}$ defined by the first coordinate of ϕ. That is,

$$\mathcal{M}_s \;=\; \phi\Big[\; [0, s] \times [0, 1] \;\Big]\,. \tag{14}$$

See Figure 3. It follows from the definition of F and equation (11) that

$$F(s) \;=\; \frac{\mathbf{area}(\mathcal{M}_s)}{\mathbf{area}(\mathcal{M})}, \tag{15}$$

which merely requires that we find an expression for the surface area of $\mathcal{M}_s$ as a function of s. Similarly, by equation (8) we have

$$s \;=\; f\left(\frac{\mathbf{area}(\mathcal{M}_s)}{\mathbf{area}(\mathcal{M})}\right). \tag{16}$$

Thus, f is the map that recovers the parameter s from the fractional area of the sub-manifold $\mathcal{M}_s$. Equation (16) can be more convenient to work with than equation (15), as it avoids an explicit function inversion step. While $G_s(\cdot)$ and $g(\cdot, \cdot)$ do not admit equally intuitive interpretations, they can often

be determined from the general form of σ, since many of the details vanish due to normalization. A good example of how this can be done is provided by the uniform parametrization derived for spherical triangles, which we discuss below.

Step 4 above is the only step that is not purely mechanical, as it involves function inversion. When this step can be carried out symbolically, the end result is a closed-form uniform parametrization ψ from $[0,1]^2$ to the manifold $\mathcal{M}$. Closed-form expressions are usually advantageous, both in terms of simplicity and efficiency. Of the two inversions entailed in step 4, it is typically equation (6) that is the more troublesome, and frequently resists symbolic solution. In such a case, it is always possible to perform the inversion numerically using a root-finding method such as Newton's method; of course, one must always weigh the cost of drawing samples against the benefits conferred by the resulting importance sampling and stratification. When numerical inversion is involved, the uniform mapping is less likely to result in a net gain in efficiency.

The steps outlined in Figure 4 generalize very naturally to the construction of volume-preserving parametrizations for arbitrary k-manifolds. For any $2 \leq k \leq n$, step 3 entails a sequence of k cumulative distribution functions, each dependent upon all of its predecessors, and step 4 requires the cascaded inversion of all k distributions, in the order of their definition. Step 5 entails a k-way function composition. In the remainder of these notes, we will consider only the case where $k = 2$ and $n \in \{2, 3\}$; that is, we will only consider the problem of generating samples over 2-manifolds (surfaces) in $\mathbb{R}^2$ or $\mathbb{R}^3$.

In the following sections we will apply the "recipe" given in Figure 4 to derive a number of useful uniform parametrizations. Each will be expressed in closed form since the functions F and G_s will be invertible symbolically; however, in the case of spherical triangles it will not be trivial to invert F.

2.1 Sampling Planar Triangles

As a first example of applying the steps in Figure 4, we shall derive a uniform parametrization for an arbitrary triangle ABC in the plane. We begin with an obvious parametrization from $[0,1] \times [0,1]$ to a given triangle in terms of barycentric coordinates. That is, let

$$\phi(s,t) = (1-s)A + s(1-t)B + stC. \tag{17}$$

It is easy to see that ϕ is a smooth mapping that is bijective except when $t = 0$, which is a set of measure zero. Since the codomain of ϕ is $\mathbb{R}^2$, σ is

SamplePlanarTriangle(**real** ξ_1, **real** ξ_2)

Compute the warping function $(\xi_1, \xi_2) \mapsto (s, t)$.
  $s \leftarrow \sqrt{\xi_1}$;
  $t \leftarrow \xi_2$;
Plug the warped coords into the original parametrization.
  $\mathbf{P} \leftarrow (1 - s)A + s(1 - t)B + stC$;
  return P;
  end

Figure 5: *Algorithm for computing a uniform parametrization of the triangle with vertices A, B, and C. This mapping can be used for uniform or stratified sampling.*

simply the Jacobian of ϕ. After a somewhat tedious computation, we obtain

$$\det(D\phi) = 2cs, \tag{18}$$

where c is the area of the triangle. From equations (6) and (7) we obtain

$$F(s) \;=\; s^2 \qquad \text{and} \qquad G_s(t) \;=\; t. \tag{19}$$

In both cases the constant c disappears as a result of normalization. These functions are trivial to invert, resulting in

$$f(z) \;=\; \sqrt{z} \qquad \text{and} \qquad g(f(z_1), z_2) \;=\; z_2. \tag{20}$$

Finally, after function composition, we have

$$\begin{aligned} \psi(z_1, z_2) &= \phi(f(z_1), g(f(z_1), z_2)) \\ &= (1 - \sqrt{z_1})A + \sqrt{z_1}(1 - z_2)B + \sqrt{z_1}z_2C. \end{aligned} \tag{21}$$

Figure 5 shows the final algorithm. If ξ_1 and ξ_2 are independent random variables, uniformly distributed over the interval $[0, 1]$, then the resulting points will be uniformly distributed over the triangle.

2.2 Sampling the Unit Disk

Next, we derive a uniform parametrization for a unit-radius disk D in the plane, centered at the origin. We start with the parametrization from $[0, 1] \times [0, 1]$ to D given by

$$\phi(s, t) = s\,[\,\cos(2\pi t)\mathbf{x} + \sin(2\pi t)\mathbf{y}\,], \tag{22}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the orthogonal unit vectors in the plane. Again, ϕ is a smooth mapping that is bijective except when $s = 0$. Computing the Jacobian of ϕ, we obtain

$$\det(D\phi) \;=\; 2\pi s. \tag{23}$$

The remaining steps proceed precisely as in the case of the planar triangle; in fact, the distributions F and G turn out to be identical. Thus, we obtain

$$\begin{aligned} \psi(z_1, z_2) &= \phi(\sqrt{z_1}, z_2) \\ &= \sqrt{\xi_1}\,[\cos(2\pi\xi_2)\mathbf{x} + \sin(2\pi\xi_2)\mathbf{y}]\,. \end{aligned} \tag{24}$$

The resulting algorithm for sampling the unit disk is exactly analogous to the algorithm shown in Figure 5 for sampling planar triangles.

2.3 Sampling the Unit Hemisphere

As a first example of applying the steps in Figure 4 to a surface in $\mathbb{R}^3$, we shall derive the well-known uniform parametrization for the unit-radius hemisphere centered at the origin. First, we define a parametrization using spherical coordinates:

$$\phi(s,t) = \begin{bmatrix} \sin\left(\frac{\pi s}{2}\right)\cos(2\pi t) \\ \sin\left(\frac{\pi s}{2}\right)\sin(2\pi t) \\ \cos\left(\frac{\pi s}{2}\right) \end{bmatrix}. \tag{25}$$

Here the parameter s defines the polar angle and t defines the azimuthal angle. Since the codomain of ϕ is $\mathbb{R}^3$, we can apply equation (13). Since

$$\phi_s(s,t) \times \phi_t(s,t) = \pi^2 \begin{bmatrix} \cos\left(\frac{\pi s}{2}\right)\cos(2\pi t) \\ \cos\left(\frac{\pi s}{2}\right)\sin(2\pi t) \\ -\sin\left(\frac{\pi s}{2}\right) \end{bmatrix} \times \begin{bmatrix} -\sin\left(\frac{\pi s}{2}\right)\sin(2\pi t) \\ \sin\left(\frac{\pi s}{2}\right)\cos(2\pi t) \\ 0 \end{bmatrix},$$

we obtain

$$\begin{aligned} \sigma(s,t) &= \|\,\phi_s(s,t) \times \phi_t(s,t)\,\| \\ &= \pi^2 \sin\left(\frac{\pi s}{2}\right). \end{aligned} \tag{26}$$

It then follows easily that

$$F(s) \;=\; 1 - \cos\left(\frac{\pi s}{2}\right) \qquad \text{and} \qquad G_s(t) \;=\; t,$$

which are trivial to invert, resulting in

$$f(z) \;=\; \frac{2\cos^{-1}(1-z)}{\pi} \qquad \text{and} \qquad g(f(z_1), z_2) \;=\; z_2.$$

Composing f and g with ϕ results in

$$\psi(z_1, z_2) = \begin{bmatrix} \sqrt{z_1(2-z_1)}\cos(2\pi z_2) \\ \sqrt{z_1(2-z_1)}\sin(2\pi z_2) \\ 1 - z_1 \end{bmatrix}. \tag{27}$$

Here the s coordinate of the parametrization uniformly selects the z-coordinate between $z = 1$ and $z = 0$, while the t coordinate parameterizes the resulting circle in the z-plane. The form of ψ can be simplified somewhat by substituting $1 - z_1$ for z_1, which does not alter the distribution. The simplified version is

$$\psi(z_1, z_2) = \begin{bmatrix} \sqrt{1-z_1^2}\cos(2\pi z_2) \\ \sqrt{1-z_1^2}\sin(2\pi z_2) \\ z_1 \end{bmatrix}. \tag{28}$$

2.4 Sampling a Phong Lobe

Now suppose that we wish to sample the hemisphere according to a Phong distribution rather than uniformly; that is, with a density proportional to the cosine of the polar angle to a power. To do this we simply include a non-constant relative density function, ρ, in the definition of σ given in equation (26). That is, we let

$$\begin{aligned} \sigma(s,t) &= \rho(s,t)\,\sin\Big(\frac{\pi s}{2}\Big) \\ &= \cos^k\Big(\frac{\pi s}{2}\Big)\sin\Big(\frac{\pi s}{2}\Big), \end{aligned} \tag{29}$$

where k is the Phong exponent. It follows that

$$F(s) \;=\; \cos^{k+1}\Big(\frac{\pi s}{2}\Big) \qquad \text{and} \qquad G_s(t) \;=\; t,$$

which implies that

$$f(z) \;=\; \frac{2}{\pi}\cos^{-1} z^{\frac{1}{n+1}} \qquad \text{and} \qquad g(f(z_1), z_2) \;=\; z_2.$$

Finally, we have

$$\psi(z_1, z_2) = \begin{bmatrix} \sqrt{1 - z_1^{\frac{2}{k+1}}} \cos(2\pi z_2) \\ \sqrt{1 - z_1^{\frac{2}{k+1}}} \sin(2\pi z_2) \\ z_1^{\frac{1}{k+1}} \end{bmatrix}, \tag{30}$$

or, equivalently,

$$\psi(z_1, z_2) = \begin{bmatrix} \sqrt{1 - w^2} \cos(2\pi z_2) \\ \sqrt{1 - w^2} \sin(2\pi z_2) \\ w \end{bmatrix}, \tag{31}$$

where $w = \sqrt[k+1]{z_1}$.

2.5 Sampling Spherical Triangles

We shall now derive a uniform parametrization for an arbitrary spherical triangle, which is significantly more challenging than the cases we've considered thus far. Let T denote the spherical triangle with vertices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, as shown in Figure 6. Such a triangle can be parameterized by using the first coordinate s to select the edge length b_s, which in turn defines sub-triangle $\mathrm{T}_s \subset \mathrm{T}$, and the second coordinate t to select a point along the edge $\mathbf{BC}_s$, as shown in Figure 6. This parameterization can be expressed as

$$\phi(s,t) \;=\; \mathbf{slerp}(\,\mathbf{B},\, \mathbf{slerp}(\,\mathbf{A}, \mathbf{C}, s\,),\, t\,), \tag{32}$$

where $\mathbf{slerp}(\mathbf{A}, \mathbf{C}, s)$ is the *spherical linear interpolation* function that generates points along the great arc connecting $\mathbf{A}$ and $\mathbf{C}$ (according to arc length) as s varies from 0 to 1. The **slerp** function can be defined as

$$\mathbf{slerp}(\mathbf{x}, \mathbf{y}, s) \;=\; \mathbf{x}\cos(\theta s) + [\,\mathbf{y} \,|\, \mathbf{x}\,]\sin(\theta s), \tag{33}$$

where $\theta = \cos^{-1} \mathbf{x} \cdot \mathbf{y}$, and $[\,\mathbf{y} \,|\, \mathbf{x}\,]$ denotes the normalized component of the vector $\mathbf{y}$ that is orthogonal to the vector $\mathbf{x}$; that is

$$[\,\mathbf{y} \,|\, \mathbf{x}\,] \;\equiv\; \frac{(\mathbf{I} - \mathbf{x}\mathbf{x}^{\mathrm{T}})\,\mathbf{y}}{||(\mathbf{I} - \mathbf{x}\mathbf{x}^{\mathrm{T}})\,\mathbf{y}\,||}, \tag{34}$$

where $\mathbf{x}$ is assumed to be a unit vector. Note that the matrix $\mathbf{I} - \mathbf{x}\mathbf{x}^{\mathrm{T}}$ is a projection matrix that projects onto the plane orthogonal to the unit vector

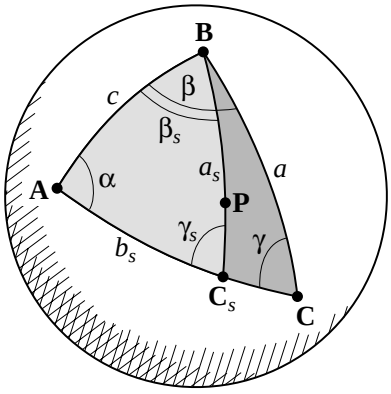


Figure 6: *Parameter s controls the edge length b_s, which determines the vertex $\mathbf{C}_s$, and consequently sub-triangle T_s. Parameter t then selects a point $\mathbf{P}$ along the arc between $\mathbf{C}_s$ and $\mathbf{B}$. Not shown is the length of the arc $\mathbf{AC}$, which is b.*

x. From the above definition of ϕ it is now possible to derive the function σ using equation (13). We find that σ is of the form

$$\sigma(s,t) \;=\; h(s)\sin(\,a_s t\,) \tag{35}$$

for some function h, where a_s is the length of the moving edge $\mathbf{BC}_s$ as a function of s. The exact nature of h is irrelevant, however, as it will not be needed to compute F, and it is eliminated from G_s by normalization. Thus, we have

$$F(\,s\,) \;=\; \frac{\mathbf{area}(\,\mathrm{T}_s\,)}{\mathbf{area}(\mathrm{T})} \tag{36}$$

$$G_s(\,t\,) \;=\; \frac{1-\cos(\,a_s t\,)}{1-\cos a_s}. \tag{37}$$

It follows immediately from inversion of equation (37) that

$$g(z_1,z_2) \;=\; \frac{1}{a_{z_1}}\cos^{-1}\Big[1-z_2(1-\cos a_{z_1})\Big]. \tag{38}$$

However, solving for f, which is the inverse of F, is not nearly as straightforward. Our approach will be to derive an expression for the function f directly, using equation (16), rather than starting from F and inverting it. To do this, we require several elementary identities from spherical trigonometry. Let $\mathcal{A}$ denote the surface area of the spherical triangle T with vertices

A, **B** and **C**. Let a, b, and c denote the edge lengths of T,

$$\begin{aligned} a &= \cos^{-1} \mathbf{B} \cdot \mathbf{C}, \\ b &= \cos^{-1} \mathbf{A} \cdot \mathbf{C}, \\ c &= \cos^{-1} \mathbf{A} \cdot \mathbf{B}, \end{aligned}$$

and let α, β, and γ denote the three internal angles, which are the dihedral angles between the planes containing the edges. See Figure 6. Listed below are a few well-known identities for spherical triangles:

$$\mathcal{A} = \alpha + \beta + \gamma - \pi \tag{39}$$

$$\frac{\sin\alpha}{\sin a} = \frac{\sin\beta}{\sin b} = \frac{\sin\gamma}{\sin c} \tag{40}$$

$$\cos\alpha = -\cos\beta\,\cos\gamma \;+\; \sin\beta\,\sin\gamma\,\cos a \tag{41}$$

$$\cos\beta = -\cos\gamma\,\cos\alpha \;+\; \sin\gamma\,\sin\alpha\,\cos b \tag{42}$$

$$\cos\gamma = -\cos\beta\,\cos\alpha \;+\; \sin\beta\,\sin\alpha\,\cos c \tag{43}$$

Each of these identities will be employed in deriving uniform parametrizations, either for spherical triangles or projected spherical polygons, which will be described in the following section. Equation (39) is known as Girard's formula, equation (40) is the spherical law of sines, and equations (41), (42), and (43) are spherical cosine laws for angles [4].

Our task will be to construct $f : [0,1] \to \mathbb{R}$ such that $f(\mathcal{A}_s/\mathcal{A}) = s$, where the parameter $s \in [0,1]$ selects the sub-triangle T_s and consequently determines the area $\mathcal{A}_s$. Specifically, the sub-triangle T_s is formed by choosing a new vertex $\mathbf{C}_s$ on the great arc between **A** and **C**, at an arc length of $b_s = sb$ along the arc from **A**, as shown in Figure 6. The point **P** is finally chosen on the arc between **B** and $\mathbf{C}_s$, according to the parameter t.

To find the parameter s that corresponds to the fractional area $\mathcal{A}_s/\mathcal{A}$, we first solve for $\cos b_s$ in terms of $\mathcal{A}_s$ and various constants associated with the triangle. From equations (39) and (42) we have

$$\begin{aligned} \cos b_s &= \frac{\cos\gamma_s\,\cos\alpha \;+\; \cos\beta_s}{\sin\gamma_s\,\sin\alpha} \\ &= \frac{-\cos(\mathcal{A}_s - \alpha - \beta_s)\,\cos\alpha \;+\; \cos\beta_s}{-\sin(\mathcal{A}_s - \alpha - \beta_s)\,\sin\alpha} \\ &= \frac{\cos(\Delta - \beta_s)\,\cos\alpha \;-\; \cos\beta_s}{\sin(\Delta - \beta_s)\,\sin\alpha}, \end{aligned} \tag{44}$$

where we have introduced $\Delta \equiv \mathcal{A}_s - \alpha$. We now eliminate β_s to obtain a function that depends only on area and the fixed parameters: in particular,

SampleSphericalTriangle(**real** ξ_1, **real** ξ_2)

Use one random variable to select the new area.

$\mathcal{A}_s \leftarrow \xi_1 \mathcal{A}$;

Save the sine and cosine of the angle Δ.

$p \leftarrow \sin(\mathcal{A}_s - \alpha)$;

$q \leftarrow \cos(\mathcal{A}_s - \alpha)$;

Compute the pair (u, v) *that determines* $\sin\beta_s$ *and* $\cos\beta_s$.

$u \leftarrow q - \cos\alpha$;

$v \leftarrow p + \sin\alpha\cos c$;

Compute the s *coordinate as normalized arc length from* $\mathbf{A}$ *to* $\mathbf{C}_s$.

$$s \leftarrow \frac{1}{b}\cos^{-1}\left[\frac{(v\,q\,-\,u\,p)\,\cos\alpha\,-\,v}{(v\,p\,+\,u\,q)\,\sin\alpha}\right];$$

Compute the third vertex of the sub-triangle.

$\mathbf{C}_s \leftarrow \mathbf{slerp}(\mathbf{A}, \mathbf{C}, s)$;

Compute the t *coordinate using* $\mathbf{C}_s$ *and* ξ_2.

$$t \leftarrow \frac{\cos^{-1}\left[1 - \xi_2(1 - \mathbf{C}_s \cdot \mathbf{B})\right]}{\cos^{-1}\mathbf{C}_s \cdot \mathbf{B}};$$

Construct the corresponding point on the sphere.

$\mathbf{P} \leftarrow \mathbf{slerp}(\mathbf{B}, \mathbf{C}_s, t)$;

return $\mathbf{P}$;

end

Figure 7: *A uniform parametrization for an arbitrary spherical triangle* **ABC**. *This procedure can be easily optimized to remove the inverse cosines used to compute the warped coordinates* s *and* t, *since the* **slerp** *function uses the cosine of its scalar argument.*

we shall construct a function of only Δ, α, and c. We accomplish this by using spherical trigonometry to find expressions for both $\sin\beta_s$ and $\cos\beta_s$. From equation (39) and plane trigonometry it follows that

$$\cos\gamma_s \;=\; -\cos(\Delta - \beta_s) \;=\; \sin\Delta\sin\beta_s - \cos\Delta\cos\beta_s. \tag{45}$$

Combining equation (45) with equation (43) we have

$$(\cos\Delta \;-\; \cos\alpha)\,\cos\beta_s \;+\; (\sin\Delta \;+\; \sin\alpha\,\cos c)\,\sin\beta_s \;=\; 0. \tag{46}$$

Consequently, $\sin\beta_s = -ru$ and $\cos\beta_s = rv$ where

$$\begin{aligned} u &\equiv \cos\Delta - \cos\alpha, \\ v &\equiv \sin\Delta + \sin\alpha\,\cos c, \end{aligned}$$

and r is a common factor that cancels out in our final expression, so it is irrelevant. Simplifying equation (44) using these new expressions for $\sin\beta_s$ and $\cos\beta_s$, we obtain an expression for $\cos b_s$ in terms of Δ, u, v, and α. It then follows that

$$s = \frac{1}{b}\,\cos^{-1}\left[\frac{(\,v\cos\Delta - u\sin\Delta\,)\,\cos\alpha - v}{(\,v\sin\Delta + u\cos\Delta\,)\,\sin\alpha}\right], \tag{47}$$

since $s = b_s/b$. Note that $\cos b_s$ determines b_s, since $0 < b_s < \pi$, and that b_s in turn determines the vertex $\mathbf{C}_s$. The algorithm shown in Figure 8 computes a uniform map from the unit square onto the triangle T; it takes two variables ξ_1 and ξ_2, each in the unit interval, and returns a point $\mathbf{P} \in \mathrm{T} \subset \mathbb{R}^3$. If ξ_1 and ξ_2 are uniformly distributed random variables in $[0,1]$, the algorithm will produce a random variable $\mathbf{P}$ that is uniformly distributed over the surface of the spherical triangle T.

The procedure in Figure 8 explicitly warps the coordinates (ξ_1, ξ_2) into the coordinates (s,t) in such a way that the resulting parametrization is uniform. If implemented exactly as shown, the procedure performs a significant amount of unnecessary computation. Most significantly, all of the inverse cosines can be eliminated by substituting the equation (33) for the **slerp** function and then simplifying [3]. Also, $\cos\alpha$, $\sin\alpha$, $\cos c$, and $[\,\mathbf{C}\,|\,\mathbf{A}\,]$, which appear in the expression for $\mathbf{slerp}(\mathbf{A}, \mathbf{C}, s)$, need only be computed once per triangle rather than once per sample.

Results of the algorithm are shown in Figure 1. On the left, the samples are identically distributed, which produces a pattern equivalent to that obtained by rejection sampling; however, each sample is guaranteed to fall within the triangle. The pattern on the right was generated by partitioning the unit square into a regular grid and choosing one pair (ξ_1, ξ_2) uniformly from each grid cell, which corresponds to stratified sampling. The advantage of stratified sampling is evident in the resulting pattern; the samples are more evenly distributed, which generally reduces the variance of Monte Carlo estimates based on these samples. The sampling algorithm can be applied to spherical polygons by decomposing them into triangles and performing stratified sampling on each component independently, which is analogous to the method for planar polygons described by Turk [12]. This is one means

SampleSphericalTriangle(**real** ξ_1, **real** ξ_2)

Use one random variable to select the new area.
$\mathcal{A}_s \leftarrow \xi_1 * \mathcal{A}$;
Save the sine and cosine of the angle Δ.
$p \leftarrow \sin(\mathcal{A}_s - \alpha)$;
$q \leftarrow \cos(\mathcal{A}_s - \alpha)$;
Compute the pair (u, v) *that determines* β_s.
$u \leftarrow q - \cos\alpha$;
$v \leftarrow p + \sin\alpha * \cos c$;
Let s *be the cosine of the new edge length* b_s.
$s \leftarrow \dfrac{[v * q \ - \ u * p] * \cos\alpha \ - \ v}{[v * p \ + \ u * q] * \sin\alpha}$;
Compute the third vertex of the sub-triangle.
$\mathbf{C}_s \leftarrow s * \mathbf{A} + \sqrt{1 - s^2} * [\,\mathbf{C} \,|\, \mathbf{A}\,]$;
Use the other random variable to select $\cos(t)$.
$t \leftarrow 1 - \xi_2 * (1 - \mathbf{C}_s \cdot \mathbf{B})$;
Construct the corresponding point on the sphere.
$\mathbf{P} \leftarrow t * \mathbf{B} \ + \ \sqrt{1 - t^2} * [\,\mathbf{C}_s \,|\, \mathbf{B}\,]$;
return P;
end

Figure 8: *This is the optimized algorithm for generating uniformly distributed samples within an arbitrary spherical triangle. All of the inverse trigonometric functions and calls to* **slerp** *have been removed using straightforward algebraic simplifications.*

of sampling the solid angle subtended by a polygon. We discuss another approach in the following section.

2.6 Sampling Projected Spherical Polygons

In this section we will see an example in which the inversion of the F function can not be done symbolically; consequently, we will resort to either approximate inversion, or inversion via a root finder.

The dot product $\vec{\omega} \cdot \vec{n}$ appearing in equation (1) is the ubiquitous "cosine" factor that appears in nearly every illumination integral. Since it is often infeasible to construct a random variable that mimics the full integrand, we settle for absorbing the cosine term into the sampling distribution; this

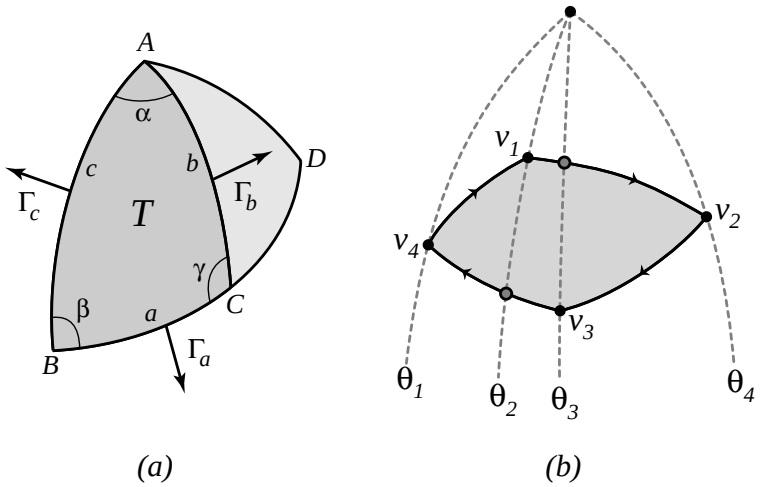


Figure 9: *(a) Spherical triangle T. We consider the projected area of triangle T as a function of α, keeping vertices A and B, and angle β fixed. (b) Partitioning a spherical polygon by great circles passing through the poles and the vertices.*

compromise is a useful special case of *importance sampling*. In this section we address the problem of generating stratified samples over the solid angle subtended by arbitrary polygons, while taking the cosine weighting into account, as shown in Figure 2. The combination of stratification and importance sampling, even in this relatively weak form, can significantly reduce the variance of the associated Monte Carlo estimator [3, 9].

We now describe a new technique for Monte Carlo sampling of spherical polygons with a density proportional to the cosine from a given axis which, by Nusselt's analogy, is equivalent to uniformly sampling the projection of the spherical polygon onto the $z = 0$ plane. The technique handles polygons directly, without first partitioning them into triangles, and is ideally suited for stratified sampling. The Jacobian of the bijection from the unit square to the polygon can be made arbitrarily close to the cosine density, making the statistical bias as close to zero as desired. After preprocessing a polygon with n vertices, which can be done in $O(n^2 \log n)$ time, each sample can be generated in $O(n)$ time.

Let $\mathcal{P}$ denote a spherical polygon. To help in defining the mapping $\psi : [0,1]^2 \rightarrow \mathcal{P}$, we first derive several basic expressions that pertain to spherical triangles. Let T be a spherical triangle, and consider the family of sub-triangles shown in Figure 9a, where the unit vectors A and B and the internal angle β are all fixed, but the internal angle α is allowed to sweep from 0 to the last vertex. Our task in this section is to express a (which is

the length of edge BC) and $\cos b$ as functions of α. From equation (41) we have

$$\cos a \;=\; \frac{\cos\alpha + \cos\beta\cos\gamma}{\sin\beta\sin\gamma}. \tag{48}$$

Let Γ_a, Γ_b, and Γ_c denote the outward unit normals for each edges of the triangle, as shown in Figure 9a. Then $\cos\gamma = -\Gamma_a \cdot \Gamma_b$, where Γ_b can be expressed as

$$\Gamma_b \;=\; -\Gamma_c\cos\alpha + (\Gamma_c \times A)\sin\alpha. \tag{49}$$

Also, from equation (40) we have $\sin\gamma = \sin c\,\sin\alpha/\sin a$. Therefore,

$$a(\alpha) \;=\; \tan^{-1}\!\left(\frac{\sin\alpha}{c_1\cos\alpha - c_2\sin\alpha}\right), \tag{50}$$

where the constants c_1 and c_2 are given by

$$c_1 = \frac{(\Gamma_a\cdot\Gamma_c)\cos\beta + 1}{\sin\beta\,\sin c}, \quad c_2 = \frac{(\Gamma_a\cdot\Gamma_c\times A)\cos\beta}{\sin\beta\,\sin c}. \tag{51}$$

These constants depend only on the fixed features of the triangle, as the vectors Γ_a and Γ_c do not depend on α. It is now straightforward to find $\cos b$ as a function of α, which we shall denote by $z(\alpha)$. Specifically,

$$z(\alpha) \;=\; (B\cdot\mathbf{N})\cos a(\alpha) \;+\; (D\cdot\mathbf{N})\sin a(\alpha), \tag{52}$$

where D is a point on the sphere that is orthogonal to B, and on the great circle through B and C. That is,

$$D \;=\; (\mathbf{I} - BB^{\mathrm{T}})C. \tag{53}$$

We now show how to sample an arbitrary spherical polygon according to a cosine distribution. The function $a(\alpha)$ will be used to invert a cumulative marginal distribution over the polygon, as a great arc sweeps across the polygon, while $z(\alpha)$ will be used to sample one-dimensional vertical slices of the polygon.

2.6.1 The Cumulative Marginal Distribution

We break the problem of computing the bijection $\psi : [0,1]^2 \to \mathcal{P}$ into two parts. First, we define a sequence of sub-polygons of $\mathcal{P}$ in much the same way that we parameterized the triangle T above; that is, we define $\mathcal{P}(\theta)$ to

be the intersection of $\mathcal{P}$ with a lune [3] whose internal angle is θ, and with one edge passing through an extremal vertex of $\mathcal{P}$. Next we define a *cumulative marginal distribution* $F(\theta)$ that gives the area of polygon $\mathcal{P}(\theta)$ projected onto the plane orthogonal to $\mathbf{N}$, which is simply the cosine-weighted area of $\mathcal{P}$. Then F is a strictly monotonically increasing function of θ. By inverting this function we arrive at the first component of our sampling algorithm. That is, if ξ_1 is a uniformly distributed random variable in $[0, 1]$, and if $\hat{\theta}$ is given by

$$\hat{\theta} \;=\; F^{-1}(\rho\,\xi_1), \tag{54}$$

then $\hat{\theta}$ defines the great circle from which to draw a sample.

To find F, we first consider the spherical triangle T and its family of sub-triangles. The projected area of the triangle T, which we denote by ρ, follows immediately from Lambert's formula for computing the irradiance from a polygonal luminaire [2]. That is

$$\rho \;=\; -(a\Gamma_a + b\Gamma_b + c\Gamma_c)\cdot\mathbf{N}, \tag{55}$$

where Γ_a, Γ_b, and Γ_c are outward normals of the triangle T, as shown in Figure 9a. If we now constrain T to be a *polar triangle*, with vertex A at the pole of the hemisphere ($A = \mathbf{N}$), then ρ becomes a very simple function of α. Specifically,

$$\rho(\alpha) \;=\; -a(\alpha)\,(\Gamma_a\cdot\mathbf{N}), \tag{56}$$

where $\Gamma_a\cdot\mathbf{N}$ is fixed; this follows from the fact that both Γ_b and Γ_c are orthogonal to $\mathbf{N}$. Equation (56) allows us to easily compute the function $F(\alpha)$ for any collection of spherical polygons whose vertices all lie on the lune with vertices at A and $-A$ as shown in Figure 11, where we restrict our attention to the *positive* or upper half of the lune. Thus,

$$F(\theta) \;=\; \sum_{i=1}^{k} \eta_i\, a_i(\theta - \theta_k), \tag{57}$$

for $\theta \in [\theta_k, \theta_{k+1}]$, where the constants $\eta_1, \eta_2, \ldots, \eta_k$ account for the slope and orientation of the edges; that is, edges that result in clockwise polar triangles are positive, while those forming counter-clockwise triangles are negative.

[3] A *lune* is a spherical triangle with exactly two vertices, which are antipodal; that is, the two verticies are at opposite poles of the sphere.

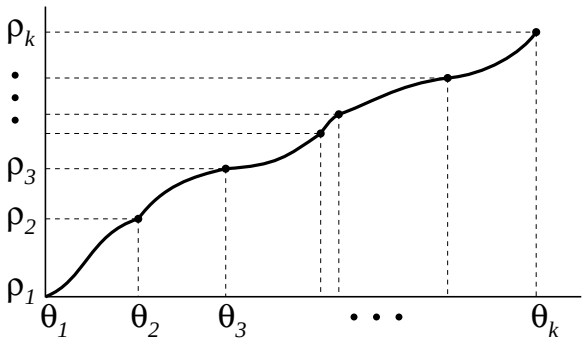


Figure 10: *The cumulative marginal distribution function F as a function of the angle θ. At each value of θ_i, the abscissa is the form factor from the origin to the polygon that is within the range $[\theta_1, \theta_i]$. This function is strictly monotonically increasing, with at most $n-2$ derivative discontinuities, where n is the number of vertices in the polygon. The fluctuations in F have been greatly exaggerated for purposes of illustration.*

We now extend $F(\theta)$ to a general spherical polygon $\mathcal{P}$ by slicing $\mathcal{P}$ into lunes with the above property; that is, we partition $\mathcal{P}$ into smaller polygons by passing a great arc through each vertex, as shown in Figure 9b. Then for any spherical polygon, we can evaluate $F(\theta)$ exactly for any value of θ by virtue of equation (50). The resulting function F is a piecewise-continuous strictly monotonically increasing function with at most $n-2$ discontinuities, where n is the number of vertices in the polygon. See Figure 10. This function is precisely the *cumulative marginal distribution function* that we must invert to perform the first stage of cosine-weighted sampling. Because it is monotonically increasing, its inverse is well-defined.

2.6.2 The Sampling Algorithm

Given two variables ξ_1 and ξ_2 in the interval $[0, 1]$ we will compute the corresponding point $\mathbf{P} = \psi(\xi_1, \xi_2)$ in the polygon $\mathcal{P}$. We use ξ_1 to determine an angle $\hat{\theta}$, as described above, and ξ_2 to select the height $\hat{z}$ according to the resulting *conditional density* defined along the intersection of the polygon $\mathcal{P}$ and the great circle at $\hat{\theta}$.

To compute $\hat{\theta}$ using equation (54), we proceed in two steps. First, we find the lune from which $\hat{\theta}$ will be drawn. This corresponds to finding the integer k such that

$$\frac{\rho_k}{\rho_{tot}} \;\leq\; \xi_1 \;\leq\; \frac{\rho_{k+1}}{\rho_{tot}}. \tag{58}$$

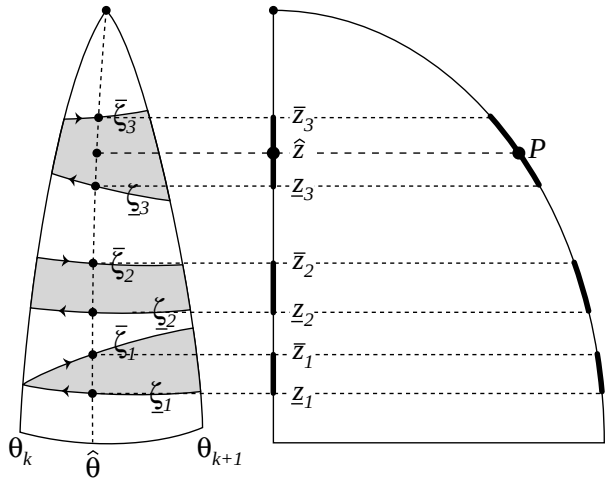


Figure 11: *On the left is an illustration of a single lune with a collection of arcs passing through it, and the points at which a great circle at $\hat{\theta}$ intersects them. On the right is a cross-section of the circle, showing the heights $\underline{z}_1$, $\overline{z}_1$, corresponding to these intersection points.*

Next, we must invert F as it is defined on this interval. Given the nature of F, as defined in equations (50) and (57), it is unlikely that this can be done symbolically in general, so we seek a numerical approximation. This is the *only* step in the algorithm which is not computed exactly; thus, any bias that is introduced in the sampling is a result of this step alone.

Approximate numerical inversion is greatly simplified by the nature of F within each lune. Since F is extremely smooth and strictly monotonic, we can approximate F^{-1} directly to high accuracy with a low-order polynomial. For example, we may use

$$F^{-1}(x) \;\approx\; a + bx + cx^2 + dx^3, \tag{59}$$

where we set

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \;=\; V^{-1} \begin{bmatrix} \theta_k \\ \theta_k + \delta_1 \\ \theta_k + \delta_2 \\ \theta_{k+1} \end{bmatrix}. \tag{60}$$

Here V is the Vandermonde matrix formed from $F(\theta_k)$, $F(\theta_k+\delta_1)$, $F(\theta_k+\delta_2)$, and $F(\theta_{k+1})$. Coupling this approximation with a Newton iteration can, of course, compute the function inverse to any desired numerical accuracy,

making the bias effectively zero. However, such a high degree of accuracy is not warranted for a typical Monte Carlo simulation.

Once the angle $\hat{\theta}$ has been computed using ξ_1, we then compute $\hat{z}$ using ξ_2. This corresponds to sampling $\hat{z}$ according to the *conditional density function* corresponding to the choice of $\hat{\theta}$. This conditional density function is defined on the intervals

$$[\underline{z}_1, \overline{z}_1] \cup [\underline{z}_2, \overline{z}_2] \cup \cdots \cup [\underline{z}_n, \overline{z}_n],$$

which correspond to the intersection points shown in Figure 11. These intervals are computed using equation (52). The conditional density is proportional to z^2 within these intervals, which distributes the samples vertically according to the cosine of the angle from the pole. The most costly part of sampling according to this density is normalization. We define

$$Z_j \;\equiv\; \sum_{i=1}^{j} \left(\overline{z}_i^2 - \underline{z}_i^2\right). \tag{61}$$

Then Z_n is the normalization constant. The random variable ξ_2 then selects the interval by finding $1 \le \ell \le n$ such that

$$\frac{Z_{\ell-1}}{Z_n} \;\le\; \xi_2 \;\le\; \frac{Z_\ell}{Z_n} \tag{62}$$

where $Z_0 \equiv 0$. Finally, the height of $\mathbf{P} = \psi(\xi_1, \xi_2)$ is

$$\hat{z} \;=\; \sqrt{\xi_2 - Z_{\ell-1} + \underline{z}_\ell}, \tag{63}$$

and the point itself is

$$\mathbf{P} \;=\; (\omega \cos\hat{\theta},\, \omega \sin\hat{\theta},\, \hat{z}), \tag{64}$$

where $\omega = \sqrt{1 - \hat{z}^2}$.

The algorithm described above also works for spherical polygons $\mathcal{P}$ that surround the pole of the sphere. In this case, each lune has an odd number of segments crossing it, and $\overline{z}_n = 1$ must be added to the list of heights defined by each $\hat{\theta}$ in sampling from the conditional distribution.

The algorithm described above is somewhat more costly than the algorithm for uniform sampling of spherical triangles [3] for two reasons: 1) evaluating piecewise continuous functions requires some searching, and 2) the cumulative marginal distribution cannot be inverted exactly. Furthermore, the sampling algorithm requires some preprocessing to make both of these operations efficient and accurate.

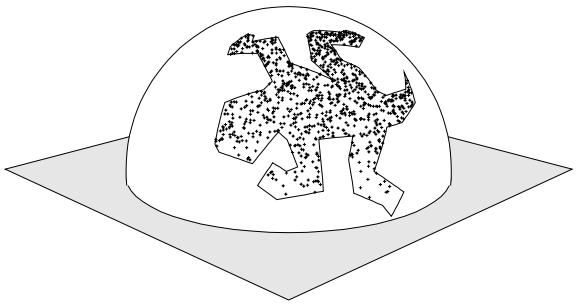


Figure 12: *A non-convex spherical polygon with cosine-weighted samples generated with the proposed mapping.*

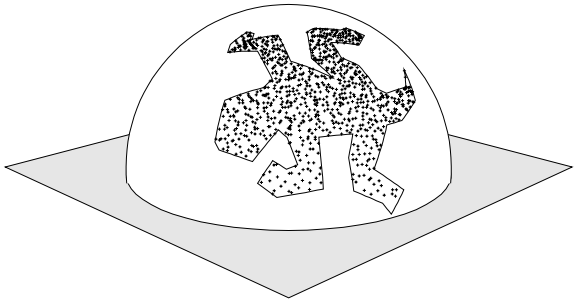


Figure 13: *A non-convex spherical polygon with cosine-weighted and stratified samples generated with the proposed mapping.*

Pre-processing includes partitioning the polygon into lunes, computing the constants c_1 and c_2 defined in equation (51) for each resulting edge, and sorting the line segments within each lune into increasing order. In the worst case, there may be $n-2$ lunes, with $\Omega(n)$ of them containing $\Omega(n)$ segments. Thus, creating them and sorting them requires $O(n^2 \log n)$ in the worst case. For convex polygons, this drops to $O(n)$, since there can be only two segments per lune.

Once the pre-processing is done, samples can be generated by searching for the appropriate $[\theta_k, \theta_{k+1}]$ interval, which can be done in $O(\log n)$ time, and then sampling according to the conditional distribution, which can be done on $O(n)$ time. The latter cost is the dominant one because all of the intervals must be formed for normalization. Therefore, in the worst case, the cost of drawing a sample is $O(n)$; however, for convex polygons this drops to $O(\log n)$.

Figure 2 shows 900 samples in a spherical quadrilateral, distributed according to the cosine distribution. Note that more of the samples are clus-

tered near the pole than the horizon. Stratification was performed by mapping "jittered" points from the unit square onto the quadrilateral. Figures 12 and 13 show 900 samples distributed according to the cosine density within a highly non-convex spherical polygon. These samples were generated without first partitioning the polygon into triangles. In both of the test cases, the cumulative marginal distribution function F is very nearly piecewise linear, and its inverse can be computed to extremely high accuracy with a piecewise cubic curve.

3 Combining Sampling Strategies

We next explore the idea of constructing effective random variables for Monte Carlo integration by combining two or more simpler random variables. For instance, suppose that we have at our disposal a convenient means of sampling the solid angle subtended by a luminaire, and also a means of sampling a brdf; how are these to be used in concert to estimate the reflected radiance from a surface? While each sampling method can itself serve as the basis of an importance sampling scheme, in isolation neither can reliably predict the shape of the resulting integrand. The problem is that the shape of the brdf may make some directions "important" (i.e. likely to make a large contribution to the integral) while the luminaire, which is potentially orders of magnitude brighter than the indirect illumination, may make other directions "important." The question that we shall address is how to construct an importance sampling method that accounts for all such "hot spots" by combining available sampling methods, but without introducing statistical bias. The following discussion closely parallels the work of Veach [13], who was the first to systematically explore this idea in the context of global illumination.

To simplify the discussion, let us assume that we are attempting to approximate some quantity $\mathcal{I}$, which is given by the integral of an unknown and potentially ill-behaved function f over the domain D:

$$\mathcal{I} = \int_D f(x)\,dx. \tag{65}$$

For instance, f may be the product of incident radiance (direct and indirect), a reflectance function, and a visibility factor, and D may be the either a collection of surfaces or the hemisphere of incident directions; in cases such as these, $\mathcal{I}$ may represent reflected radiance. In traditional *importance sampling*, we select a *probability density function* (pdf) p over D and rewrite

the integral as

$$\mathcal{I} \;=\; \int_D \left[\frac{f(x)}{p(x)}\right] p(x)\,dx \;=\; \left\langle \frac{f(\mathbf{X})}{p(\mathbf{X})} \right\rangle, \tag{66}$$

where $\mathbf{X}$ denotes a random variable on the domain D distributed according to the pdf p, and $\langle\cdot\rangle$ denotes the *expected value* of a random variable. The second equality in equation (66) is simply the definition of expected value. It follows immediately that the *sample mean* of the new random variable $f(\mathbf{X})/p(\mathbf{X})$ is an estimator for $\mathcal{I}$; that is, if

$$\mathcal{E} \;=\; \frac{1}{N}\sum_{i=1}^{N} \frac{f(\mathbf{X}_i)}{p(\mathbf{X}_i)}, \tag{67}$$

for $N \geq 1$, where $\mathbf{X}_1, \ldots, \mathbf{X}_N$ are iid (independent identically distributed) random variables, each distributed according to the pdf p, then $\langle\mathcal{E}\rangle = \mathcal{I}$. Consequently, $\mathcal{E} \approx \mathcal{I}$, and the quality of the approximation can be improved by increasing N, the number of samples, and/or by increasing the similarity between the original integrand f and the pdf p. Since evaluating $f(\mathbf{X}_i)$ is potentially very costly, we wish to pursue the second option to the extent possible. This is precisely the rationale for importance sampling.

3.1 Using Multiple PDFs

Now let us suppose that we have k distinct pdfs, $p_1, p_2, \ldots, p_k$, that each mimic some potential "hot spot" in the integrand; that is, each concentrates samples in a region of the domain where the integrand may be relatively large. For instance, p_1 may sample according to the brdf, concentrating samples around specular directions of glossy surfaces, while $p_2, \ldots, p_k$ sample various luminaires or potential specular reflections. Let us further suppose that for each p_i we draw N_i iid samples, $\mathbf{X}_{i,1}, \mathbf{X}_{i,2}, \ldots, \mathbf{X}_{i,N_i}$, distributed according to p_i. Our goal is to combine them into an estimator $\mathcal{E}$ that has several desirable properties. In particular, we wish to ensure that

1. $\langle\mathcal{E}\rangle \;=\; \mathcal{I}$,
2. $\mathcal{E}$ is relatively easy to compute,
3. $\mathrm{var}(\mathcal{E})$ is small.

That is, we wish to have the expected value of $\mathcal{E}$ match the actual value of the integral, $\mathcal{I}$, to pay a low computational price for drawing each sample,

and to reduce the variance of $\mathcal{E}$ as much as possible, thereby reducing the number of samples required to attain a reliable approximation. The first requirement ensures that the estimator is *unbiased*. Unbiased estimators have the highly desirable property that they allow us to converge to the exact answer by taking a sufficiently large number of samples. As a general rule, this is the first property that any random variable designed for Monte Carlo integration should possess [7, 8].

As we shall see, there is a large family of functions ϕ that meet property 1, leaving much flexibility in choosing one that meets both properties 2 and 3. We begin by identifying such a class of estimators, then imposing the other constraints. First, consider an estimator of the form

$$\mathcal{E}_\phi \equiv \sum_{i=1}^{k} \sum_{j=1}^{N_i} \phi_i(\mathbf{X}_{i,j}) f(\mathbf{X}_{i,j}), \tag{68}$$

for some suitable choice of the functions ϕ_i. That is, let us allow a different function ϕ_i to be associated with the samples drawn from each pdf p_i, and also allow the weight of each sample to depend on the sample itself. Equation (68) is extremely general, and also reasonable, as we can immediately ensure that $\mathcal{E}_\phi$ is unbiased by constraining the functions ϕ_i to be of the form

$$\phi_i(x) \equiv \frac{w_i(x)}{N_i\, p_i(x)}, \tag{69}$$

where for all $x \in D$ and $1 \leq i \leq k$, the *weighting functions* w_i satisfy

$$w_i(x) \geq 0, \tag{70}$$

$$w_1(x) + \cdots + w_k(x) = 1. \tag{71}$$

To see that the resulting estimator is unbiased, regardless of the choice of the weighting functions w_i, provided that they satisfy constraints (70) and (71), let us define $\mathcal{E}_w$ to be the estimator $\mathcal{E}_\phi$ where the ϕ_i are of the form shown

in equation (69), and observe that

$$\begin{aligned}
\langle \mathcal{E}_w \rangle &= \left\langle \sum_{i=1}^{k} \sum_{j=1}^{N_i} \phi_i(\mathbf{X}_{i,j}) f(\mathbf{X}_{i,j}) \right\rangle \\
&= \sum_{i=1}^{k} \sum_{j=1}^{N_i} \langle \phi_i(\mathbf{X}_{i,j}) f(\mathbf{X}_{i,j}) \rangle \\
&= \sum_{i=1}^{k} \sum_{j=1}^{N_i} \int_D \phi_i(x) f(x) p_i(x)\, dx \\
&= \sum_{i=1}^{k} \int_D \frac{w_i(x)}{p_i(x)} f(x) p_i(x)\, dx \\
&= \int_D f(x) \left[\sum_{i=1}^{k} w_i(x) \right] dx \\
&= \int_D f(x)\, dx \\
&= \mathcal{I}.
\end{aligned}$$

Thus, by considering only estimators of the form $\mathcal{E}_w$, we may henceforth ignore property 1 and concentrate strictly on selecting the weighting functions w_i so as to satisfy the other two properties.

3.2 Possible Weighting Functions

In some sense the most obvious weighting functions to employ are given by

$$w_i(x) \equiv \frac{c_i p_i(x)}{q(x)}, \tag{72}$$

where

$$q(x) \equiv c_1 p_1(x) + \cdots + c_k p_k(x), \tag{73}$$

is a pdf obtained by taking a convex combination of the original pdfs; that is, the constants c_i satisfy $c_i \geq 0$ and $c_1 + \cdots + c_k = 1$. Clearly, these w_i are positive and sum to one at each x; therefore the resulting estimator is unbiased, as shown above. This particular choice is "obvious" in the sense that it corresponds exactly to classical importance sampling based on the

pdf defined in equation (73), when a very natural constraint is imposed on $N_1, \ldots, N_k$. To see this, observe that

$$\begin{aligned}
\mathcal{E}_w &= \sum_{i=1}^{k} \sum_{j=1}^{N_i} \frac{w_i(\mathbf{X}_{i,j})}{N_i\, p_i(\mathbf{X}_{i,j})} f(\mathbf{X}_{i,j}) \\
&= \sum_{i=1}^{k} \left[\frac{1}{N_i} \sum_{j=1}^{N_i} \frac{c_i}{q(\mathbf{X}_{i,j})} f(\mathbf{X}_{i,j}) \right] \\
&= \sum_{i=1}^{k} \frac{c_i}{N_i} \left[\sum_{j=1}^{N_i} \frac{f(\mathbf{X}_{i,j})}{q(\mathbf{X}_{i,j})} \right]. \qquad (74)
\end{aligned}$$

Now, let $N = N_1 + \cdots + N_k$ be the total number of samples, and let us further assume that the samples have been partitioned among the pdfs $p_1, \ldots, p_k$ in proportion to the weights $c_1, \ldots, c_k$, that is, with $N_i = c_i N$. Then the ratio c_i/N_i is constant, and equation (74) simplifies to

$$\mathcal{E}_w = \frac{1}{N} \sum_{i=1}^{k} \sum_{j=1}^{N_i} \frac{f(\mathbf{X}_{i,j})}{q(\mathbf{X}_{i,j})}. \qquad (75)$$

Note that in equation (75) all samples are handled in exactly the same manner; that is, the weighting of the samples does not depend on i, which indicates the pdfs they are distributed according to. This is precisely the formula we would obtain if we began with q as our pdf for importance sampling. Adopting Veach's terminology, we shall refer to this particular choice of weighting functions as the *balance heuristic* [13]. Other possibilities for the weighting functions, which are also based on convex combinations of the original pdfs, include

$$w_i(x) = \begin{cases} 1 & \text{if } c_i p_i(x) = \max_j c_j p_j(x) \\ 0 & \text{otherwise} \end{cases}, \qquad (76)$$

and

$$w_i(x) = c_i p_i^m(x) \left[\sum_{j=1}^{k} c_j p_j^m(x) \right]^{-1}, \qquad (77)$$

for some exponent $m \geq 1$. Again, we need only verify that these weighting functions are non-negative and sum to one for all x to verify that they give rise to unbiased estimators. Note, also, that each of these strategies is extremely simple to compute, thus satisfying property 2 noted earlier.

3.3 Obvious is also Nearly Optimal

Let $\widehat{\mathcal{E}}_w$ denote an estimator that incorporates the balance heuristic, and let $\mathcal{E}_w$ be an estimator with any other valid choice of weighting function. Veach has shown [13] that

$$\mathrm{var}\Big(\widehat{\mathcal{E}}_w\Big) \;\leq\; \mathrm{var}(\mathcal{E}_w) + \mathcal{I}^2 \left[\frac{1}{N_{\min}} - \frac{1}{N}\right], \tag{78}$$

where $N_{\min} = \min_i N_i$. Inequality (78) indicates that the variance of the estimator $\widehat{\mathcal{E}}_w$ compares favorably with the optimal strategy, which would be infeasible to determine in any case. In fact, as the number of samples of the least-sampled pdf approaches to infinity, the balance heuristic approaches optimality.

Fortunately, the balance heuristic is also extremely easy to apply; it demands very little beyond the standard requirements of importance sampling, which include the ability to generate samples distributed according to each of the original pdfs p_i, and the ability to compute the density of a given point x with respect to each of the original pdfs [8]. This last requirement simply means that for each $x \in D$ and $1 \leq i \leq k$, we must be able to evaluate $p_i(x)$. Thus, $\widehat{\mathcal{E}}_w$ satisfies all three properties noted earlier, and is therefore a reasonable heuristic in itself for combining multiple sampling strategies.

References

[1] James Arvo. *Analytic Methods for Simulated Light Transport.* PhD thesis, Yale University, December 1995.

[2] James Arvo. Applications of irradiance tensors to the simulation of non-Lambertian phenomena. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 335–342, August 1995.

[3] James Arvo. Stratified sampling of spherical triangles. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 437–438, August 1995.

[4] Marcel Berger. *Geometry*, volume II. Springer-Verlag, New York, 1987. Translated by M. Cole and S. Levy.

[5] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, 1986.

[6] M. H. Kalos and Paula A. Whitlock. *Monte Carlo Methods*, volume I, *Basics*. John Wiley & Sons, New York, 1986.

[7] David Kirk and James Arvo. Unbiased sampling techniques for image synthesis. *Computer Graphics*, 25(4):153–156, July 1991.

[8] David Kirk and James Arvo. Unbiased variance reduction for global illumination. In *Proceedings of the Second Eurographics Workshop on Rendering*, Barcelona, May 1991.

[9] R. Y. Rubinstein. *Simulation and the Monte Carlo Method*. John Wiley & Sons, New York, 1981.

[10] Peter Shirley, Changyaw Wang, and Kurt Zimmerman. Monte Carlo methods for direct lighting calculations. *ACM Transactions on Graphics*, 15(1):1–36, January 1996.

[11] Jerome Spanier and Ely M. Gelbard. *Monte Carlo Principles and Neutron Transport Problems*. Addison-Wesley, Reading, Massachusetts, 1969.

[12] Greg Turk. Generating random points in triangles. In Andrew S. Glassner, editor, *Graphics Gems*, pages 24–28. Academic Press, New York, 1990.

[13] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for Monte Carlo rendering. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 419–428, August 1995.

Direct Illumination

SIGGRAPH 2004 Course

Kavita Bala
Cornell University

Overview

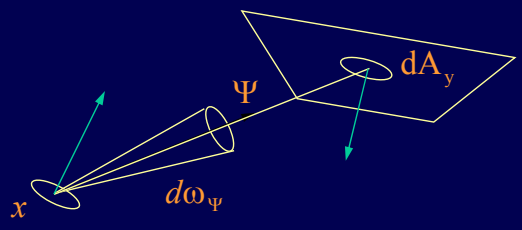
- Reformulation of RE for direct illumination
- Sampling luminaires
- Sampling for many lights

In this lecture, we are going to focus on solving the direct illumination component of the rendering equation.

We will first reformulate the rendering equation so that the direct illumination component is specified over the area of luminaires (rather than the hemisphere at a point). We will then discuss how to sample luminaires. And finally we will discuss how sampling of many lights should be done.

RE for Direct Illumination

$$L_{direct}(x \rightarrow \Theta) = \int_{\Omega_x} f_r(\Psi \leftrightarrow \Theta) \cdot L(x \leftarrow \Psi) \cdot \cos\theta_x \cdot d\omega_\Psi$$



$$y = vp(x, \Psi)$$

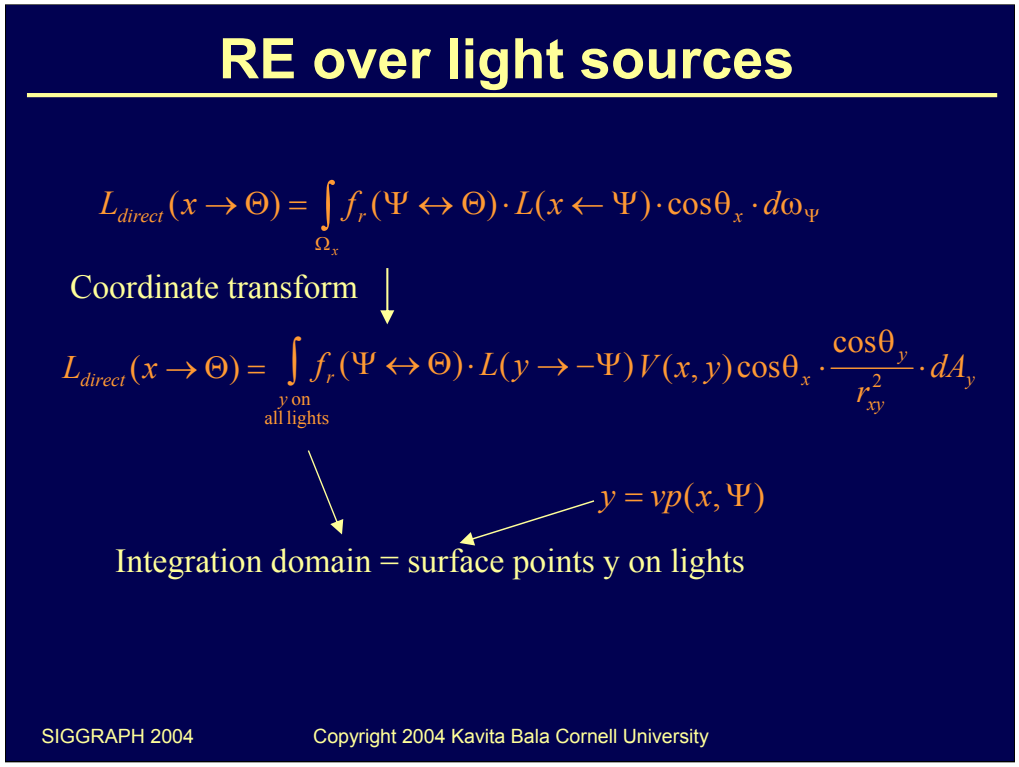
$$L(x \leftarrow \Psi) = L(vp(x, \Psi) \rightarrow -\Psi)$$

$$d\omega_\Psi = \frac{dA_y \cos\theta_y}{r_{xy}^2}$$

The direct illumination component of the rendering equation specified over the hemisphere is given above.

We introduce a function vp that determines the visible point y from a point x in a direction \psi.

Also, the solid angle subtended by a light can be converted into an area formulation over the light.

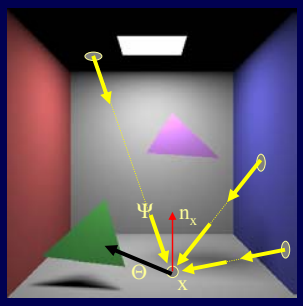


Substituting these transformations into the original equation we arrive at a formulation of direct illumination that is defined over the points on light sources.

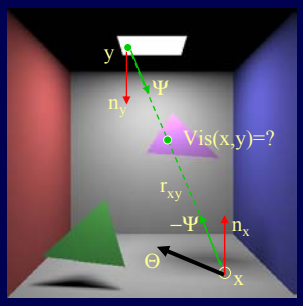
RE over light sources

$$L(x \to \Theta) = \int_{A_{source}} f_r(x, -\Psi \leftrightarrow \Theta) \cdot L(y \to -\Psi) \cdot G(x,y) \cdot dA_y$$

$$G(x,y) = \frac{\cos(n_x, \Theta)\cos(n_y, \Psi) Vis(x,y)}{r_{xy}^2}$$



hemisphere integration



area integration

The light at x because of direct illumination can then be given by the equation above. Note that a geometric term G is introduced to represent both visibility and geometric scaling terms between the light source and the point x.

Sampling over the light's area

- Pick a point on the light's surface
- Therefore, $p(y) = \frac{1}{Area_{source}}$
- For N samples, direct light at point x is:

$$E(x) = \frac{Area_{source} f_r L_{source}}{N} \sum_{i=1}^{N} \frac{\cos\theta_x \cos\theta_{\bar{y}_i}}{r_{x\bar{y}_i}^2} Vis(x, \bar{y}_i)$$

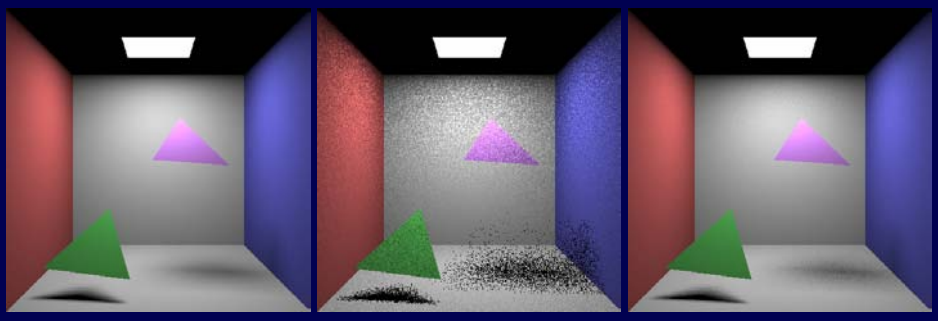
SIGGRAPH 2004

Let us initially assume we have a single light. We want to pick samples uniformly distributed over the area of the light.

The pdf p(y) is inversely proportional to the area of the light source.

Using this sampling technique, N samples are used to compute the estimate of direct lighting as given.

Sampling over area



Reference solution | 1 shadow ray | 100 shadow rays

SIGGRAPH 2004 Copyright 2004 Kavita Bala Cornell University

Using that sampling strategy we get the following results with an increasing number of samples.

How to sample many lights?

- Many lights in scenes: N_L lights
- Various choices:
  - Number of shadow rays
  - Shadow rays per light source
  - Distribution of shadow rays within a light source

Now, let us consider the problem in scenes with many lights.

Various choices exist depending on the following parameters

1. Total number of shadow rays
2. Number of shadow rays to each light
3. Distribution of shadow rays within a light source

We now discuss the issues that arise with these various choices.

Number of shadow rays

- For each evaluation at point x, N_s rays
- With pixel anti-aliasing, k rays through pixel
  - Therefore, kN_S shadow rays per pixel
- Extremes N_S: 1 shadow ray, N_L shadow rays

For each evaluation at point x we can pick the number of shadow rays that must be shot.

The more shadow rays that are shot, the more accurate the solution.

With pixel anti-aliasing, k rays are shot through each pixel. So kN_s shadow rays are required per pixel.

One can imagine with NL lights, that it seems logical to shoot at least NL shadow rays per point.

However, when NL and k are large, this can end up being very expensive.

To avoid this large cost, Ns can be set to a smaller value. One extreme is to shoot only 1 shadow ray for each ray through a pixel. That ray is weighted by NL because it is treated as a “representative” shadow ray for all the NL lights.

How to sample many lights?

- A discrete pdf $p_L(k_i)$ picks the light k_i
- A surface point is then picked with pdf $p(y_i|k_i)$

SIGGRAPH 2004

Once we decide on a certain number of shadow rays, various strategies can distribute samples between lights.

Thus, a discrete pdf is used to pick the light.

Given that the light is picked, a surface point is picked with pdf p(y|k). This can be done uniformly over the light source.

Strategies for picking light

– Uniform $p_L(k) = \frac{1}{N_L}$

– Area $p_L(k) = \frac{A_k}{\sum A_k}$

– Power $p_L(k) = \frac{P_k}{\sum P_k}$

How should the light be picked?

The samples can be distributed uniformly over the lights.

However, consider scenes where the lights are not all equal; I.e., some lights are bigger or brighter. It makes more sense to shoot more rays to the brighter lights, or the bigger lights.

Thus, it should be possible to pick lights based on their area or power.

Example for 2 lights

- Light 0 has power 1, Light 1 has power 2
- Using power weighting:
  - $p_L(L_0) = 1/3$, $p_L(L_1) = 2/3$

SIGGRAPH 2004

We now describe a simple example of two lights to show how these strategies will work. We pick lights based on their power and then uniformly distribute samples over the area of the light selected.

Assume that light0 has power 0, and light 1 has power 2. Using power weighting, light 0 is picked with probability 1/3. Light 1 is picked with probability 2/3.

Example for 2 lights

- Pick a random pair: (ξ_0, ξ_1)
- If $\xi_0 < \frac{1}{3}$
- Sample Light 0 and compute estimate e0
- Overall estimate is $\frac{e_0}{\frac{1}{3}}$

When rendering a point p a pair of random numbers is selected.

If the first random number psi_0 < 1/3, we sample light 0, and compute estimate e0. The estimate for the light is then weighted appropriately since the shadow ray to light 0 is assumed to be “representative”.

Example for 2 lights

- If

$$\frac{1}{3} \leq \xi_0 < 1$$

- Sample Light 1 and compute estimate e1
- Overall estimate is $\frac{e_1}{\frac{2}{3}}$

Similarly, if psi_0 is greater than 1/3 and < 1, we sample light 1 and compute the estimate e1. The overall estimate then is appropriately weighted.

Example for 2 lights

- ξ_0 used to pick light
- Therefore, it cannot be used again directly
- If, $\xi_0 < \frac{1}{3}$
- Use $(3\xi_0, \xi_1)$ to pick samples on light 0
- Similarly, for light 1

When we decide on the light to sample, light 0 or light 1, we must pick a sample on the light. This can be done perhaps by uniformly distributing a sample over the area.

We need two random variables to find the point. We cannot directly reuse psi_0 because we already used some information about psi_0 in the selection of the light. Therefore, we reweight psi_0 (depending on whether it is < 1/3 or not), and then use the reweighted psi_0 to pick samples on lights.

Visibility

- Try to capture visibility during sampling
- Wald et al. use importance sampling
  - Estimates of visibility used to determine pdf for sampling
  - Assumes highly occluded environments

SIGGRAPH 2004

Ideally we would sample each light exactly proportional to the contribution it makes to the final shading a point x. This would require us knowing (ahead of time) what the visibility of the light is.

Recently, work by Wald et al. uses importance sampling to try to capture this visibility term. They construct estimates of visibility for highly occluded environments and construct pdfs that include the visibility information during sampling.

Conclusion

- Sampling lights for direct illumination depends on
  - Number of shadow rays
  - Shadow rays per light source
  - Distribution of shadow rays within a light source

In conclusion, several parameters must be selected carefully when sampling direct illumination. Care must be taken to maintain stratification of samples for lower variance.

Contents

Chapter 1

Metropolis Sampling
by Matt Pharr

A new approach to solving the light transport problem was recently developed by Veach and Guibas, who applied the Metropolis sampling algorithm [**?**, **?**] (first introduced in Section **??** of these notes.)[1]. The Metropolis algorithm generates a series of samples from a non-negative function f that are distributed proportionally to f's value [**?**]. Remarkably, it does this without requiring anything more than the ability to evaluate f; it's not necessary to be able to integrate f, normalize it, and invert the resulting pdf. Metropolis sampling is thus applicable to a wider variety of sampling problems than many other techniques. Veach and Guibas recognized that Metropolis could be applied to the image synthesis problem after it was appropriately reformulated; they used it to develop a general and unbiased Monte Carlo rendering algorithm which they named Metropolis Light Transport (MLT).

MLT is notable for its *robustness*: while it is about as efficient as other unbiased techniques (e.g. bidirectional ray tracing) for relatively straightforward lighting problems, it distinguishes itself in more difficult settings where most of the light transport happens along a small fraction of all of the possible paths through the scene. Such settings were difficult for previous algorithms unless they had specialized advance-knowledge of the light transport paths in the scene (e.g. "a lot of light is coming through that doorway"); they thus suffered from noisy images due to high variance because most of they generated would have a low contribution, but when they randomly sampled an important path, there would be a large spike in the

[1] We will refer to the Monte Carlo sampling algorithm as "the Metropolis algorithm" here. Other commonly-used shorthands for it include M(RT)^2, for the initials of the authors of the original paper, and Metropolis-Hastings, which gives a nod to Hastings, who generalized the technique [**?**]. It is also commonly known as Markov Chain Monte Carlo.

contribution to the image. In contrast, the Metropolis method leads to algorithms that naturally and automatically adapt to the subtleties of the particular transport problem being solved.

The basic idea behind MLT is that a sequence of light-carrying paths through the scene is computed, with each path generated by mutating the previous path in some manner. These mutations are done in a way that ensures that the overall distribution of sampled paths in the scene is proportional to the contribution these paths make to the image being generated. This places relatively few restrictions on the types of mutations that can be applied; in general, it is possible to invent unusual sampling techniques that couldn't be applied to other MC algorithms without introducing bias.

MLT has some important advantages compared to previous unbiased approaches to image synthesis:

- *Path re-use*: because paths are often constructed using some of the segments of the previous one, the incremental cost (i.e. number of rays that need to be traced) for generating a new path is much less than the cost of generating a path from scratch.

- *Local exploration*: when paths that make large contributions to the final image are found, it's easy to sample other paths that are similar to that one by making small perturbations to the path.

The first advantage increases overall efficiency by a relatively fixed amount (and in fact, path re-use can be applied to some other light transport algorithms.) The second advantage is the more crucial one: once an important transport path has been found, paths that are similar to it (which are likely to be important) can be sampled. When a function has a small value over most of its domain and a large contribution in only a small subset of it, local exploration amortizes the expense (in samples) of the search for the important region by letting us stay in that area for a while.

In this chapter, we will introduce the Metropolis sampling algorithm and the key ideas that it is built on. We will then show how it can be used for some low-dimensional sampling problems; this setting allows us to introduce some of the important issues related to the full Metropolis Light Transport algorithm without getting into all of the tricky details. We first show its use in one-dimension. We then demonstrate how it can be used to compute images of motion-blurred objects; this pushes up the domain of the problem to three dimensions and also provides a simpler setting in which to lay more groundwork. Finally, we will build on this basis to make connections with and describe the complete MLT algorithm. We will not attempt to describe every detail of MLT here; however the full-blown

ξ	Uniform random number between 0 and 1
$f(x)$	Function being sampled
Ω	State space over which f is defined
x	A sample value, $x \in \Omega$
x'	A proposed new sample value, based on some mutation strategy
x_i	The ith sample in a Markov chain $x_0, x_1, \ldots, x_n$ generated by the Metropolis sampling algorithm
$p(x)$	A probability density function
$\mathbf{I}(f)$	The integrated value of $f(x)$ over all of Ω, $\mathbf{I}(f) = \int_\Omega f(x)\mathrm{d}\Omega$
f_{pdf}	Normalized probability density of f's distribution, $f_{\mathrm{pdf}} = f/\mathbf{I}(f)$
$\hat{f}(x)$	Reconstructed function that approximates f_{pdf}
$T(x \to x')$	Density of proposed transition from one state x to another x'
$a(x \to x')$	Acceptance probability of mutating from one state x to another x'
$h_j(u, v)$	Value of the jth pixel's reconstruction filter for an image sample at (u, v)
I_j	Image function value at pixel j

Figure 1.1: Notation used in this chapter

presentation in MLT paper [**?**] and the MLT chapter in Veach's thesis [**?**] should be much more approachable with this groundwork.

1.1 Overview

The Metropolis algorithm generates a set of samples x_i from a function f, which is defined over a state space Ω, $f : \Omega \to \mathbb{R}$. (See Figure 1.1 for notation used in this chapter.) After the first sample x_0 is selected (more details on this later), each subsequent sample x_i is generated by proposing a random *mutation* to x_{i-1} to compute a proposed sample x'. The mutation may be accepted or rejected, and x_i is set to either x' or x_{i-1}, respectively. When these transitions from one state to another are chosen subject to a few limitations (described shortly), the distribution of x_i values that results eventually reaches equilibrium; this distribution is the *stationary distribution*.

The way in which mutations are accepted or rejected guarantees that in the limit, the distribution of the set of samples $x_i \in \Omega$ is proportional to $f(x)$'s density function. Thus, even if we are unable to integrate f analytically, normalize it, and invert the integral so that we can sample from it, the Metropolis algorithm still generates samples from f's normalized density function f_{pdf}.

1.1.1 Detailed Balance

In order to generate this particular distribution of samples, we need to accept or reject mutations in a particular way. Assume that we have a method of proposing mutations that makes a given state x into a proposed state x' (this might be done by perturbing x in some way, or even by generating a completely new value.) We also need to be able to compute a tentative transition function $T(x \to x')$ that gives the probability density of the mutation technique's proposing a transition to x', given that the current state is x. (In general, the need to be able to compute this density is the only limitation on how we might choose to mutate—there's a lot of freedom left over!) We also define an *acceptance probability* $a(x \to x')$ that gives the probability of accepting a proposed mutation from x to x'.

The key to the Metropolis algorithm is the definition of $a(x \to x')$ such that the distribution of samples is proportional to $f(x)$. If the random walk is already in equlibrium, the transition density between any two states must be equal:[2]

$$f(x)\,T(x \to x')\,a(x \to x') = f(x')\,T(x' \to x)\,a(x' \to x). \tag{1.1}$$

This property is called *detailed balance*. Since f and T are set, Equation 1.1 tells us how a must be defined. In particular, a definition of a that maximizes the rate at which equilibrium is reached is

$$a(x \to x') = \min\left(1, \frac{f(x')\,T(x' \to x)}{f(x)\,T(x \to x')}\right) \tag{1.2}$$

One thing to notice from Equation 1.2 is that if the transition probability density is the same in both directions, the acceptance probability simplifies to

$$a(x \to x') = \min\left(1, \frac{f(x')}{f(x)}\right) \tag{1.3}$$

For some of the basic mutations that we'll use, this condition on T will be met, which simplifies the implementation.

Put together, this gives us the basic Metropolis sampling algorithm shown in pseudo-code in Figure 1.2. We can apply the algorithm to estimating integrals such as $\int f(x)g(x)\mathrm{d}\Omega$. The standard Monte Carlo estimator, Equation **??**, says that

$$\int_\Omega f(x)g(x)\,\mathrm{d}\Omega \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)g(x_i)}{p(x_i)} \tag{1.4}$$

[2] See Kalos and Whitlock [**?**] or Veach's thesis [**?**] for a rigorous derivation.

```
x = x0
for i = 1 to n
    x′ = mutate(x)
    a = accept(x, x′)
    if (random() < a)
        x = x′
    record(x)
```

Figure 1.2: Pseudocode for the basic Metropolis sampling algorithm. We generate n samples by mutating the previous sample and computing acceptance probabilities as in Equation 1.2. Each sample x_i is then recorded in a data structure.

where x_i are sampled from a density function $p(x)$. Thus, if we apply Metropolis sampling and generate a set of samples $x_1, \ldots, x_N$, from a density function $f_{\mathrm{pdf}}(x)$ proportional to $f(x)$, we have

$$\int_\Omega f(x)g(x)\,\mathrm{d}\Omega \approx \left[\frac{1}{N}\sum_{i=1}^{N} g(x_i)\right] \cdot \mathbf{I}(f) \tag{1.5}$$

where $\mathbf{I}(f)$ is the value of $f(x)$ integrated over all of Ω.

1.1.2 Expected Values

Because the Metropolis algorithm naturally avoids parts of Ω where $f(x)$'s value is relatively low, few samples will be accumulated there. In order to get some information about $f(x)$'s behavior in such regions, the *expected values* technique can be used to enhance the basic Metropolis algorithm.

At each mutation step, we record a sample at both the current sample x and the proposed sample x', regardless of which one is selected by the acceptance criteria. Each of these recorded samples has a weight associated with it, where the weights are the probabilities $(1-a)$ for x and a for x', where a is the acceptance probability. Comparing the pseudocode in Figures 1.2 and 1.3, we can see that in the limit, the same weight distribution will be accumulated for x and x'. Expected values more quickly gives us more information about the areas where $f(x)$ is low, however.

Expected values doesn't change the way we decide which state, x or x' to use at the next step; that part of the computation remains the same.

```
x = x0
for i = 1 to n
    x' = mutate(x)
    a = accept(x, x')
    record(x, (1-a) * weight)
    record(x', a * weight)
    if (random() < a)
        x = x'
```

Figure 1.3: The basic Metropolis algorithm can be improved using *expected values*. We still decide which state to transition into as before, but we record a sample at each of x and x', proportional to the acceptance probability. This gives smoother results, particularly in areas where f's value is small, where otherwise few samples would be recorded.

1.2 One Dimensional Setting

Consider using Metropolis to sample the following one-dimensional function, defined over $\Omega = [0, 1]$ and zero everywhere else (see Figure 1.4).

$$f^1(x) = \begin{cases} (x - .5)^2 & : \quad 0 \leq x \leq 1 \\ 0 & : \quad \text{otherwise} \end{cases} \tag{1.6}$$

For this example, assume that we don't actually know the exact form of f^1—it's just a black box that we can evaluate at particular x values. (Clearly, if we knew that f^1 was just Equation 1.6, there'd be no need for Monte Carlo sampling!)

We'd like to generate a set of samples of f^1 using Metropolis sampling. These samples will be used to reconstruct a new function $\hat{f}^1$ that approximates f^1's pdf. A random choice between the strategies will be made each time a mutation is proposed, according to a desired distribution of how frequently each is to be used.

1.2.1 Mutations and Transition Functions

We will first describe two basic mutation strategies, each of which depends on a uniform random number ξ between zero and one. Our first mutation, mutate_1, discards the current sample x and uniformly samples a new one x' from the entire state space $[0, 1]$. Mutations like this one that sample from scratch are important to make sure that we don't get stuck in one part of state space and never sample the rest of it (an example of the problems that ensue when this happens will be shown below.) The transition function for this mutation is straightforward. For mutate_1,

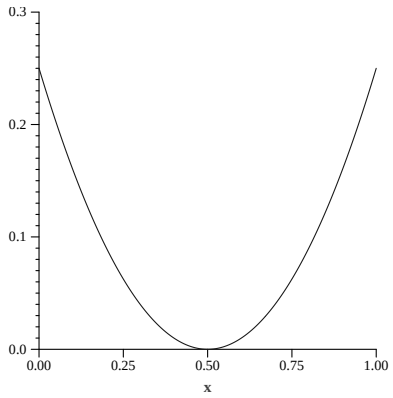


Figure 1.4: Graph of the function f^1, which is used to demonstrate Metropolis sampling in this section.

since we are uniformly sampling over $[0, 1]$, the probability density is uniform over the entire domain; in this case, the density is just one everywhere. We have

$$\begin{aligned} \mathrm{mutate}_1(x) &\rightarrow \xi \\ T_1(x \rightarrow x') &= 1 \end{aligned}$$

The second mutation adds a random offset between $\pm.05$ to the current sample x in an effort to sample repeatedly in the parts of f that make a high contribution to the overall distribution. The transition probability density is zero if x and x' are far enough away that mutate_2 will never mutate from one to the other; otherwise the density is constant. Normalizing the density so that it integrates to one over its domain gives the value $\frac{1}{0.1}$.

$$\begin{aligned} \mathrm{mutate}_2(x) &\rightarrow x + .1 * (\xi - .5) \\ T_2(x \rightarrow x') &= \begin{cases} \frac{1}{0.1} & : \quad |x - x'| \leq .05 \\ 0 & : \quad \mathrm{otherwise} \end{cases} \end{aligned}$$

The second mutation is important for overall efficiency: when we find a sample x_i where $f^1(x_i)$ is larger than most other values of $f^1(x)$, we would like to examine samples close to x_i in the state space, since $f(x)$ is also likely to be large there. Furthermore, this mutation has the important property that it makes us more likely to accept proposed mutations: if we only used the first mutation strategy, we would find it difficult to mutate away from a sample x_i where $f^1(x_i)$ had a relatively large

value; proposed transitions to other states where $f^1(x') \ll f(x_i)$ are rejected with high probability, so many samples in a row would be accumulated at x_i. (Recall the definition of the acceptance probability $a(x \to x')$ in Equation 1.3.) Staying in the same state for many samples in a row leads to increased variance—intuitively, it makes sense that the more we move around Ω, the better the overall results will be. Adding the second mutation to the mix increases the probability of being able to accept mutations around such samples where $f^1(x)$ is relatively high, giving a better distribution of samples in the end.

1.2.2 Start-up bias

Before we can go ahead and use the Metropolis algorithm, one other issue must be addressed: *start-up bias*. The transition and acceptance methods above tell us how to generate new samples x_{i+1}, but all presuppose that the current sample x_i has itself *already* been sampled with probability proportional to f. A commonly used solution to this problem is to run the Metropolis sampling algorithm for some number of iterations from an arbitrary starting state, discard the samples that are generated, and then start the process for real, assuming that that has brought us to an appropriately sampled x value. This is unsatisfying for two reasons: first, the expense of taking the samples that were then discarded may be high, and second, we can only guess at how many initial samples must be taken in order to remove start-up bias.

Veach proposes another approach which is unbiased and straightforward. If an alternative sampling method is available, we sample an initial value x_0 using any density function $x_0 \sim p(x)$. We start the Markov chain from the state x_0, but we weight the contributions of all of the samples that we generate by the weight

$$w = \frac{f(x_0)}{p(x_0)}.$$

This method eliminates start-up bias completely and does so in a predictable manner.

The only potential problem comes if $f(x_0) = 0$ for the x_0 we chose; in this case, all samples will have a weight of zero, leading to a rather boring result. This doesn't mean that the algorithm is biased, however; the expected value of the result still converges to the correct distribution (see [**?**] for further discussion and for a proof of the correctness of this technique.)

To reduce variance from this step, we can instead sample a set of N candidate sample values, $y_1, \ldots, y_N$, defining a weight for each by

$$w_i = \frac{f(y_i)}{p(y_i)}. \tag{1.7}$$

We then choose the starting x_0 sample for the Metropolis algorithm from the y_i with probability proportional to their relative weights and compute a sample weight w as the average of all of the w_i weights. All subsequent samples x_i that are generated by the Metropolis algorithm are then weighted by the sample weight w.

For our particular f^1 example above, we only need to take a single sample with a uniform pdf over Ω, since $f^1(x) > 0$ except for a single point in Ω which there is zero probability of sampling.

$$x_0 = \xi$$

The sample weight w is then just $f^1(x_0)$.

1.2.3 Initial Results

We can now run the Metropolis algorithm and generate samples x_i of f^1. At each transition, we have two weighted samples to record (recall Figure 1.3.) A simple approach for reconstructing the approximation to f^1's probability distribution $\hat{f^1}$ is just to store sums of the weights in a set of buckets of uniform width; each sample falls in a single bucket and contributes to it. Figure 1.5 shows some results. For both graphs, we followed a chain of 10,000 mutations, storing the sample weights in fifty buckets over $[0, 1]$. The weighting method for eliminating start-up bias was used.

On the left graph, we used only mutate_1 when a new x' value is to be proposed. This alone isn't a very useful mutation, since it doesn't let us take advantage of the times when we find ourselves in a region of Ω where f has a relatively large value and generate many samples in that neighborhood. However, the graph does suggest that the algorithm is converging to the correct distribution.

On the right, we randomly chose between mutate_1 and mutate_2 with probabilities of 10% and 90%, respectively. We see that for the same number of samples taken, we converge to f's distribution with less variance. This is because we are more effectively able to concentrate our work in areas where f's value is large, and propose fewer mutations to parts of state space where f's value is low. For example, if $x = .8$ and the second mutation proposes $x' = .75$, this will be accepted $f(.75)/f(.8) \approx 69\%$ of the time, while mutations from .75 to .8 will be accepted $\min(1, 1.44) = 100\%$ of the time. Thus, we see how the algorithm naturally tends to try to avoid spending time sampling around dip in the middle of the curve.

One important thing to note about these graphs is that the y axis has units that are different than those in Figure 1.4, where f^1 is graphed. Recall that we just have a set of samples distributed according to the probability density f^1_{pdf}; as such (for example), we would get the same sample distribution for another function $g = 2f^1$. If we wish to reconstruct an approximation to f^1 directly, we must

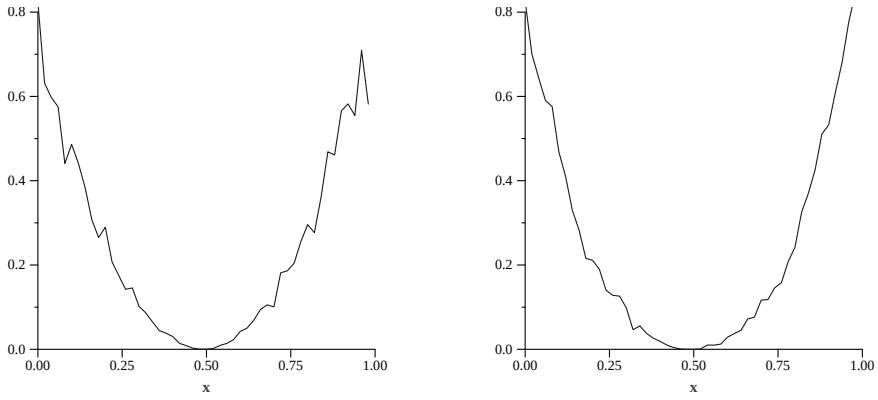


Figure 1.5: On the left, we always mutate by randomly selecting a completely new x value. Convergence is slow, but the algorithm is finding the right distribution. On the right, we perturb the current sample by $\pm.05$ 90% of the time, and pick a completely new x value the remaining 10%.

compute a normalization factor and use it to scale f^1_{pdf}. We explain this process in more detail in Section 1.3.2.

1.2.4 Ergodicity

Figure 1.6 shows the surprising result of what happens if we only use mutate_2 to suggest sample values. On the left, we have taken 10,000 samples using just that mutation. Clearly, things have gone awry—we didn't generate *any* samples $x_i > .5$ and the result doesn't bear much resemblance to f^1.

Thinking about the acceptance probability again, we can see that it would take a large number of mutations, each with low probability of acceptance, to move x_i down close enough to $.5$ such that mutate_2's short span would be enough to get us to the other side. Since the Metropolis algorithm tends to keep us away from the lower-valued regions of f (recall the comparison of probabilities for moving from $.8$ to $.75$, versus moving from $.75$ to $.8$), this happens quite rarely. The right side of Figure 1.6 shows what happens if we take 300,000 samples. This was enough to make us jump from one side of $.5$ to the other a few times, but not enough to get us close to the correct distribution.

This problem is an example of a more general issue that must be addressed with Metropolis sampling: it's necessary that it be possible to reach all states $x \in \Omega$ where $f(x) > 0$ with non-zero probability. In particular, it suffices that $T(x \to x') > 0$ for all x and x' where $f(x) > 0$ and $f(x') > 0$. Although the first condition is in fact met when we use only mutate_2, many samples would be

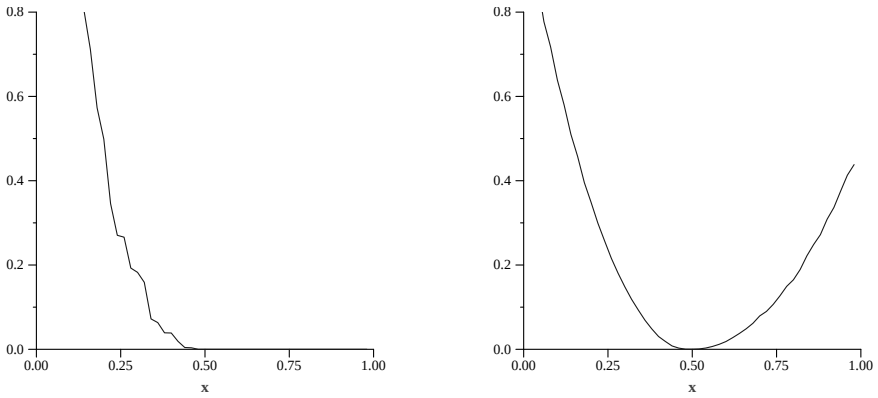


Figure 1.6: Two examples that show why it is important to periodically pick a completely new sample value. On the left, we ran 10,000 iterations using only mutate_2, and on the right, 300,000 iterations. It is very unlikely that a series of mutations will be able to move from one side of the curve, across 0.5, to the other side, since mutations to areas where f^1's value is low will usually be rejected. As such, the results are inaccurate for these numbers of iterations. (It's small solace that they would be correct in the limit.)

necessary in practice to converge to an accurate distribution. Periodically using mutate_1 ensures sufficiently better coverage of Ω such that that this problem goes away.

1.2.5 Mutations via PDFs

If we have a pdf that is similar to some component of f, then we can use that to derive one of our mutation strategies as well. Note that if we had a pdf that was exactly proportional to f, all this Metropolis sampling wouldn't be necessary, but lacking that we often can still find pdfs that approximate some part of the function being sampled. Adding such an extra mutation strategy to the mix can improve overall robustness of the Metropolis algorithm, by ensuring good coverage of important parts of state space.

If we can generate random samples from a probability density function p, $x \sim p$, the transition function is straightforward:

$$T(x \to x') = p(x').$$

i.e. the current state x doesn't matter for computing the transition density: we propose a transition into a state x' with a density that depends only on the newly proposed state x' and not at all on the current state.

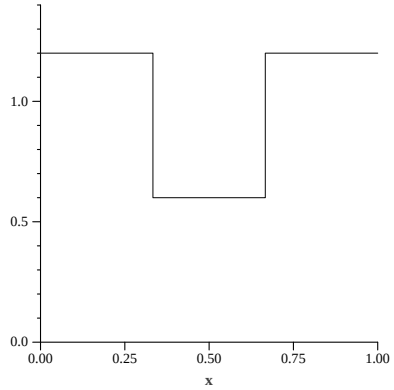


Figure 1.7: Linear pdf used to sample f^1. We can also develop mutation strategies based on pdfs that approximate some component of the function that we're sampling. Here, we're using a simple linear function that is roughly similar to the shape of f^1.

To apply this to the one-dimensional f^1 example, we add a mutation based on a linear probability density function p_1 that is somewhat similar to the shape of f^1. (see Figure 1.7).

$$p_1(x) = \begin{cases} 1.2 & : & x \leq 1/3 \\ .6 & : & 1/3 < x \leq 2/3 \\ 1.2 & : & 2/3 < x \end{cases}$$

Note that $p_1(x)$ is a valid probability density function; it integrates to one over the domain $[0, 1]$. We can apply the function inversion technique described in Section **??** to derive a sampling method. Inverting the integral $\xi = \int_0^x p_1(x)\mathrm{d}x$ gives us:

$$x = \begin{cases} \frac{1}{3}\frac{\xi}{.4} & : & \xi \leq .4 \\ \frac{1}{3} + \frac{1}{3}\frac{(\xi - .4)}{.2} & : & .4 < \xi \leq .6 \\ \frac{2}{3} + \frac{1}{3}\frac{(\xi - .6)}{.4} & : & \xi > .6 \end{cases} \tag{1.8}$$

Our third mutation strategy, mutate_3, just generates a uniform random number ξ and proposes a mutation to a new state x' according to Equation 1.8. Results of using this mutation alone are shown in Figure 1.8; the graph is not particularly better than the previous results, but in any case, it is helpful to have a variety of methods with which to develop mutations.

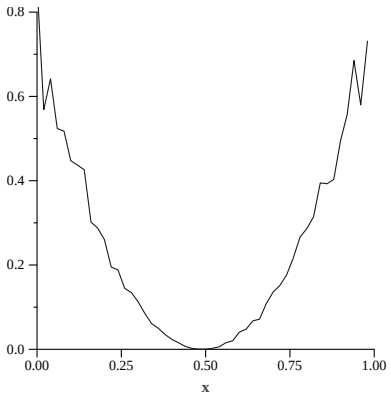


Figure 1.8: Result of reconstructing f^1's pdf using the new mutate_3.

1.3 Motion Blur

We will now show how Metropolis sampling can be used to solve a tricky problem with rendering motion-blurred objects. For objects that are moving at a high speed across the image plane, the standard distribution ray tracing approach can be inefficient; if a pixel is covered by an object for only a short fraction of the overall time, most of the samples taken in that pixel will be black, so that a large number are needed to accurately compute the pixel's color. We will show in this section how Metropolis can be applied to solve this problem more efficiently. In addition to being an interesting application of Metropolis, this also gives us an opportunity to lay further groundwork before moving onto the complete MLT algorithm.

The basic setting of the motion blur problem is that we have a two-dimensional image plane, with some number of pixels, p_u and p_v, in the u and v directions. The time range is from zero to one. We define the scene's radiance function $L(u, v, t)$, which gives the radiance visible along the ray through the scene from the position (u, v) on the image plane, at time t. (L can be computed with standard ray tracing methods.) The state space Ω for this problem is thus the triples $(u, v, t) \in \mathbb{R}^3$, where L's value is zero for (u, v) outside the image and where $t < 0$ or $t > 1$. The measure is just the product measure $\mathrm{d}u\,\mathrm{d}v\,\mathrm{d}t$.

The key step to being able to apply Metropolis sampling to image synthesis problems is the definition of the *image contribution function* [**?**, Chapter 11, Section 3]. For an image with j pixels, each pixel I_j has a value that is the product of the pixel's image reconstruction filter h_j and the radiance L that contributes to the image:

$$I_j = \int_\Omega h_j(u, v)\, L(u, v, t)\, \mathrm{d}u\, \mathrm{d}v\, \mathrm{d}t$$

The standard Monte Carlo estimate of I_j is[3]

$$I_j \approx \frac{1}{N} \sum_{i=1}^{N} \frac{h_j(x_i)\, L(x_i)}{p(x_i)}, \tag{1.9}$$

where p is a probability density function used for sampling $x_i \in \Omega$ and where we have written h_j and L as functions of samples $x_i \in \mathbb{R}^3$ (even though h_j typically only depends on the u and v sample location of x_i.)

In order to be able to apply Metropolis sampling to the problem of estimating pixel values I_j, we can apply Equation 1.5 to rewrite this as

$$I_j \approx \frac{1}{N} \sum_{i=1}^{N} h_j(x_i) \cdot \left(\int_{\Omega} L(x)\, \mathrm{d}\Omega \right), \tag{1.10}$$

since the Metropolis sampling algorithm generates a set of samples x_i according to the distribution that exactly matches L's probability density function.

In the remainder of this section, we will describe an application of Metropolis sampling to the motion blur problem and compare the results to standard approaches. Our example scene is a series of motion-blurred balls, moving across the screen at increasing velocities from top-to-bottom (see Figure 1.9).

1.3.1 Basic Mutation Strategies

We will start with two basic mutation strategies for this problem. First, to ensure ergodicity, we generate a completely new sample 10% of the time. This is just like the one dimensional case of sampling f^1. Here, we choose three random numbers from the range of valid image and time sample values.

Our second mutation is a pixel and time perturbation that helps with local exploration of the space. Each time it is selected, we randomly move the pixel sample location up to ± 8 pixels in each direction (for a 512 by 512 image), and up to $\pm .01$ in time. If we propose a mutation that takes us off of the image or out of the valid time range, the transition is immediately rejected. The performance of the algorithm isn't too sensitive to these values, though see below for the results of some experiments where they were pushed to extremes.

The transition probabilities for each of these mutations are straightforward, analogous to the one dimensional examples.

Figure 1.9 shows some results. The top image was rendered with distribution ray tracing (with a stratified sampling pattern), and the bottom the Metropolis sampling approach with these two mutations was used. The sample total number of

[3] Because the filter support of h_j is usually only a few pixels wide, a small number of samples x_i will contribute to each pixel.

samples was taken for each. Note that Metropolis does equally well regardless of the velocity of the balls, while fast moving objects are difficult for distribution ray tracing to handle well. Because Metropolis sampling can locally explore the path space after it has found a sample that hits one of the balls, it is likely to find other samples that hit them as well, thus being more efficient—the small time perturbation is particularly effective for this. Note, however, that Metropolis doesn't do as well with the ball that is barely moving at all, while this is a relatively easy case for stratified sampling to handle well.

It's interesting to see the effect of varying the parameters to mutate_2. First, we tried greatly increasing the amount we can move in Ω, up to ± 80 pixels in each direction and $\pm .5$ in time. Figure 1.10 (top) shows the result. Because we are no longer doing a good job of local exploration of the space, the image is quite noisy. (One would expect it to degenerate to something like distribution ray tracing, but without the advantages of stratified sampling patterns that are easily applied in that setting.)

We then dialed down the parameters, to $\pm .5$ pixels of motion and $\pm .001$ in time, for a single mutation; see Figure 1.10 (bottom). Here the artifacts in the image are more clumpy—this happens because we find a region of state space with a large contribution but then have trouble leaving it and finding other important regions. As such, we don't do a good job of sampling the entire image.

As a final experiment, we replaced mutate_2 with a mutation that sampled the pixel and time deltas from an exponential distribution, rather than a uniform distribution. Given minimum and maximum pixel offsets $r_{\max}$ and $r_{\min}$ and time offsets $t_{\max}$ and $t_{\min}$, we computed

$$\begin{aligned} r &= r_{\max} \mathrm{e}^{-\log(r_{\max}/r_{\min})\xi} \\ dt &= t_{\max} \mathrm{e}^{-\log(t_{\max}/t_{\min})\xi} \end{aligned}$$

Given these offsets, a new pixel location was computed by uniformly sampling an angle $\theta = 2\pi\xi$, and the new image (u, v) coordinates were computed by

$$(u, v) = (r \sin\theta, r \cos\theta)$$

The new time value was computed by offsetting the old one by $\pm dt$, where addition and subtraction were chosen with equal probability.

We rendered an image using this mutation; the range of pixel offsets was $(.5, 40)$, and the range of time deltas was $(.0001, .03)$. Figure 1.11 shows the result. The improvement is not major, but is noticeable. In particular, see how noise is reduced along the edges of the fast-moving ball.

The advantage of sampling with an exponential distribution like this is that it naturally tries a variety of mutation sizes. It preferentially makes small mutations,

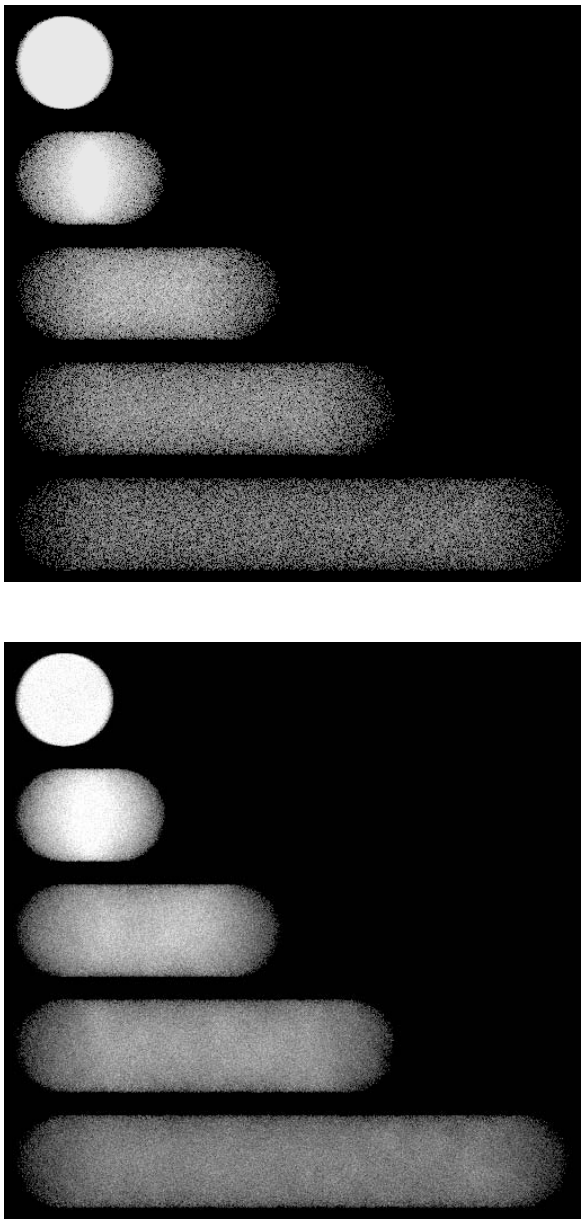


Figure 1.9: Basic motion blur results. On the top, we have applied distribution ray tracing with a stratified sampling pattern, and on the bottom, we have applied Metropolis sampling. The images are 512x512 pixels, with an average of 9 samples per pixel.

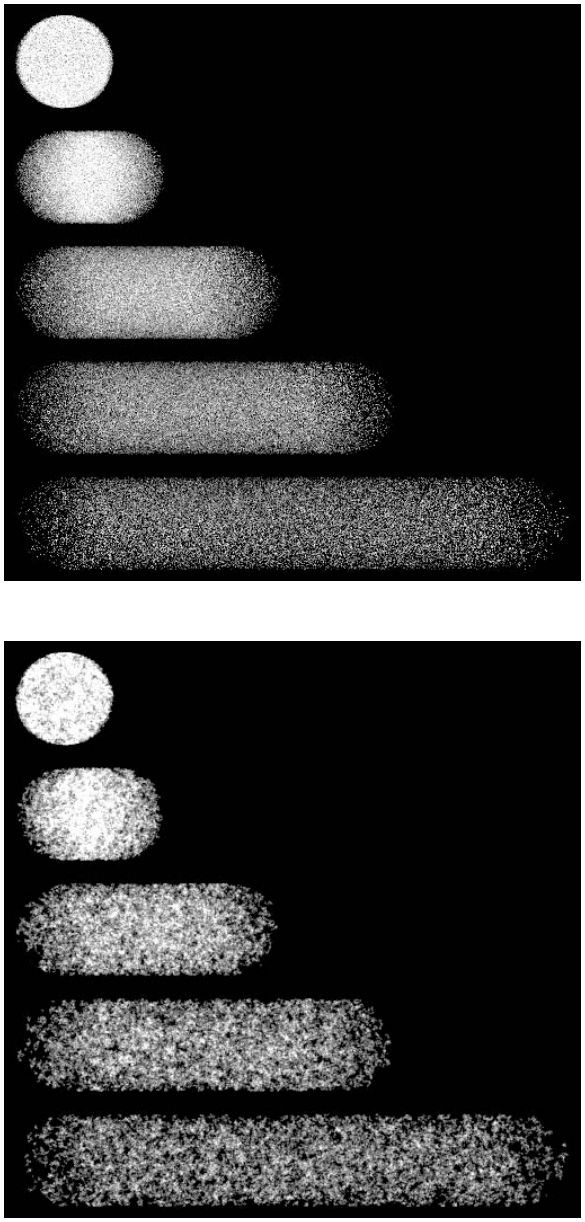


Figure 1.10: The effect of changing the parameters to the second motion blur mutation strategy. Top: if the mutations are too large, we get a noisy image. Bottom: too small a mutation leads to clumpy artifacts, since we're not doing a good job of exploring the entire state space.

close to the minimum magnitudes specified, which help locally explore the path space in small areas of high contribution where large mutations would tend to be rejected. On the other hand, because it also can make larger mutations, it also avoids spending too much time in a small part of path space, in cases where larger mutations have a good likelihood of acceptance.

1.3.2 Re-normalization

Recall that the set of samples generated by the Metropolis algorithm is from the normalized distribution f_{pdf} of the function that we apply it to. When we are computing an image, as in the motion blur example, this means that the image's pixel values need to be rescaled in order to be correct.

This problem can be solved with an additional short pre-processing step. We take a small number of random samples (e.g. 10,000) of the function f and estimate its integral over Ω. After applying Metropolis sampling, we have an image of sample densities. We then scale each pixel by the precomputed total image integral divided by the total number of Metropolis samples taken. It can be shown that the expected value is the original function f:

$$f(x) \approx \frac{1}{N} \sum_{i=1}^{N} \hat{f}(x_i)\, \mathbf{I}(f)$$

1.3.3 Adapting for large variation in f

When the image function I has regions with very large values, Metropolis sampling may not quite do what we want. For example, if the image has a very bright light source directly visible in some pixels, most of the Metropolis samples will naturally be clustered around those pixels. As a result, the other regions of the image will have high variance due to under-sampling. Figure 1.12 (top) shows an example of this problem. The bottom ball has been made a very bright red, 1,000 times brighter than the others; most of the samples concentrate on it, so the other balls are under-sampled.

Veach introduced *two-stage Metropolis* to deal with this problem [**?**]. Two-stage Metropolis attempts to keep the relative error at all pixels the same, rather than trying to just minimize absolute error. We do this by renormalizing the function L to get a new function L':

$$L'(x) = \frac{L(x)}{n(x)}$$

where n is a normalization function that roughly approximates $L(x)$'s magnitude, such that the range of values taken on by $L'(x)$ is much more limited. The

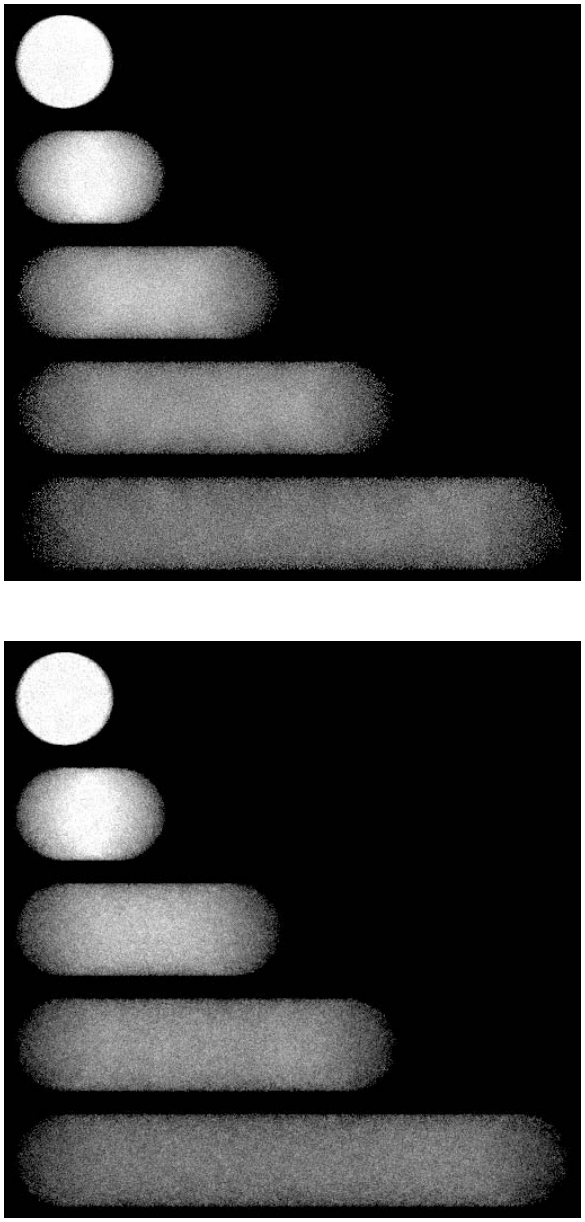


Figure 1.11: Comparison of Metropolis sampling with the first two mutation strategies (top) versus Metropolis sampling where the second strategy is replaced a mutation based on sampling pixel and time offsets from an exponential distribution (bottom). Note how noise is reduced along the edges of the fast-moving ball.

Metropolis algorithm progresses as usual, just evaluating $L'(x)$ where it otherwise would have evaluated $L(x)$. The result is that samples are distributed more uniformly, resulting in a better image. We correct for the normalization when accumulating weights in pixels; by multiplying each weight by $n(x)$, the final image pixels have the correct magnitude.

For our example, we computed a normalization function by computing a low-resolution image (32 by 32 pixels) with distribution ray tracing and then blurring it. We then made sure that all pixels of this image had a non-zero value (we don't want to spend all of our sampling budget in areas where we inadvertently underestimated $n(x)$, such that $L'(x) = L(x)/n(x)$ is large) and so we also set pixels in the normalization image with very low values to a fixed minimum. Applying Metropolis as before, we computed the image on the bottom of Figure 1.12. Here all of the balls have been sampled well, resulting in a visually more appealing result (even though absolute error is higher, due to the red ball being sampled with fewer samples.)

1.3.4 Color

For scenes that aren't just black-and-white, the radiance function $L(u, v, t)$ returns a spectral distribution in some form. This distribution must be converted to a single real value in order to compute acceptance probabilities (Equation 1.2). One option is to compute computing the luminance of the spectrum and use that; the resulting image (which is still generated by storing spectral values) is still correct, and there is the added advantage that the image is sampled according to its perceived visual importance.

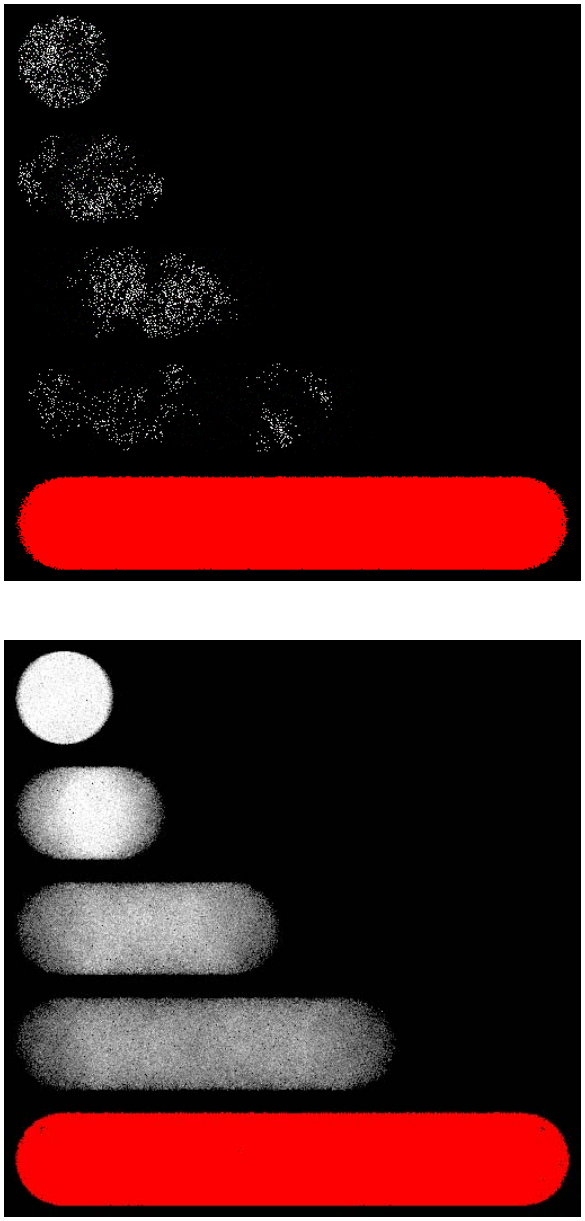


Figure 1.12: Two stage sampling helps when the image has a large variation in pixel brightness. Top: the red ball is much brighter than the others, resulting in too few samples being taken over the rest of the image. Bottom: by renormalizing the image function I, we distribute samples more evenly and generate a less noisy image.

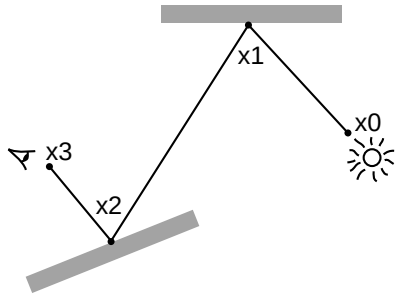


Figure 1.13: A path with three edges through a simple scene. The first vertex v_0 is on a light source, and the last, v_3 is at the eye.

1.4 Metropolis Light Transport

Using the groundwork of the last few sections, the Metropolis Light Transport algorithm can now be described. We will not explain all of its subtleties or all of the details of how to implement it efficiently; see Chapter 11 of Veach's thesis for both of these in great detail [**?**]. Rather, we will try to give a flavor for how it all works and will make connections between MLT and the sampling problems we have described in the previous few sections.

In MLT, the samples x from Ω are now sequences $v_0 v_1 \ldots v_k$, $k \geq 1$, of vertices on scene surfaces. The first vertex, v_0, is on a light source, and the last, v_k is at the eye (see Figure 1.13). This marks a big change from our previous examples: the state space is now an infinite-dimensional space (paths with two vertices, paths with three vertices, ...). As long as there is non-zero probability of sampling any particular path length, however, this doesn't cause any problems theoretically, but it's is another idea that needs to be juggled when studying the MLT algorithm.

As before, the basic strategy is to propose mutations x' to paths x, accepting or rejecting mutations according to the detailed balance condition. The function $f(x)$ represents the differential radiance contribution carried along the path x, and the set of paths sampled will be distributed according to the image contribution function 1.10. (See [**?**] for a more precise definition of $f(x)$.) Expected values are also used as described previously to accumulate contributions at both the current path x's image location, as well as the image location for the proposed path x'.

A set of n starting paths $\bar{x}_i$ are generated with bidirectional path tracing, in a manner that eliminates startup bias. N candidate paths are sampled (recall Section 1.2.2), where $n \ll N$ and we select n of them along with appropriate weights. We start with n separate paths, rather than just one, primarily to be able to improve stratification of samples within pixels (see Section 1.4.2 below as well as [**?**,

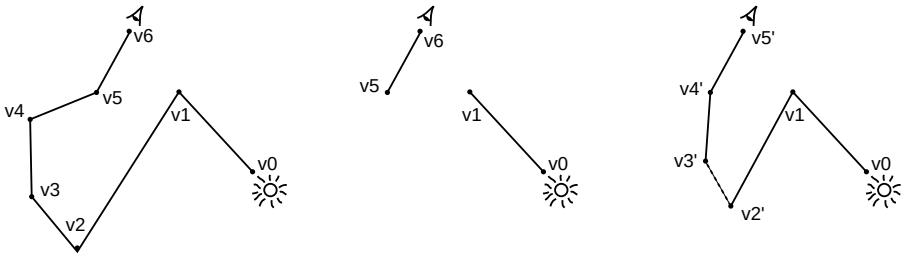


Figure 1.14: Example of a bidirectional mutation to a path. Left: we have a path of length 6 from the eye to the light source. Each vertex of the path represents a scattering event at a surface in the scene. Middle: the bidirectional mutation has decided to remove a sub-path from the middle of the current path; we are left with two short paths from the eye and from the light. Right: we have decided to add one vertex to each of the paths; their locations are computed by sampling the BRDF at the preceding vertices. We then trace a shadow ray between the new vertices v_2' and v_3' to complete the path.

Section 11.3.2].)

1.4.1 Path Mutations

A small set of mutations is applied to the paths in an effort to efficiently explore the space of all possible paths through the scene. The most important of the mutations is the *bidirectional mutation*. The bidirectional mutation is conceptually quite straightforward; a subpath from the middle of the current path is removed, and the two remaining ends are extended with zero or more new vertices (see Figure 1.14). The resulting two paths are then connected with a shadow ray; if it is occluded, the mutation is rejected, while otherwise the standard acceptance computation is performed.

Computation of the acceptance probability for bidirectional mutations requires that we figure out the values of the path contribution function $f(x)$ and $f(x')$ and the pair of $T(x \to x')$ densities. Recall from the introduction of the path integral that that $f(x)$ is a product of emitted importance from the eye, BRDFs values along the path, geometry terms $G(p, p') = \cos\theta_i \cos\theta_o / r^2$, and the differential irradiance from the light source; as such, computation of two the $f(x)$ values can be simplified by ignoring the terms of each of them that are shared between the two paths, since they just cancel out when the acceptance probability is computed.

Computation of the proposed transition densities is more difficult; see [**?**, Section 11.4.2.1] for a full discussion. The basic issue is that it is necessary to consider

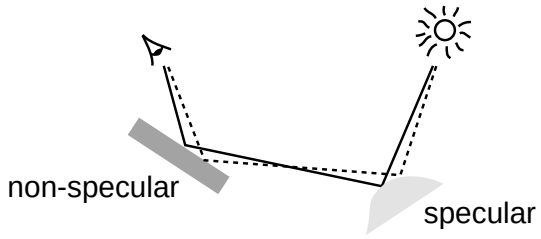


Figure 1.15: The caustic perturbation in action. Given an old path that leaves the light source, hits a specular surface, and then hits a non-specular surface before reaching the eye, we perturb the direction leaving the light source. We then trace rays to generate a new path through the scene and to the eye. (The key here is that because the last surface is non-specular, we aren't required to pick a particular outgoing direction to the eye—we can pick whatever direction is needed.)

all of the possible ways that one *could* have sampled the path you ended up with, given the starting path. (For example, for the path in Figure 1.14, we might have generated the same path by sampling no new vertices from the light source, but two new vertices along the eye path.) This computation is quite similar in spirit to how importance sampling is applied to bidirectional path tracing.

The bidirectional mutation by itself is enough to ensure ergodicity; because there is some probability of throwing away the entire current path, we are guaranteed to not get stuck in some subset of path space.

Bidirectional mutations can be ineffective when a very small part of path space is where the most important light transport is happening—almost all of the proposed mutations will cause it to leave the interesting set of paths (e.g. those causing a caustic to be cast from a small specular object.) This problem can be ameliorated by adding *perturbations* to the mix; these perturbations try to offset some of the vertices of the current path from their current location, while still leaving the path mostly the same (e.g. preserving the mode of scattering—specular or non-specular, reflection or transmission, at each scattering event.)

One such perturbation is the *caustic perturbation* (see Figure 1.15). If the current path hits one or more specular surfaces before hitting a single diffuse surface and then the eye, then it's a caustic path. For such paths, we can make a slight shift to the outgoing direction from the light source and then trace the resulting path through the scene. If it hits all specular surfaces again and if the final diffuse surface hit is visible to the eye, we have a new caustic sample at a different image location. The caustic perturbation thus amortizes the possibly high expense of finding caustic paths.

Lens perturbations are based on a similar idea to caustic perturbations, where

the direction of outgoing ray from the camera is shifted slightly, and then followed through the same set of types of scattering at scene surfaces. This perturbation is particularly nice since it keeps us moving over the image plane, and the more differently-located image samples we have, the better the quality of the final image.

1.4.2 Pixel Stratification

Another problem with using Metropolis to generate images is that random mutations won't do a good job of ensuring that pixel samples are well-stratified over the image plane. In particular, it doesn't even ensure that all of the pixels have *any* samples taken within them. While the resulting image is still unbiased, it may be perceptually less pleasing than an image computed with alternative techniques.

Veach has suggested a *lens subpath mutation* to address this problem. A set of pixel samples that must be taken is generated (e.g. via a Poisson-disk process, a stratified sampling pattern, etc.) As such, each pixel has some number of required sample locations associated with it. The new mutation type first sees if the pixel corresponding to the current sample path has any precomputed sample positions that haven't been used. If so, it mutates to that sample and traces a new path into the scene, following as many specular bounces as are found until a non-specular surface is found. This *lens subpath* is then connected with a path to a light source. If the randomly selected pixel does have its quota of lens subpath mutations already, the other pixels are examined in a pseudo-random ordering until one is found with remaining samples.

By ensuring a minimum set of well-distributed samples that are always taken, overall image quality improves and we do a better job of (for example) anti-aliasing geometric edges than we would to otherwise. Remember that this process just sets a minimum number of samples that are taken per pixel—if the number of samples allocated to ensuring pixel stratification is 10% of the total number of samples (for example), then most of our samples will still be taken in regions with high contribution to the final image.

1.4.3 Direct Lighting

Veach notes that MLT (as described so far) often doesn't do as well with direct lighting as standard methods, such as those described in Chapter **??** of these notes, or in Shirley et al's TOG paper [**?**]. The root of the problem is that the Metropolis samples over the light sources aren't as well stratified as they can be with standard methods.

A relatively straightforward solution can be applied: when a lens subpath is generated (recall that lens subpaths are paths from the eye that follow zero or more

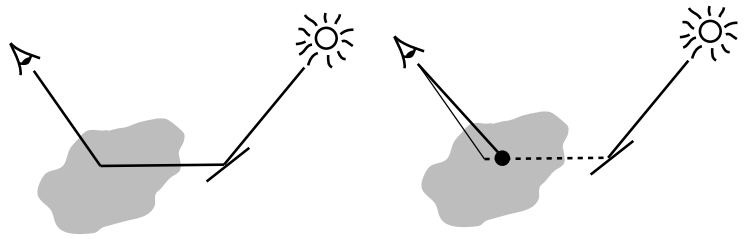


Figure 1.16: For path vertices that are in the scattering volume, rather than at a surface, the scattering propagation perturbation moves a vertex (shown here as a black circle) a random distance along the two path edges that are incident the vertex. Here we have chosen the dashed edge. If the connecting ray to the eye is occluded, the mutation is immediately rejected; otherwise the usual acceptance probability is computed.

specular bounces before hitting a diffuse surface), standard direct lighting techniques are used at the diffuse surface to compute a contribution to the image for the lens subpath. Then, whenever a MLT mutation is proposed that includes direct lighting, it is immediately rejected since direct lighting was already included in the solution.

Veach notes, however, that this optimization may not always be more effective. For example, if the direct lighting cannot be sampled efficiently by standard techniques (e.g. due to complex visibility, most of the light sources being completely occluded, etc.), then MLT would probably be more effective.

1.4.4 Participating Media

Pauly et al have described an extension of MLT to the case of participating media [**?**]. The state space and path measure are extended to include points in the volume in addition to points on the surfaces. Each path vertex may be on a surface or at a point in the scattering volume. The algorithm proceeds largely the same way as standard MLT, but places some path vertices on scene surfaces and others at points in the volume. As such, it robustly samples the space of all contributing transport paths through the medium.

They also describe a new type of mutation, tailored toward sampling scattering in participating media—the *propagation perturbation*. This perturbation randomly offsets a path vertex along one of the two incident edges (see Figure 1.16). Like other perturbations, this mutation helps concentrate work in important regions of the path space.

Acknowledgements

Eric Veach was kind enough to read these notes through a series of drafts and offer extensive comments and many helpful suggestions for clearer presentation.

Chapter 2

Implementating the Metropolis Light Transport Algorithm

by Matt Pharr

These notes were not complete in time for the SIGGRAPH course notes deadline. They can be downloaded from:

`http://graphics.stanford.edu/~mmp/mlt-implementation.pdf`

Biased Techniques

By Henrik Wann Jensen

In the previous chapters we have seen examples of several *unbiased* Monte Carlo ray tracing (MCRT) techniques. These techniques use pure Monte Carlo sampling to compute the various estimates of radiance, irradiance etc. The only problem with pure MCRT is variance — seen as noise in the rendered images. The only way to eliminate this noise (and still have an unbiased algorithm) is to sample more efficiently and/or to use more samples.

In this chapter we will discuss several approaches for removing noise by introducing bias. We will discuss techniques that uses interpolation of irradiance to exploit the smoothness of the irradiance field. This approach makes it possible to use more samples at selected locations in the model. We will also discuss photon mapping, which stores information about the flux in a scene and performs a local evaluation of the statistics of the stored flux in order to speedup the simulation of global illumination.

Biased vs. Unbiased

An *unbiased* Monte Carlo technique does not have any systematic error. It can be stopped after any number of samples and the expected value of the estimator will be the correct value. This does not mean that all biased methods give the wrong result. A method can converge to the correct result as more samples are used and still be biased, such methods are *consistent*.

The integral of a function $g(x)$ can be expressed as the expected value of an estimator Ψ where Ψ is:

$$\Psi = \frac{1}{N} \sum_{i=1}^{N} \frac{g(x)}{p(x)} \ . \tag{1}$$

where $p(x)$ is a p.d.f. distributed according to x such that $p(x) > 0$ when $g(x) > 0$. The expected value of Ψ is the value of the integral:

$$E\{\Psi\} = I = \int_{x \in S} g(x)\, d\mu \,. \tag{2}$$

Since the expected value of the Ψ is our integral Ψ is an unbiased estimate of I.

In contrast a biased estimator Ψ^* have some other source of error such that:

$$E\{\Psi^*\} = I + \epsilon \,, \tag{3}$$

where ϵ is some error term. For a consistent estimator the error term will diminish as the number of samples is increased:

$$\lim_{N \to \infty} \epsilon = 0 \,. \tag{4}$$

Why should we be interested in anything, but unbiased methods? To answer this problem recall the slow convergence of pure Monte Carlo methods. To render images without noise it can be necessary to use a very high number of samples. In addition the eye is very sensitive to the high frequency noise that is typical with unbiased MCRT methods.

Going into the domain of biased methods we give up the ability to classify the error on the estimate by looking only at the variance. However the variance only gives us a probability the error is within a certain range and as such it is not a precise way of controlling the error. In addition the requirements for unbiased algorithms are quite restrictive; we cannot easily use adaptive sampling or stop the sampling once the estimate looks good enough — such simple decisions lead to biased methods [8].

With biased methods other sources of error are introduced to reduce the variance. This can lead to artifacts if the error is uncontrollable, so naturally we want a consistent method that will converge to the correct result as we use more samples. It is important to have the right balance between bias and variance. We want to eliminate the noise, but not introduce other strange artifacts. As we will see in the following the most common way of reducing noise is to blur the estimates; the eye is fairly insensitive to slowly changing illumination. The trick is to avoid blurring edges and sharp features in the image and to have control over the amount of blur.

Consider the simple box scene in Figure 1. The figure contains two images of the box scene: one in which the box contains two diffuse spheres, and one

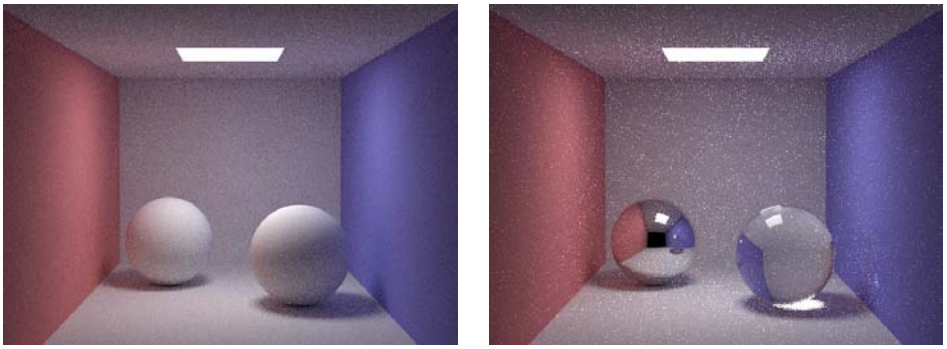


Figure 1: A path traced box scene with 10 paths/pixel. On the left the box has two diffuse spheres, and on the right box box has a mirror and a glass sphere. Note how the simple change of the sphere material causes a significant increase in the noise in the right image.

in which the box contains a glass and a mirror sphere. Both images have been rendered with 10 paths per pixel. Even though the illumination of the walls is almost the same in the two images the introduction of the specular spheres is enough to make the path tracing image very noisy. In the following sections we will look at a number of techniques for eliminating this noise without using more samples.

Filtering Techniques

An obvious idea for reducing the noise in a rendered image is to postprocess the image and try to filter out the high frequency noise. This can be done to some degree with a low pass filter or with a median filter [3, 9]. The problem with these filters is that they remove other features than just the noise. Low pass filtering in particular will blur sharp edges of objects shadows etc. and in general low-pass filtered images look too blurry. Median filtering is much better at removing noise, but it still introduces artifacts along edges and other places since it cannot distinguish between noisy pixels and features of the illumination (such as small highlights). In addition median filtering is not *energy preserving*. By simply removing outliers in the image plane it is very likely that we take away energy or add energy to the rendered image. The effect of low-pass and median filtering on the box scene is shown in Figure 2

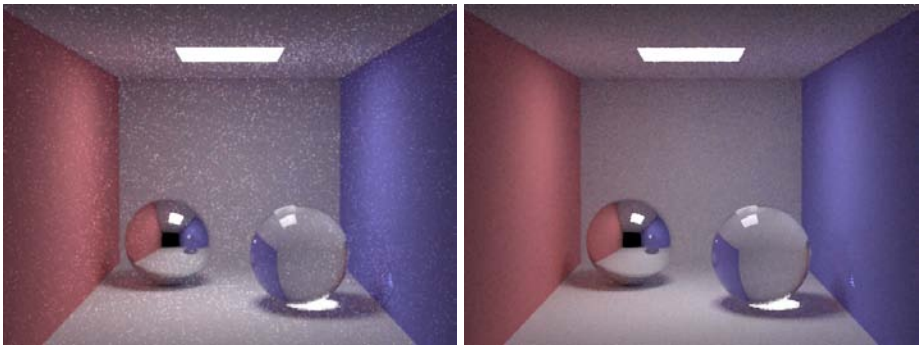


Figure 2: Filtered versions of the image of the box with specular spheres. On the left is the result of a 3x3 low-pass filter, and on the right the result of a 3x3 median filter.

Several other more sophisticated filtering techniques have been used.

Jensen and Christensen [6] applied a median filter to the indirect illumination on diffuse surfaces based on the assumption that most of the noise in path tracing is found in this irradiance estimate. By filtering only this estimate before using it they removed a large fraction of the noise without blurring the edges, and features such as highlights and noisy textures. The problem with this approach is that it softens the indirect illumination and therefore blurs features due to indirect lighting. Also the technique is not energy-preserving.

Rushmeier and Ward [13] used a energy-preserving non-linear filter to remove noisy pixels by distributing the extra energy in the pixel over several neighboring pixels.

McCool [11] used an anisotropic diffusion filter which also preserves energy and allows for additional input about edges and textures in order to preserve these features.

Suykens and Willems [16] used information gathered during rendering (they used bidirectional path tracing) about the probabilities of various events. This enabled them to predict the occurrence of spikes in the illumination. By distributing the power of the spikes over a larger region in the image plane their method removes noise and preserves energy. The nice thing about this approach is that it allows for progressive rendering with initially blurry results that converges to the correct result as more samples are used.

Adaptive Sampling

Another way of eliminating noise in Monte Carlo ray traced images is to use adaptive sampling. Here the idea is to use more samples for the problematic (i.e. noisy) pixels.

One techniques for doing this is to compute the variance of the estimate based on the samples used for the pixel [10, 12]. Instead of using a fixed number of samples per pixel each pixel is sampled until the variance is below a given threshold. An estimate, s^2, of the variance for a given set of samples can be found using standard techniques from statistics:

$$s^2 = \frac{1}{N-1}\left[\frac{1}{N}\sum_{i=1}^{N} L_i^2 - \left(\frac{1}{N}\sum_{i=1}^{N} L_i\right)^2\right] \tag{5}$$

This estimate is based on the assumption that the samples L_i are distributed according to a normal distribution. This assumption is often reasonable, but it fails when for example the distribution within a pixel is bimodal (this happens when a light source edge passing through a pixel).

Using the variance as a stopping criteria can be effective in reducing noise, but it introduces bias as shown by Kirk and Arvo [8]. They suggested using a pilot sample to estimate the number of samples necessary. To eliminate bias the pilot sample must be thrown away.

The amount of bias introduced by adaptive sampling is, however, usually very small as shown by Tamstorf and Jensen [17]. They used bootstrapping to estimate the bias due to adaptive sampling and found that it is insignificant for most pixels in the rendered image (a notable exception is the edges of light sources).

Irradiance Caching

Irradiance caching is a technique that exploits the fact that the irradiance field often is smooth [19]. Instead of just filtering the estimate the idea is to cache and interpolate the irradiance estimates. This is done by examining the samples of the estimate more carefully to, loosely speaking, compute the expected smoothness of the irradiance around a given sample location. If the irradiance is determined to be sufficiently smooth then the estimate is re-used for this region.

To understand in more detail how this works let us first consider the evaluation of the irradiance, E, at a given location, x, using Monte Carlo ray tracing.

$$E(x) = \frac{\pi}{MN} \sum_{j=1}^{M} \sum_{i=1}^{N} L_{i,j}(\theta_j, \phi_i) \; , \tag{6}$$

where

$$\theta_j = \sin^{-1}\left(\sqrt{\frac{j-\xi_1}{M}}\right) \quad \text{and} \quad \phi_i = 2\pi \frac{i-\xi_2}{N} \; . \tag{7}$$

Here (θ_j, ϕ_i) specify a direction on the hemisphere above x in spherical coordinates. $\xi_1 \in [0,1]$ and $\xi_2 \in [0,1]$ are uniformly distributed random numbers, and M and N specify the subdivision of the hemisphere. $L_{i,j}(\theta_j, \phi_i)$ is evaluated by tracing a ray in the (θ_j, ϕ_i) direction. Note that the formula uses stratification of the hemisphere to obtain a better estimate than pure random sampling.

To estimate the smoothness of the local irradiance on the surface around the sample location Ward et al. [19] looked at the distances to the surfaces intersected by the rays as well as the local changes in the surface normal. This resulted in an estimate of the local relative change, ϵ_i, in irradiance as the surface location is changed away from sample i:

$$\epsilon_i(x, \vec{n}) = \frac{||x_i - x||}{R_0} + \sqrt{1 - 1\vec{n} \cdot \vec{n}_i} \; . \tag{8}$$

Here x_i is the original sample location and x is the new sample location (for which we want to compute the change), R_0 is the harmonic distance to the intersected surfaces, $\vec{n}_i$ is the sample normal, and $\vec{n}$ is the new normal.

Given this estimate of the local variation in irradiance Ward et al. developed a caching method where previously stored samples re-used whenever possible. All samples are stored in an octree-tree — this structure makes it possible to quickly locate previous samples. When a new sample is requested the octree is queried first for previous samples near the new location. For these nearby samples the change in irradiance, ϵ, is computed. If samples with a sufficiently low ϵ is found then these samples are blended using weights inversely proportional to ϵ:

$$E(x, \vec{n}) \approx \frac{\sum\limits_{i, w_i > 1/a} w_i(x, \vec{n}) E_i(x_i)}{\sum\limits_{i, w_i > 1/a} w_i(x, \vec{n})} \; . \tag{9}$$

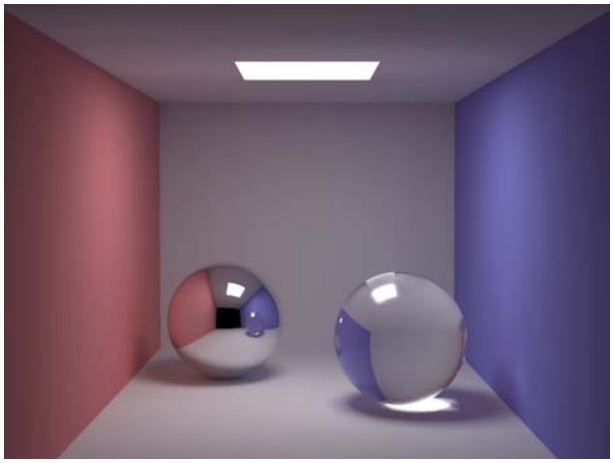


Figure 3: The box scene rendered using irradiance caching.

Here $w_i = \frac{1}{\epsilon_i}$, a is the desired accuracy and E_i is the irradiance of sample i.

If no previously computed sample has a sufficiently high weight then a new sample is computed.

To further improve this estimate Ward and Heckbert added estimates of the gradients of the irradiance [18]. Their approach looks not only at the distances to the nearest surfaces, but also in the relative change of the incoming radiance from different directions in order to more accurately estimate the gradient due to a change in position as well as orientation. The great thing about this approach is that it does not require any further samples, but simply uses a more sophisticated analysis of the samples in the irradiance estimate.

With a few minor modifications this interpolation scheme works quite well as can be seen in Figure 3. The areas where it fails are in the case where the overall smoothness assumption for the irradiance field is no longer true. This is particularly the case for caustics (e.g. the light focused through the glass sphere; the caustic on the wall is missed completely).

Another issue with irradiance caching is that the method can be quite costly in scenes where multiple diffuse light reflections are important, since the costly irradiance estimates are computed recursively for each sample ray. The recursive evaluation also results in a global error (a global bias). Each light bounce has some approximation and multiple light bounces distributes this error all over the scene. This global aspect can make the error hard to control.

Photon Mapping

Photon mapping [4] is a method that uses biasing to reduce variance in many places. This is in part done by aggressively storing and re-using information whenever possible. This section contains a short overview of the photon mapping technique (for all the details consult [5]).

From a light transport point of view photon mapping exploits that:

- The irradiance field is mostly smooth, but has important focusing effects such as caustics
- There are two important sources of light paths in a scene: the light sources and the observer

In addition photon mapping uses a point sampling data-structure that is independent of the scene geometry, and it therefore works with complex geometry as well as non-Lambertian reflection models.

Photon mapping is a two-pass method in which the first pass is building the *photon map*. This is done using *photon tracing* in which photons are emitted from the light sources and traced through the scene. When a photon intersects a diffuse surface it is stored in the photon map. The second pass is rendering in which the photon map is a static structure with information about the illumination in the model. The renderer computes various statistics from the photon map to make the rendering faster.

Pass 1: Photon Tracing

The first step is building the photon map. This is done by emitting photons from the light sources and tracing them through the scene using photon tracing. The photons are emitted according to the power distribution of the light source. As an example, a diffuse point light emits photons with equal probability in all directions. When a photon intersects a surface it is stored in the photon map (if the surface material has a diffuse component). In addition the photon is either scattered or absorbed based on the albedo of the material. For this purpose Russian roulette [1] is used to decide if a photon path is terminated (i.e. the photon is absorbed), or if the photon is scattered (reflected or transmitted). This is shown in Figure 4.

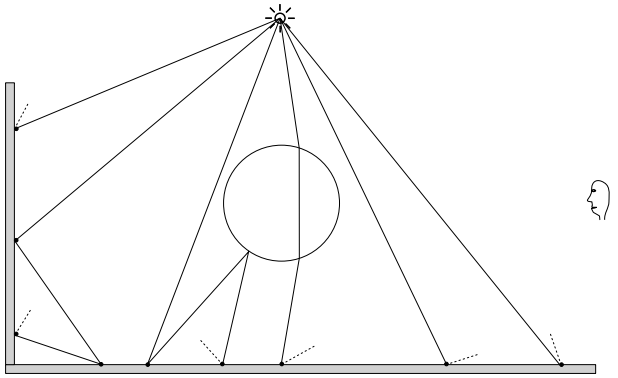


Figure 4: The photon map is build using photon tracing. From [5].

The photon tracing algorithm is unbiased. No approximations or sources of systematic error is introduced in the photon tracing step. In contrast many other light propagation algorithms, such as progressive refinement radiosity [2], introduces a systematic error at each light bounce (due to the approximate representation of the illumination).

For efficiency reasons several photon maps are constructed in the first pass. A *caustics photon map* that stores only the photons that correspond to a caustic light path, a *global photon map* that stores all photon hits at diffuse surfaces (including the caustics), and a *volume photon map* that stores multiple scattered photons in participating media. In the following we will ignore the case of participating media (see [7, 5] for details).

The Radiance Estimate

The photon map represents incoming flux in the scene. For rendering purposes we want radiance. Using the expression for reflected radiance we find that:

$$L_r(x,\vec{\omega}) = \int_{\Omega} f_r(x,\vec{\omega}',\vec{\omega}) L_i(x,\vec{\omega}')(\vec{n}_x \cdot \vec{\omega}')\, d\vec{\omega}' \,, \tag{10}$$

where L_r is the reflected radiance at x in direction $\vec{\omega}$. Ω_x is the hemisphere of incoming directions, f_r is the BRDF and L_i is the incoming radiance. To use the

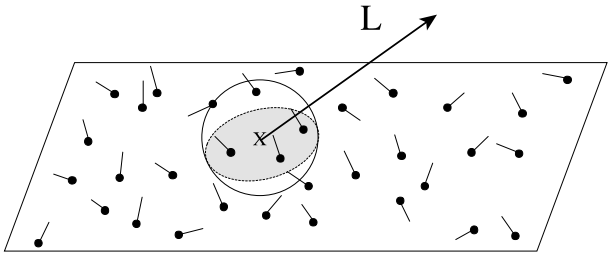


Figure 5: The radiance estimate is computed from the nearest photons in the photon map. From [5].

information in the photon map we can rewrite this integral as follows:

$$\begin{aligned} L_r(x,\vec{\omega}) &= \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) \frac{d^2\Phi_i(x,\vec{\omega}')}{(\vec{n}_x \cdot \vec{\omega}')\, d\vec{\omega}_i'\, dA_i} (\vec{n}_x \cdot \vec{\omega}')\, d\vec{\omega}_i' \\ &= \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) \frac{d^2\Phi_i(x,\vec{\omega}')}{dA_i}\,. \end{aligned} \tag{11}$$

Here we have used the relationship between radiance and flux to rewrite the incoming radiance as incoming flux instead. By using the nearest n photons around x from the photon map to estimate the incoming flux, we get:

$$L_r(x,\vec{\omega}) \approx \sum_{p=1}^{n} f_r(x,\vec{\omega}_p,\vec{\omega}) \frac{\Delta\Phi_p(x,\vec{\omega}_p)}{\Delta A}\,. \tag{12}$$

This procedure can be seen as expanding a sphere around x until it contains enough photons. The last unknown piece of information is ΔA. This is the area covered by the photons, and it is used to computed the photon density (the flux density). A simple approximation for ΔA is the projected area of the sphere used to locate the photons. The radius of this sphere is r (where r is the distance to the n'th nearest photon), and we get $\Delta A = \pi r^2$. This is equivalent to a nearest neighbor density estimate [15]. The radiance estimate is illustrated in Figure 5.

The radiance estimate is biased. There are two approximations in the estimate. It assumes that the nearest photons represents the illumination at x, and it uses a nearest neighbor density estimate. Both of these approximations can introduce artifacts in the rendered image. In particular the nearest neighbor density estimate is often somewhat blurry.

However, the radiance estimate is also consistent. As more photons are used in the photon map as well as the estimate it will converge to the correct value.

A useful property of the radiance estimate is that the bias is purely local (the error is a function of the local geometry and the local photon density).

Pass 2: Rendering

For rendering the radiance through each pixel is computed by averaging the result of several sample rays. The radiance for each ray is computed using distribution ray tracing. This ray tracer is using the photon map both to guide the sampling (importance sampling) as well as limit the recursion.

There are several strategies by which the photon map can be used for rendering. One can visualize the radiance estimate directly at every diffuse surface intersected by a ray. This approach will work (a very similar strategy is used by [14]), but it requires a large number of photons in both the photon map as well as the radiance estimate. To reduce the number of photons the two-pass photon mapping approach uses a mix of several techniques to compute the various components of the reflected radiance at a given surface location.

We distinguish between specular and diffuse reflection. Here specular means perfect specular or highly glossy, and diffuse reflection is the remaining part of the reflection (not only Lambertian).

For all specular surface components the two-pass method uses recursive ray tracing to evaluate the incoming radiance from the reflected direction. Ray tracing is pretty efficient at handling specular and highly glossy reflections.

Two different techniques are used for the diffuse surface component. The first diffuse surface seen either directly through a pixel or via a few specular reflections is evaluated accurately using Monte Carlo ray tracing. The direct illumination is computed using standard ray tracing techniques and similarly the irradiance is evaluated using Monte Carlo ray tracing or gathering (this sampling is improved by using the information in the photon map to importance sample in the directions where the local photons originated). Whenever a sample ray from the gathering step reaches another diffuse surface the estimate from the global photon map is used). The use of a gathering step means that the radiance estimate from the global photon map can be fairly coarse without affecting the quality of the final rendered image. The final component of the two-pass method is caustics, to reduce noise in the gathering step the caustics component is extracted and caustics are instead

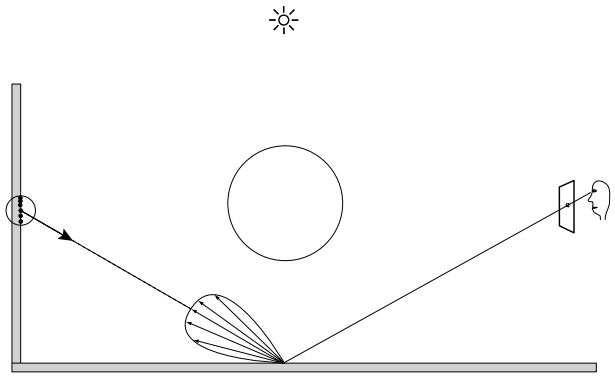


Figure 6: The rendering step uses a gathering step to compute the first diffuse bounce more accurately. From [5].

rendered by directly visualizing a caustics photon map — naturally this photon map should have a higher density than the global photon map used in the gathering step. Fortunately, most noticeable caustics are often caused by focusing of light, so this happens automatically. The gathering approach is illustrated in Figure 6

To make the gathering step faster it pays to use the irradiance caching method for Lambertian surfaces. Photon mapping works very well with irradiance caching since photon mapping handles the caustics which irradiance caching cannot handle very well. In addition photon mapping does not need the expensive recursive evaluation of irradiance values.

Figure 7 shows the rendering of the box scene using photon mapping. Here the global photon map has 200,000 photons and the caustics photon map has 50,000 photons. Notice the smooth overall appearance as well as the caustic on the right wall due to light reflected of the mirror sphere and focused through the glass sphere.

The bias in the photon mapping method is difficult to quantify. It is a mix of a local bias (the radiance estimate) and global bias (irradiance caching), and it is a function of the number of photons in the photon map, the number of photons in the radiance estimate and the number of sample rays. In general the bias from the photon map tends to appear as blurry illumination. Features gets washed out if too few photons are used. However, since the photon map is not visualized directly in the two-pass method it is somewhat harder to predict what the potential errors of using too few photons may be.

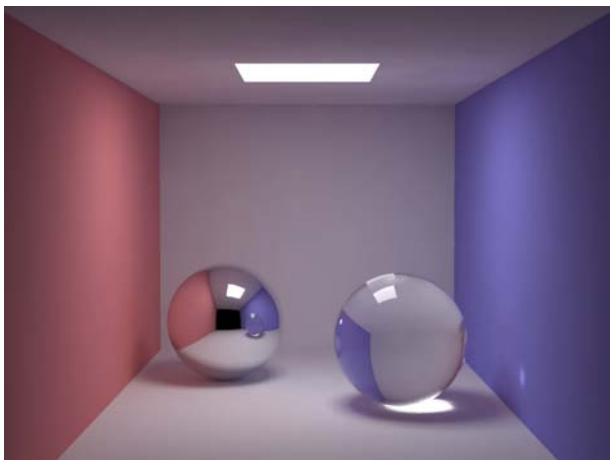


Figure 7: The box scene rendered using photon mapping.

Bibliography

[1] James Arvo and David B. Kirk. Particle transport and image synthesis. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 63–66, August 1990.

[2] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 75–84, August 1988.

[3] Rafael C. Gonzalez and Paul Wintz. *Digital Image Processing (2nd Ed.)*. Addison-Wesley, Reading, MA, 1987.

[4] Henrik Wann Jensen. Global illumination using photon maps. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 21–30, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4.

[5] Henrik Wann Jensen. *Realistic Image Synthesis using Photon Mapping*. AK Peters, 2001.

[6] Henrik Wann Jensen and Niels J. Christensen. Optimizing path tracing using noise reduction filters. In *Winter School of Computer Graphics 1995*, February 1995. held at University of West Bohemia, Plzen, Czech Republic, 14-18 February 1995.

[7] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 311–320. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

[8] David B. Kirk and James Arvo. Unbiased sampling techniques for image synthesis. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 153–156, July 1991.

[9] Mark E. Lee and Richard A. Redner. A note on the use of nonlinear filtering in computer graphics. *IEEE Computer Graphics and Applications*, 10(3):23–29, May 1990.

[10] Mark E. Lee, Richard A. Redner, and Samuel P. Uselton. Statistically optimized sampling for distributed ray tracing. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 61–67, July 1985.

[11] Michael McCool. Anisotropic diffusion for monte carlo noise reduction. *ACM Transactions on Graphics*, pages 171–194, April 1999.

[12] Werner Purgathofer. A statistical method for adaptive stochastic sampling. *Computers and Graphics*, 11(2):157–162, 1987.

[13] Holly E. Rushmeier and Gregory J. Ward. Energy preserving non–linear filters. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 131–138. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[14] Peter Shirley, Bretton Wade, Philip Hubbard, David Zareski, Bruce Walter, and Donald P. Greenberg. Global illumination via density estimation. In *Eurographics Rendering Workshop 1995*. Eurographics, June 1995.

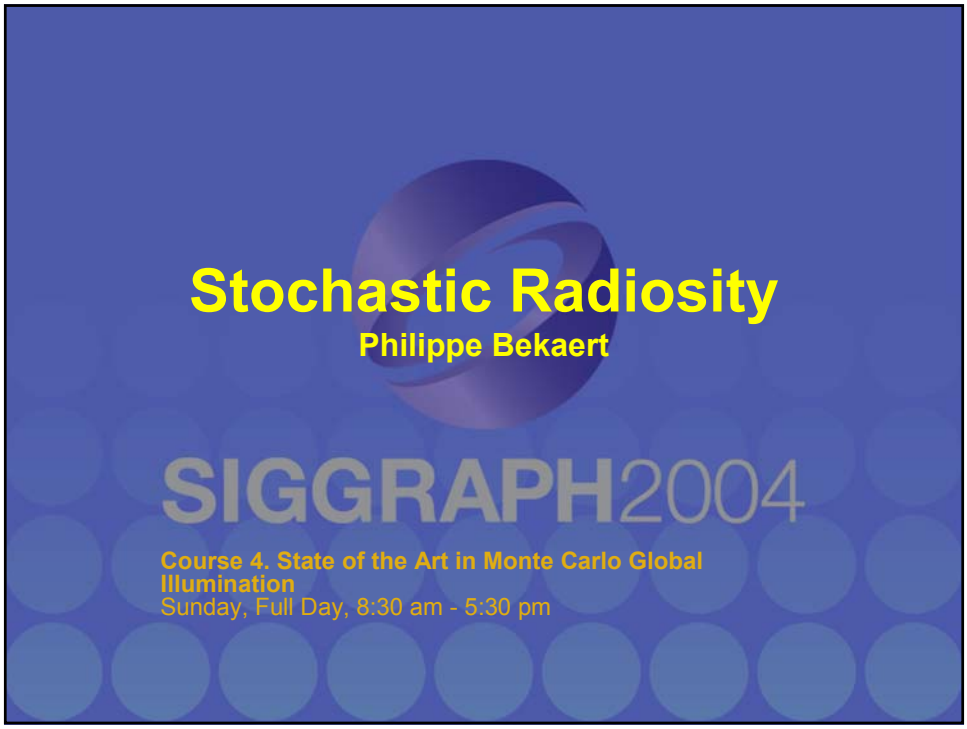
[15] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, 1986.

[16] Frank Suykens and Yves Willems. Adaptive filtering for progressive monte carlo image rendering. 2000.

[17] Rasmus Tamstorf and Henrik Wann Jensen. Adaptive sampling and bias estimation in path @acing. In Julie Dorsey and Philipp Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 285–296, New York City, NY, June 1997. Eurographics, Springer Wien. ISBN 3-211-83001-4.

[18] Gregory J. Ward and Paul Heckbert. Irradiance gradients. *Third Eurographics Workshop on Rendering*, pages 85–98, May 1992.

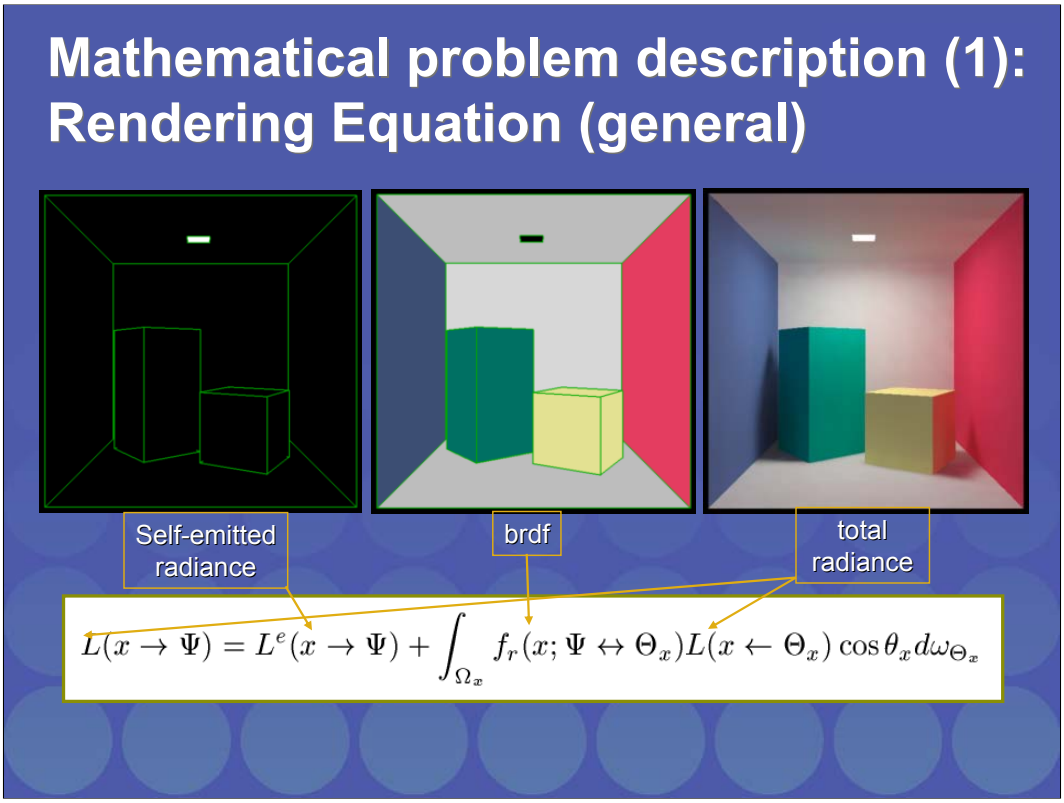
[19] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 85–92, August 1988.


Stochastic Radiosity
Philippe Bekaert
SIGGRAPH2004
Course 4. State of the Art in Monte Carlo Global Illumination
Sunday, Full Day, 8:30 am - 5:30 pm

What will we learn?

- Case study: computation of world-space representation of diffuse illumination

  Over 100 papers on stochastic radiosity.
- Diffuse light path generation using stochastic iteration and random walks
- Different ways how to measure diffuse illumination
- Variance reduction: more efficient light path generation and usage.



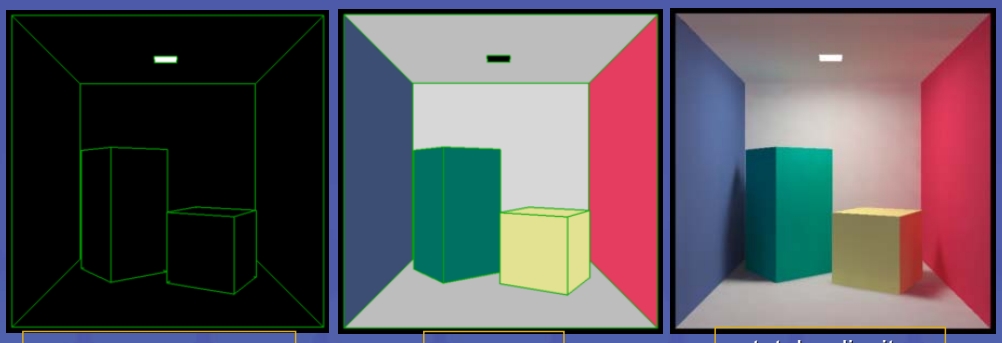
This is the rendering equation, the equation describing light transport that has been introduced earlier in the course. It is the basic equation of all global illumination.

The rendering equation says that the light intensity observed at each point x and into a direction psi is the sum of two terms, on the right hand side of the equation:

-The first term describes spontaneously emitted light, by light sources. This is the light that surfaces emit even if there are no other surfaces around or if all surfaces around are perfectly black (no reflections or refractions)

-The second term, with the integral, describes reflected and refracted light at x. The integral is over all directions theta at x. The integrand is the product of the light intensity (radiance) coming in at x from theta, the brdf fr which tells how intensely light scatters from theta into psi at x, and a cosine factor. The cosine factor is there because radiance is always measured per unit of surface area perpendicular to the light propagation direction.

Mathematical problem description (2): Radiosity Integral Equation (diffuse)



Self-emitted radiosity | reflectivity | total radiosity

$$B(x) = E(x) + \int_S \rho(x) G(x,y) B(y) dA_y$$

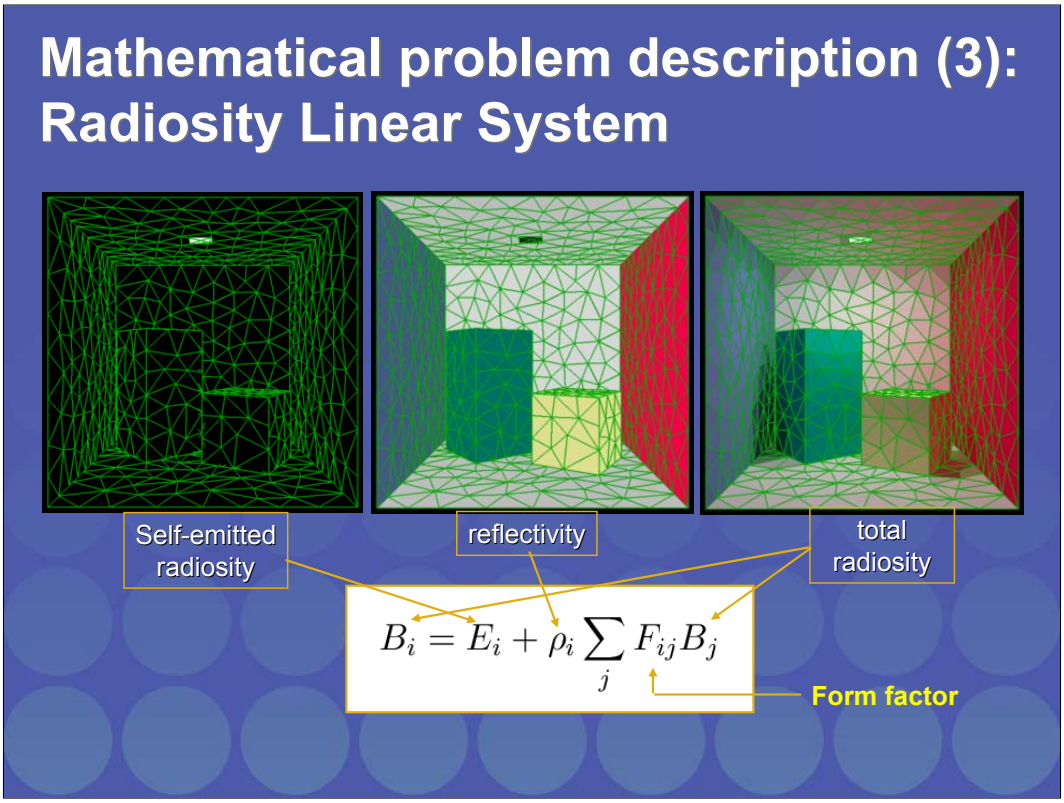
$$G(x,y) = \frac{\cos\theta_x \cos\theta_y}{\pi r_{xy}^2} \mathrm{vis}(x,y)$$

The rendering equation becomes considerably more simple if light emission and reflection is all diffuse (Lambertian).

In that case, the radiance leaving a surface does not depend on direction. It is more convenient to express the integral as an integral over surface points rather than over directions. It is also more convenient to express illumination intensity by means of the quantity “radiosity” (Watts per square meter), rather than radiance (Watts per square meter perpendicular to the propagation direction, and per unit of solid angle). For a diffuse surface, radiosity equals pi times radiance. Instead of the brdf fr, we use the quantity “reflectivity”, which equals pi times the (constant) brdf on a diffuse surface.

-The additional cosine, the multiplication by vis(x,y), the visibility predicate, and the division by square distance between surface points x and y, are introduced by the transformation of integration variable: from directions (or better: differential solid angles) to points (or better: differential surface area).

-The division by pi is there because reflectivity rho is pi times the brdf.



And this is what results if we also assume that radiosity B(x) and self-emitted radiosity E(x) are constant over small pieces of object surface called patches or elements: a system of equations relating the (to be computed) radiosities Bi on the surface elements with each other.

This system of equations says basically the same as the rendering equation: that the light leaving a surface element is the sum of two terms: the self-emitted light, and the light reflected of other surfaces. The light reflected of other surfaces is a weighted sum of the light emitted by these other surfaces. The weights Fij are called form factors. The sum yields the amount of light incident on an element. Multiplication by the reflectivity rho, tells us how much of that incident light is reflected. The reflectivities are numbers between 0 and 1, one number per considered wave length. (More about the form factors a few slides further in this course)

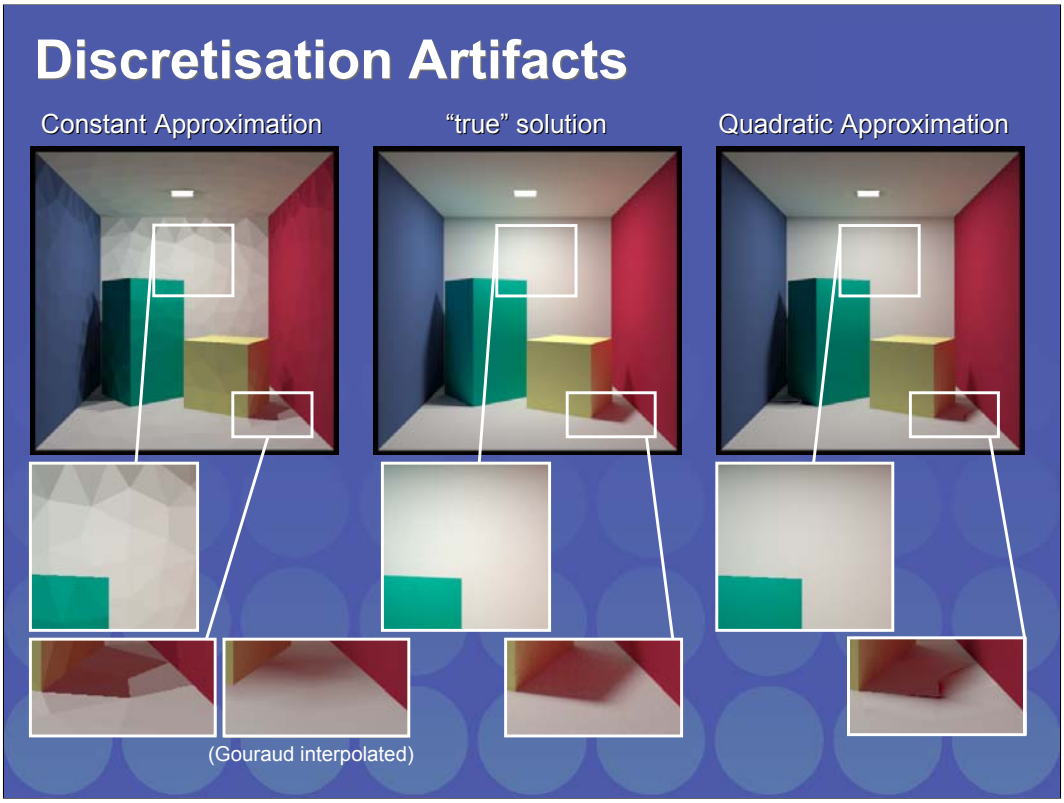
If we are given the self-emitted radiosity for all the elements, and the reflectivity, then we can compute the total radiosities, also containing the effect of light interreflections, by solving this system of equations. There are as many equations and unknowns as there are surface elements. The systems to be solved can be very large: 100,000 surface elements, equations and unknowns is common.

Classical Radiosity

1. Discretise the input scene
   Problem: **discretisation artifacts**
2. Compute form factors
   Problem: huge number of non-trivial integrals: 95% of the **computation time**, very large **storage requirements**, **computational error**.
3. Solve radiosity system
4. Tone mapping and display

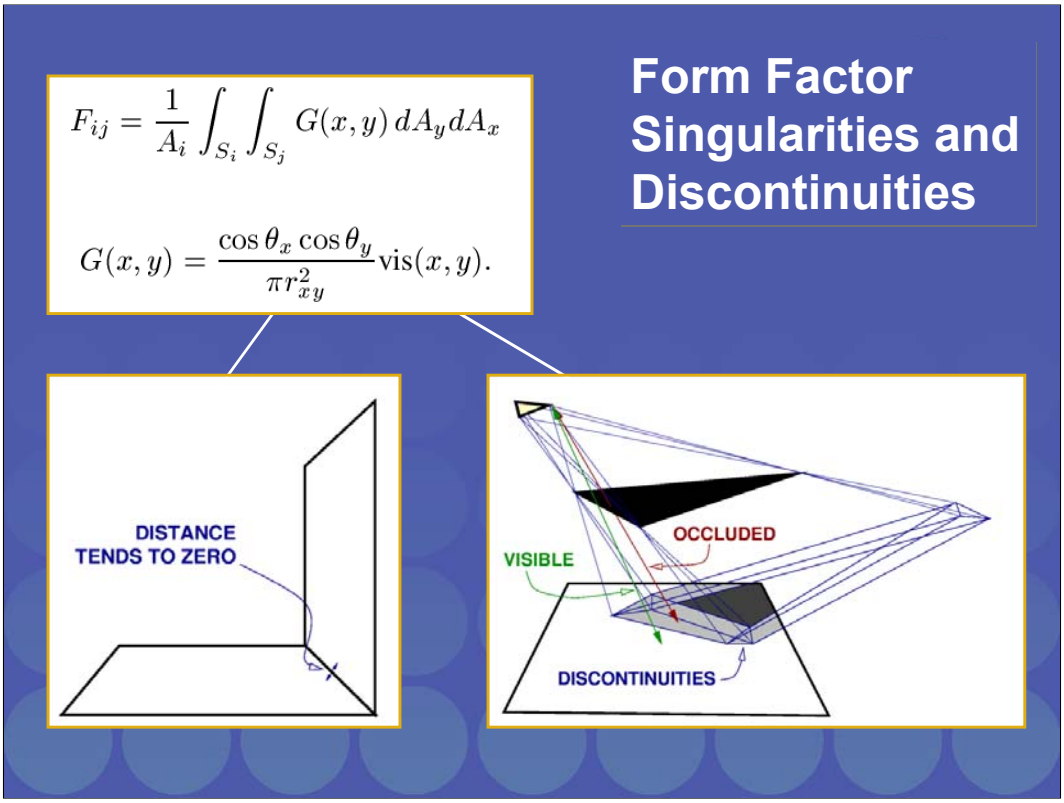
❹ **In practice intertwined!**

This slide describes the steps taken to solve the problem, in the so called classical radiosity method as it was proposed in the early 1980-ies. The problems are described in more detail on the next two slides.



One of the major problems of such radiosity methods are discretisation artifacts. The left column shows what our assumption of constant radiosity B(x) over the surface elements leads to: discontinuities at surface element edges. You can smooth these out, by means of Gouraud interpolation on graphics hardware. But you probably agree that it does not really look very well either, near shadow boundaries. The middle column shows what the images should look like (it was computed with bidirectional path tracing and a lot of CPU time).

A lot of research has been spent on reducing or eliminating such discretisation artifacts. Proposed techniques include the use of higher order (linear, quadratic, …) radiosity approximations on the elements, and discontinuity meshing. In the latter, one tries to shape the elements in such a way that their edges correspond with illumination discontinuities, e.g. inner and outer penumbra boundaries. As you see in the right column, a higher order approximation on the same set of elements does yield very nice results in smoothly lit areas. Near the shadow boundary however, higher order approximations are basically no better than constant approximations, and discontinuity meshing would be needed for a perfect picture.



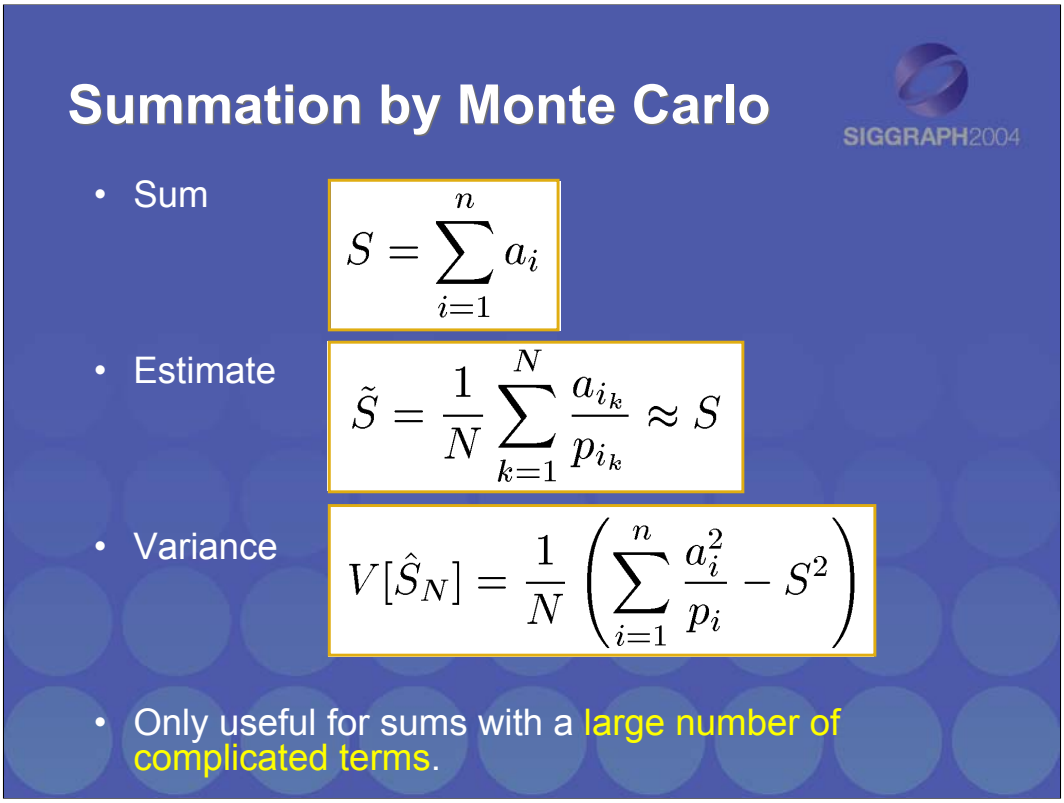
The other main problem with the classical radiosity method is in the computation of the form factors. The number of form factors is huge: N*N if there are N elements, so 10G if you have 100K elements (this is common). But each form factor also requires the solution of a non-trivial 4-dimensional integral, with potentially discontinuous (where visibility changes, e.g. at shadow boundaries) and singular (where the squared distance in the denominator goes to zero, e.g. abutting elements) integrand. The mere storage of the form factors can be overwhelming: even on very high-end contemporary computers, you won't have sufficient RAM to storage 10 billion floating point numbers. In spite of a variety of improvements suggested in the past, including progressive radiosity, and grouping of elements (clustering), these problems have limited the radiosity method to models of rather moderate complexity.

We shall see that Monte Carlo solution of the radiosity system of equations is an interesting alternative that suffers much less from these problems.

Monte Carlo Methods

- Principle:
  - Formulate solution of a problem as the expectation of a random variable
  - Sample the random variable
  - Mean of samples yields estimate for the expectation, and thus for the solution
- Example: **simple estimation of a sum**

First, a brief reminder is given of what Monte Carlo methods are.



And here's an example: not an integral this time. You can use Monte Carlo methods also for computing sums.

The random variable used here consists of events with outcomes or scores a_i/p_i and probabilities p_i. The expectation equals S, the sum to be computed. In plain English: we randomly select terms from the sum, with probabilities p_i. The average ratio of the value of the selected terms, over the probability by which we selected them, yields an estimate for the sum.

We do not have deterministic error bounds for the result, but rather we have pretty realistic stochastic error estimates. The variance tells us how accurate we will be. With approximately 68% probability, our estimate will have error less than the square root of the variance. In about 95% of the cases, we will be wrong by less than 2 times the square root of the variance. And the chance that we are off by more than 3 times the square root of the variance is less than 0.3%. The square root of the variance is called "standard deviation".

We are free to choose the probabilities as we like, as long as they are normalised (they sum up to one) and they are not zero unless a_i is zero as well. But the probabilities do have a major impact on the accuracy. The whole art therefore is in choosing easy-to-sample probabilities, that yield a low variance. A good choice often results by taking probabilities that are approximately proportional to the magnitude of the terms (importance sampling). But this is not the only way to get low variance, as we shall see.

Since computers are already very good at adding up numbers, Monte Carlo summation is only useful if you have very large sums, or sums in which each term is not a simple number, but the result of a lengthy computation in turn, for instance. And that's exactly what we have in radiosity.

Monte Carlo Methods

- Advantages:
  - Correct result eventually
  - Simple (at least in theory)
  - Wide applicability
- Disadvantage:
  - Slow convergence rate (method of last resort!!)
- Remedies:
  - Variance reduction
  - Low-discrepancy Sampling
  - Sequential Sampling

This slide describes the advantages and disadvantages of Monte Carlo methods. The particular mix of wide applicability, a simple and elegant concept, and relatively easy error control, in combination with a slow convergence rate, makes that there are probably very few, if any, other numerical methods in which so many CPU cycles are burnt.

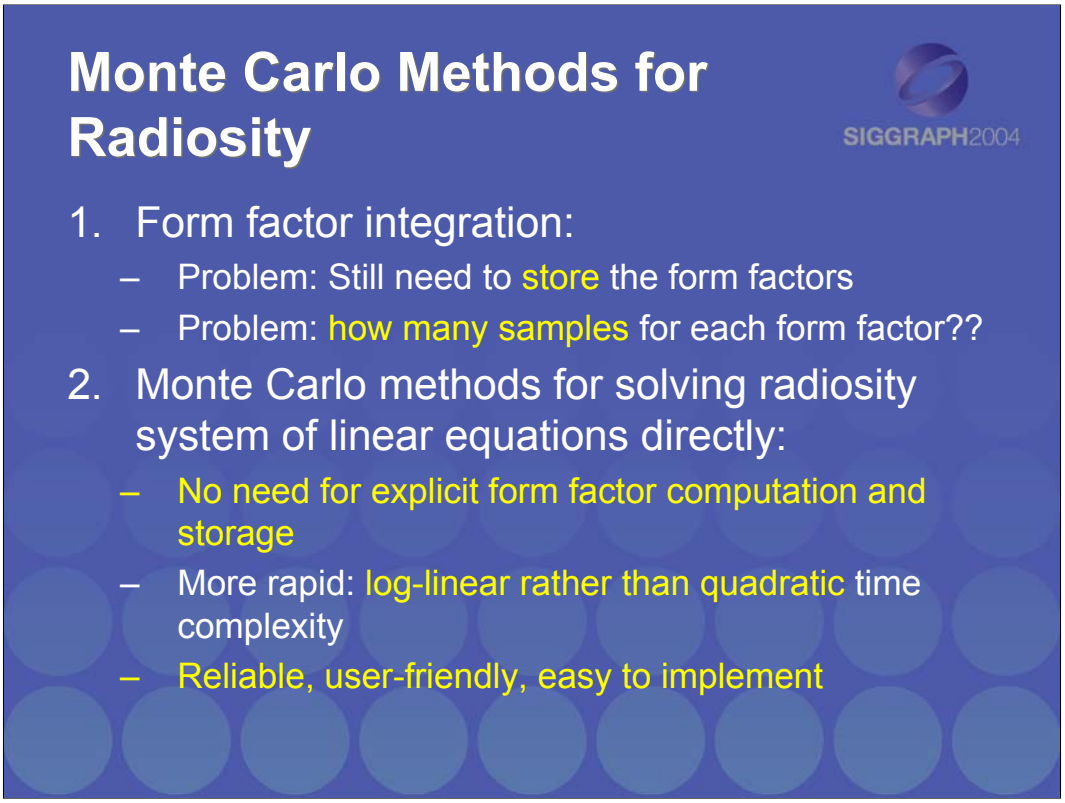
This particular mix of properties has also given rise to considerably research in improving Monte Carlo methods:

-The transformation of a random variable into a better random variable with same expectation, where better means “lower variance”, is called variance reduction. Importance sampling is an example of a variance reduction technique, so is stratification, correlated sampling, weighted importance sampling and a bunch of other techniques found in standard Monte Carlo text books such as the 1986 book by Kalos and Whitlock (citation at the end of these slides).

-Low discrepance sampling basically means to use specially crafted deterministic number sequences instead of (pseudo-) random numbers (“pseudo” because you can’t make true random numbers with an algorithm alone: it takes special hardware). The deterministic numbers give up unpredictability in favor of uniformity: they better “fill” an interval, area, volume, … Better uniformity can imply faster convergence, but giving up unpredictability means that one has to take care not to introduce other sources of error.

-Sequential sampling refers to certain multi-stage algorithms that are currently being developed. Very roughly stated, the idea is to learn good variance reduction (importance sampling pdf, or correlation strategies) while going on. But this is much easier said than done …

In short: keep an eye on Monte Carlo methods. They are getting better, computers are getting faster, people remain as lazy as they are, and so Monte Carlo methods will probably only gain importance in the future. But don’t try to solve every problem with Monte Carlo either: it’s a method of last resort, after all: Many, especially relatively simple, problems can be solved much more efficiently with other methods.



We now turn to Monte Carlo methods for radiosity.

Form factors can be computed using well understood and often applied Monte Carlo integration methods, instead of deterministic integration methods, and in the context of usual solution methods for the radiosity system of equations such as Gauss-Seidel or progressive radiosity (Southwell iterations). But that does not really solve the problem.

It is better to look into Monte Carlo solution methods for linear systems directly. Indeed, you can use Monte Carlo not only to calculate integrals or for summation, but also for solving linear systems. And that has a couple of nice properties in the context of radiosity.

Jacobi Iterative Method for Radiosity

- Power equations:

$$P_i = \Phi_i + \sum_j P_j F_{ji} \rho_i$$

- Deterministic Jacobi Algorithm:

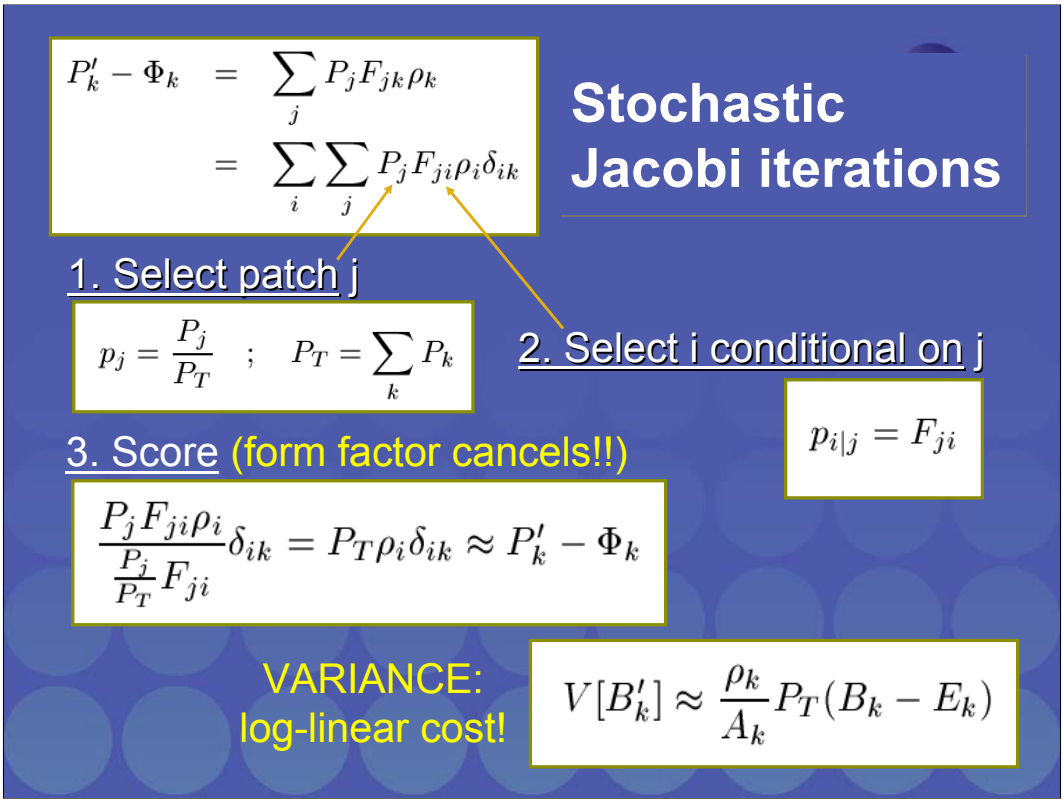
1. Initialise $P_i = \Phi_i$;
2. While not converged, do
   (a) for all i: $P'_i \leftarrow \Phi_i + \sum_j P_j F_{ji} \rho_i$;
   (b) for all i: $P_i \leftarrow P'_i$.

Quadratic cost!

Here's a first method for doing so: the stochastic Jacobi method. Later, we shall see another method: by means of random walks.

The classical (deterministic) Jacobi method works as follows: find an initial approximation of the solution of a linear system. Fill in that approximation in the right hand side of the equations. What comes out on the left hand side is our new approximation. Under certain conditions (fulfilled in the case of radiosity), the new approximation will be closer to the solution than the previous one. Each such step takes a matrix-vector product and thus has quadratic cost in terms of the number of unknowns.

We will do the same, but using Monte Carlo estimtation for these matrix-vector products. This is illustrated on the next slide. We will do so using an alternative formulation of the radiosity equations: in terms of the flux, or power, emitted by the patches (unit: Watts) instead of radiosity (units: Watts per square meter). The power equations are obtained from the radiosity equatiions by multiplying left and right by the patch surface area. We do so to get the indices of the form factor in reversed order that makes it better suited for Monte Carlo sampling: A_i F_ij = A_j F_ji .



The matrix-vector products are estimated in a two-step procedure, illustrated here. We basically randomly select pairs of patches in our environment to be rendered, and average a contribution (shown in 3) associated with each pair. The equations show one component of the solution, on the patch with index k. The other components are calculated simultaneously (using the same set of patch pairs) and in an identical fashion (only the scores (3) are different).

The pairs are sampled as follows: First we randomly select a “source” patch j. Next, we choose a “destination” patch i, conditional on j. j is selected based on its amount of power: more chance to select a bright patch j. I is selected conditional on j with probability equal to the form factor F_ji: high chance of the form factor is large, little chance if the form factor is small. Indeed, form factors are numbers between zero and one, they sum to at most one, so they can be interpreted as probabilities. There exist practical algorithms for sampling them (next slide), based on ray tracing. We typically sample a few tens or hundreds of thousands of such pairs of patches for estimating the matrix vector product in each iteration.

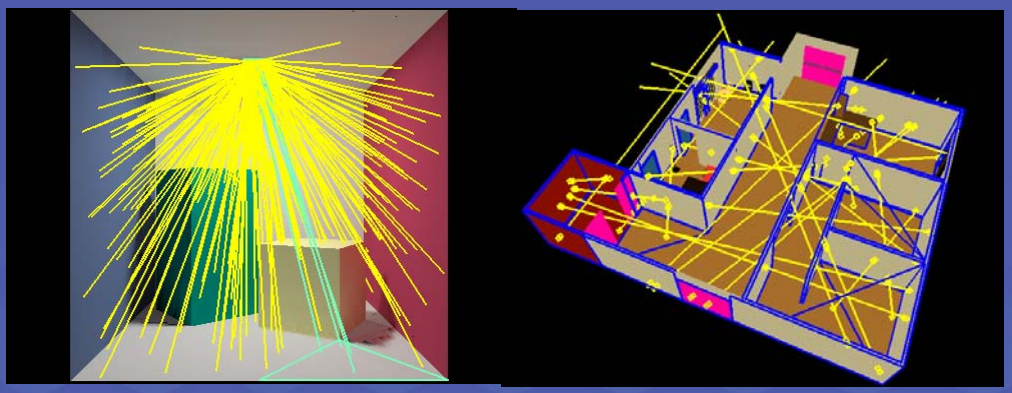
Because we take the form factor into account while sampling, its numerical value appears in numerator and denominator of the ratios “value over probability” shown in (3), so it cancels out. This means that we never need to compute the value of any form factor. And for the same reason, we also don’t need to store any form factors. And that paves the way to rendering much more complex environments with the radiosity method, than with other, deterministic, strategies.

The delta in the top-left formula box, is Kroneckers delta: 1 if the indices are equal, and 0 otherwise.

The accuracy of our estimate is dictated by the variance. Several studies have been made of this variance. It turns out that the cost of the matrix-vector estimation in this way is roughly log-linear in the number of patches, rather than quadratic. The truth is more complicated than that, but Monte Carlo estimation is often much faster than deterministic summation in this context.

Form factor sampling

SIGGRAPH2004

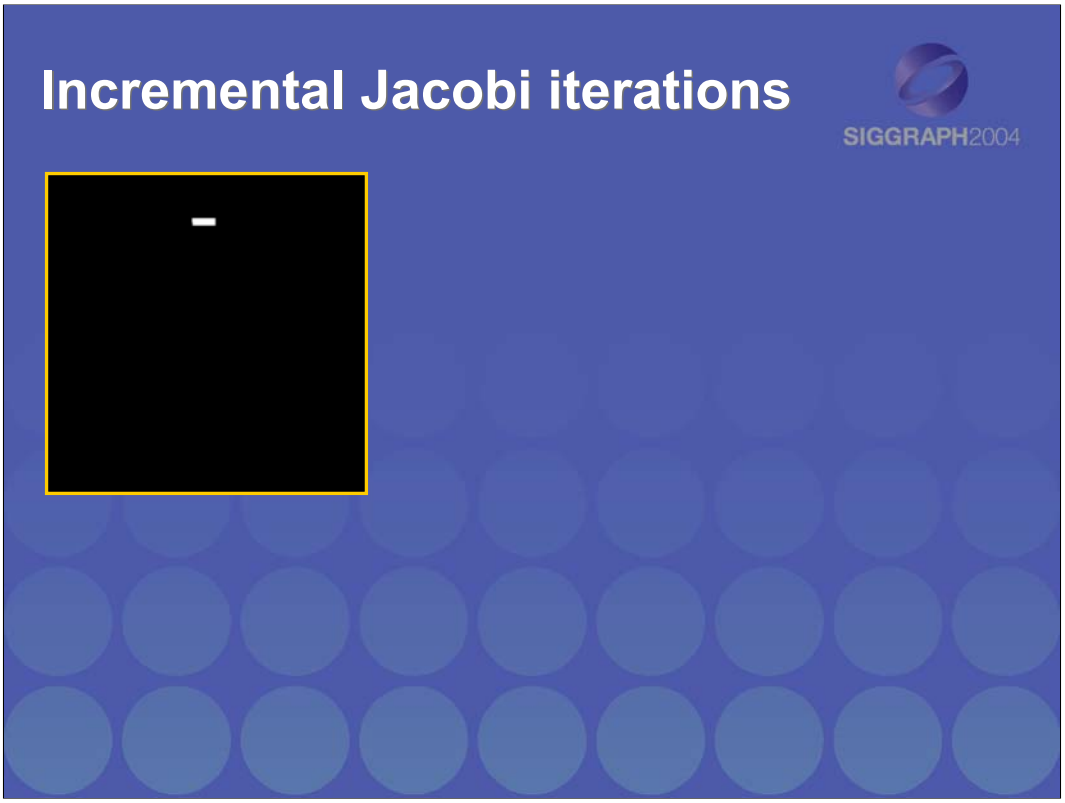


Local Lines

Global Lines

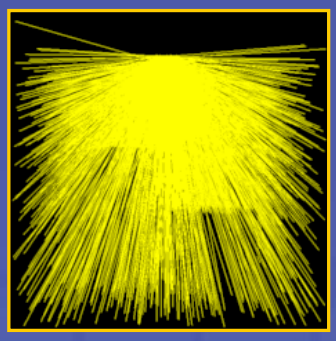
- Form factors F_{ij} for fixed patch i form a probability distribution that can be sampled efficiently by tracing rays.

The best known method for sampling form factors is by shooting a ray with direction making a cosine-distributed angle w.r.t. the surface normal at the source patch. This is called “local” line sampling. It is not the only way to sample according to form factors: Several other algorithms have been developed, based on integral geometry principles.

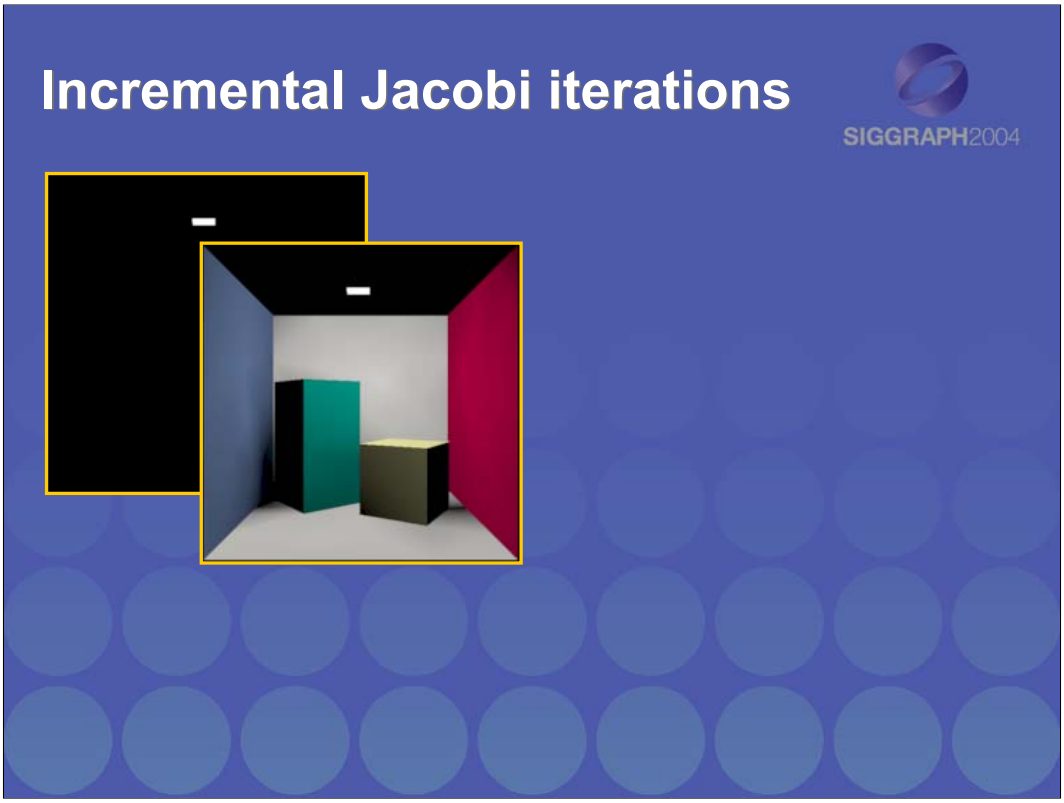


Here, we illustrate the working of the algorithm in practice. Initially, only light sources have power to emit.

Incremental Jacobi iterations



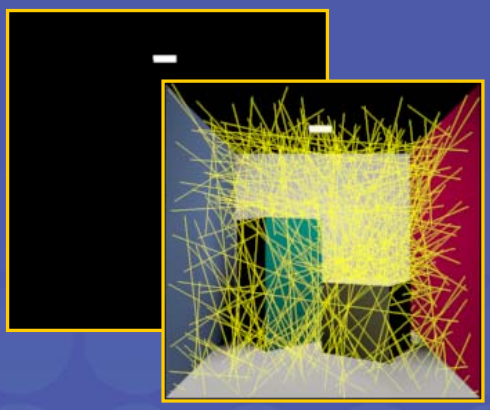
In the first iteration step, therefore, a shower of cosine distributed rays is shot from the light source.



Each shot ray can be viewed as a particle carrying some power from the light source. The particles deposit their power on the surface they hit. This yields an estimate for direct illumination at this stage.

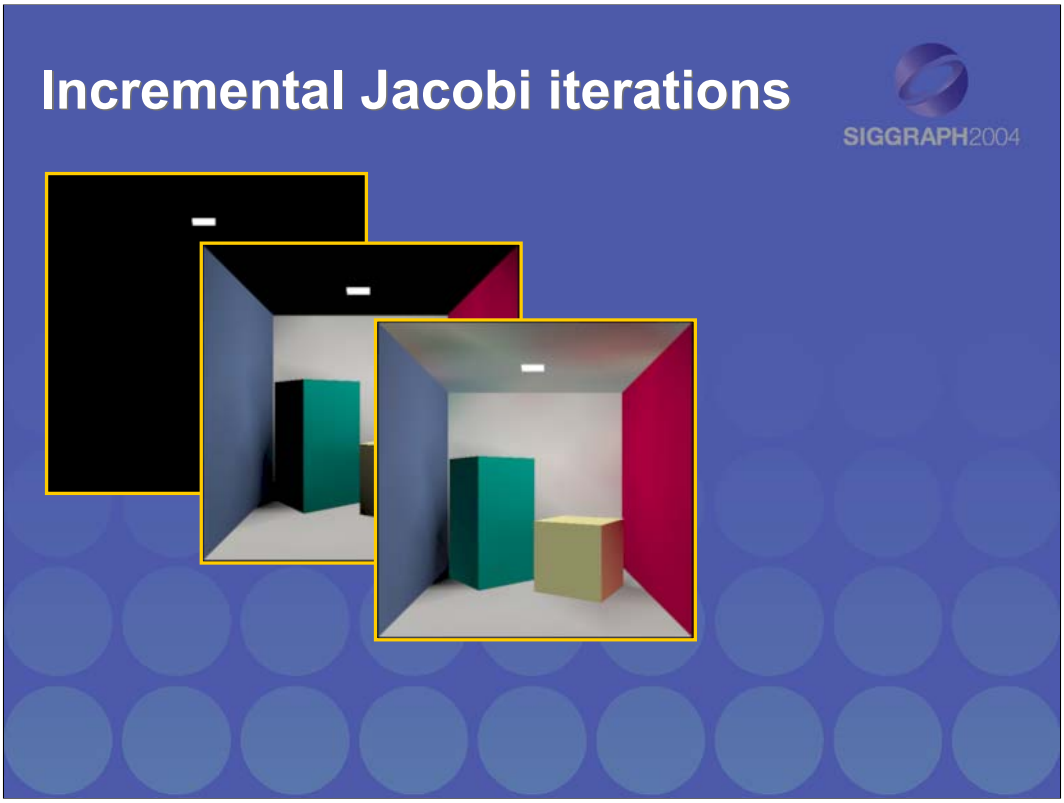
Incremental Jacobi iterations

SIGGRAPH2004

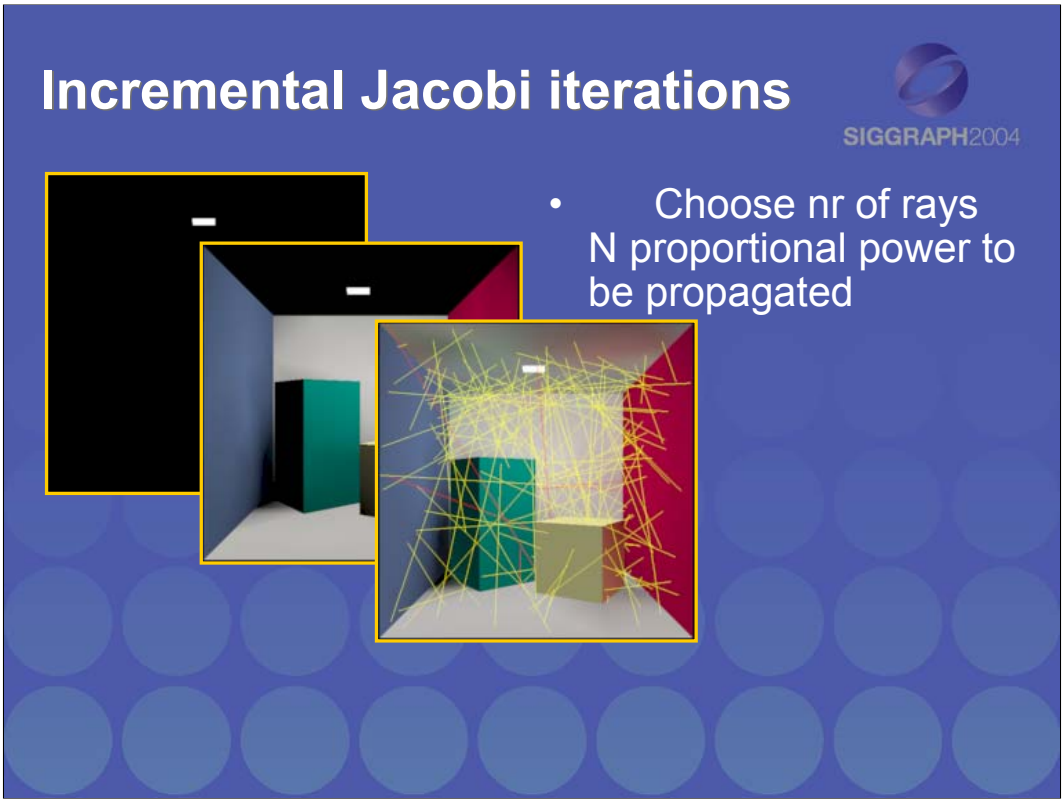


- Propagate only power received in last iteration until the amount drops below a certain threshold.
- Result = sum of results of all steps.

The procedure is repeated. At this stage, we only propagate the directly received light energy.



And the rays that propagate directly received light energy yield first-order indirect illumination.



We shoot new rays for distributing that first-order indirect illumination.

And that yields second-order indirect illumination.

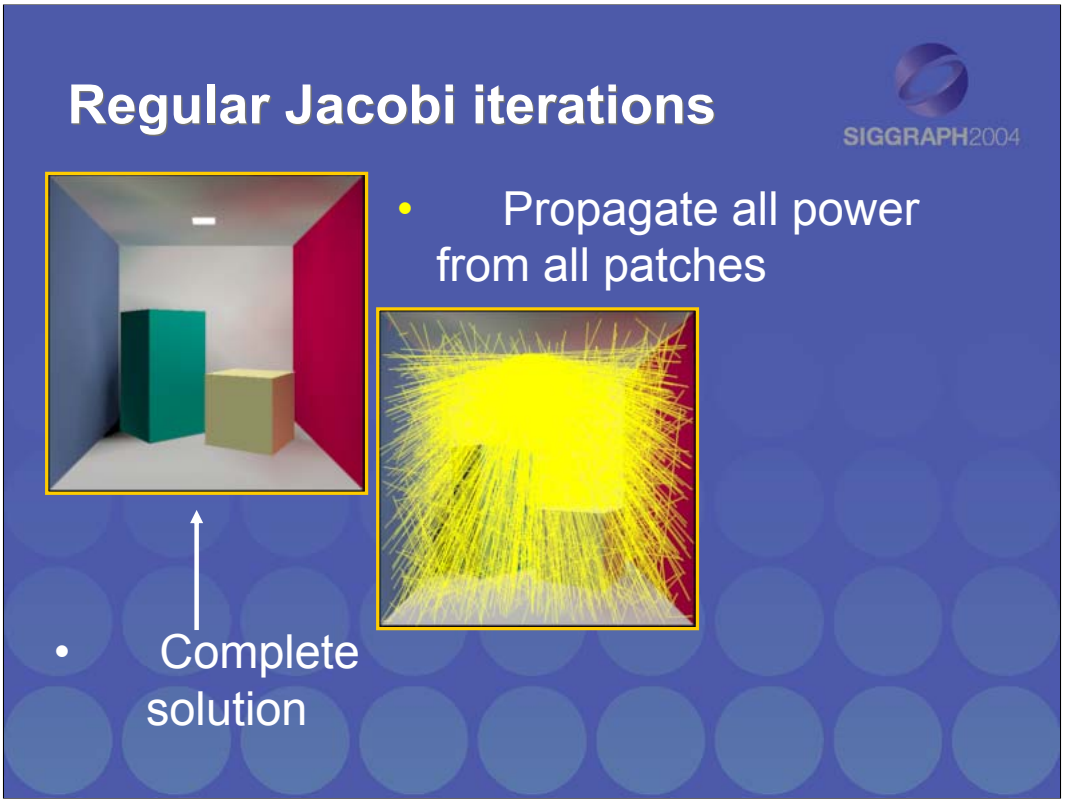
And so we continue …

… until our radiosity (or power) distribution does not significantly change anymore. At this point, we have a first, quick, rough, approximation for the total radiosity (including indirect illumination) in the scene.

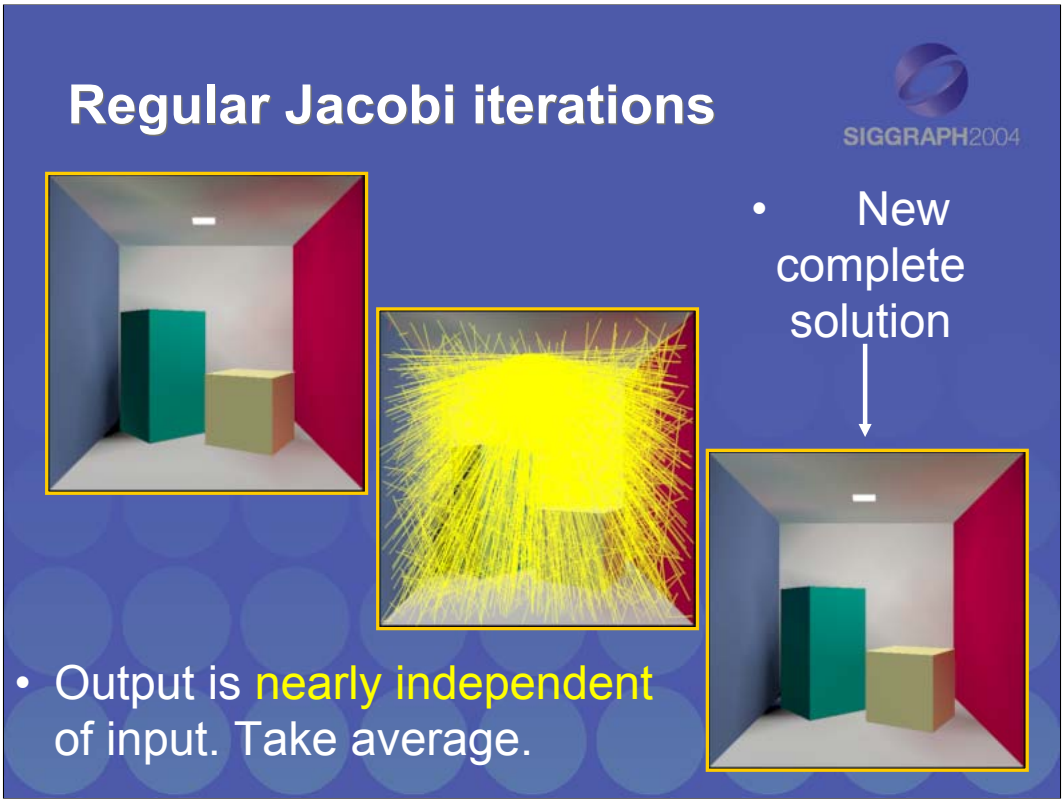
Regular Jacobi iterations

- Complete solution

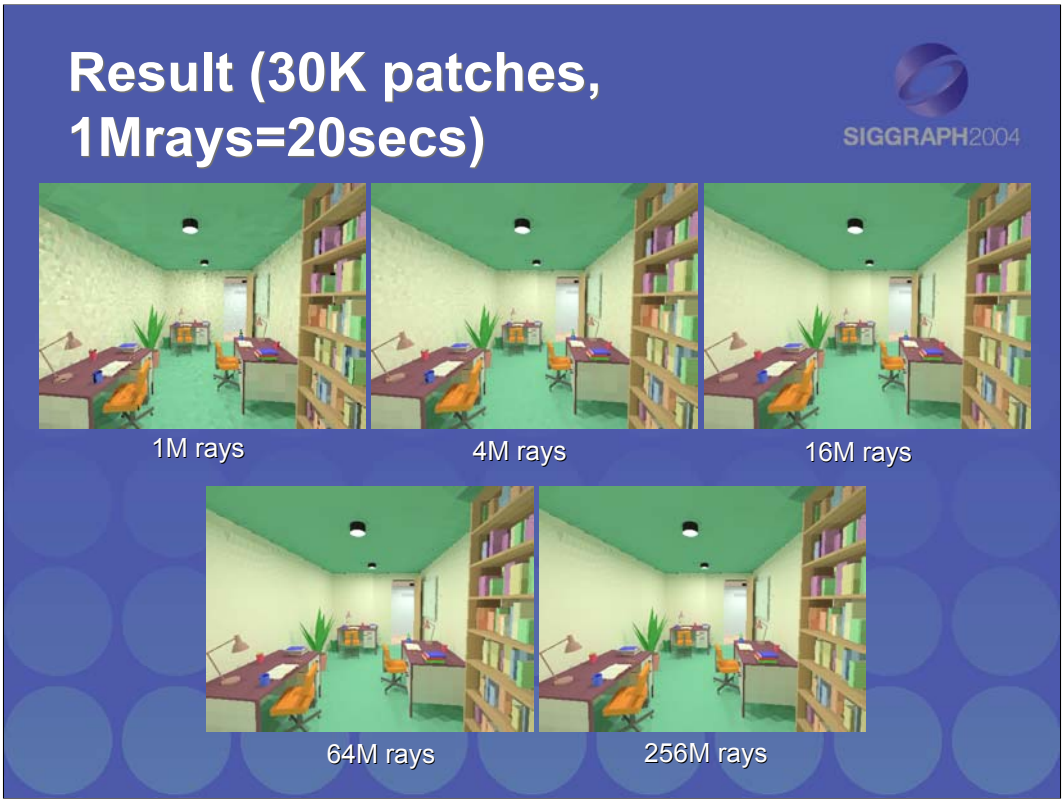
This initial approximation is complete, but may be noisy. Subsequent iterations serve to reduce noise.



We do so by repeating the same ray-shooting iterations, but propagating all power from the patches, not only the power received in the previous iteration. The number of rays is always chosen proportional to the amount of power to be propagated (each ray shoots the same amount of power).



It turns out that we can simple average the newly received power distribution with the old complete one.



Progression of the algorithm: if we would shoot just a single ray for each form factor to be computed in this 30k patches environment, we would need to shoot 900 million rays. The stochastic Jacobi method yields final results with just a few tens of millions of rays in this case and yields useable feedback early on.

Note that we did not even use a very fast ray tracing implementation here. With high-end ray tracing engines, for instance used in interactive global illumination work, you can shoot about a million rays per second in an environment as shown here, on a single CPU.

Random Walks

- Sequence of "states" X_i generated as follows:
  1. Sample origin X_0 according to some source density $S(X_0)$
  2. At each visited state X_i,
     1. EITHER terminate the random walk according to some absorption probability function $A(X_i)$
     2. OR make transition to new state X_{i+1} according to a transition probability density $T^*(X_i,X_{i+1})$

Combined transition pdf $T(X,Y) = (1-A(X))\ T^*(X,Y)$

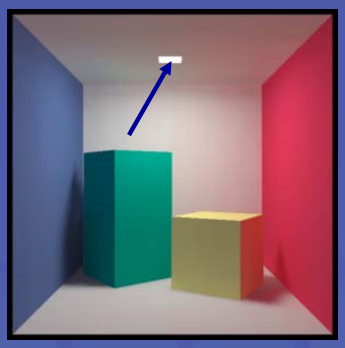
Survival probability

We now turn to a, at first sight, very different way of computing radiosity, based on the notion of random walks. We introduce some "jargon" on this slide: states, source density, absorption, transition and survival probability density.

The random walks we will use here, correspond to light particle paths, originating at light sources, and bouncing around in the scene to be rendered according to the laws of physics (or our approximation thereof), until absorption.

We're interested in the locations where these particles hit surfaces. The states referred to on this slide, correspond to object surface locations.

Light source sampling



- Sample point on light source with probability proportional to self-emitted radiosity: $S(x) = E(x)/\Phi_T$

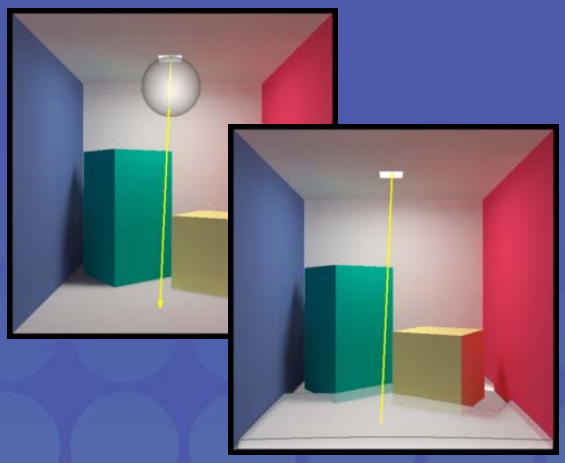
Random walk generation (as needed here) starts with sampling a location on a light source: large probability for bright sources, small probability on dim sources.

Making the first transition (1)

- No absorption at the origin
- Sample direction according to directional distribution of self-emitted radiance.
  - Diffuse emission: pdf is $\cos(\theta_x)/\pi$

Choose a direction for a ray originating at the light source position.

Making the first transition (2)

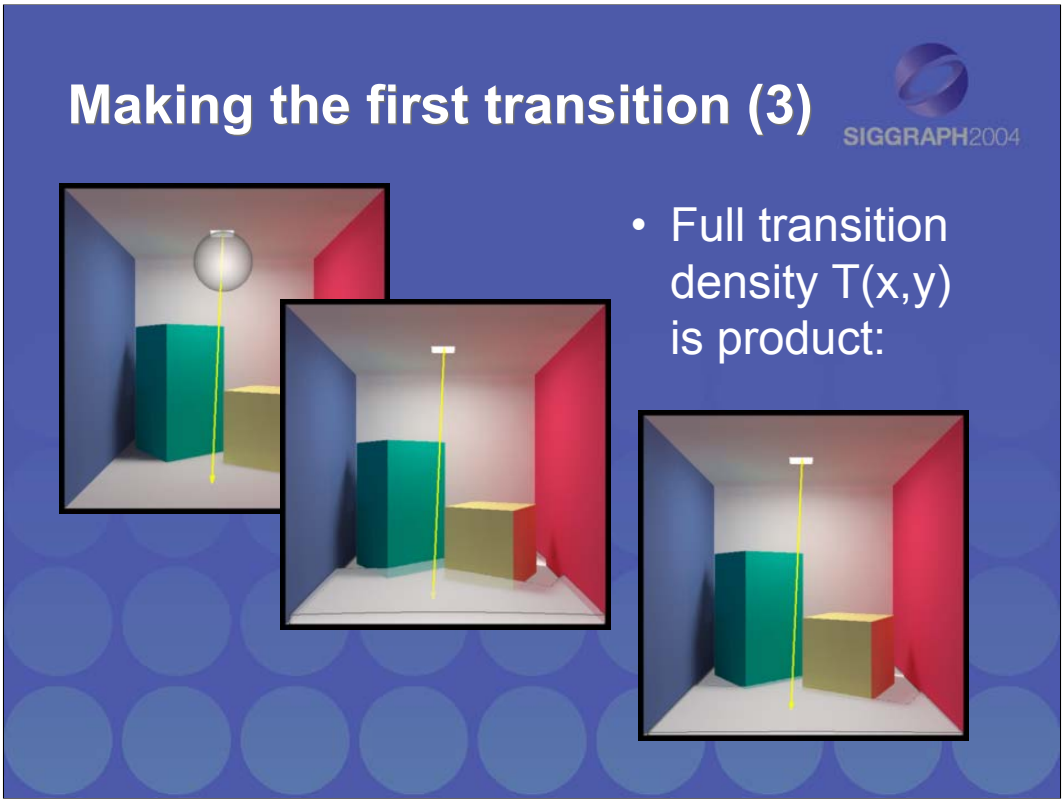


- Shoot ray along sampled direction.
- Geometric density factor:

$\cos(\theta_y) / r^2_{xy}\, vis(x,y)$

Shoot the ray. The transparent surface shows the probability density of hitting locations on the bottom of this simple environment. There's a similar density on the other surfaces.

Even if we would choose ray directions uniformly, the "hit point density" of such rays is not uniform: it decreases with the square of the distance, it depends on the orientation of the hit surface w.r.t. the ray origin (cosine factor) and it is zero for surfaces invisible from the ray origin location.



The transition density for our random walk is the product of both previous densities: the probability density for sampling a ray direction, and the “hit point density” of the rays.

Further transitions

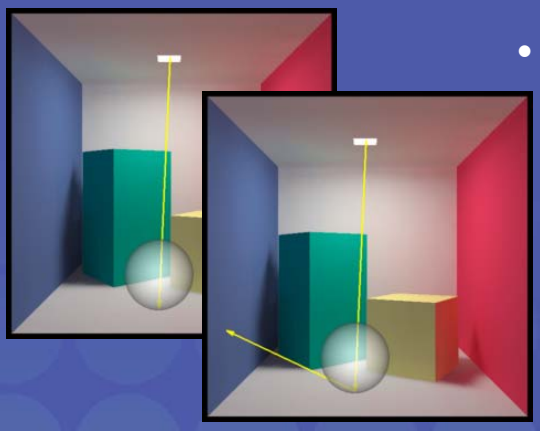
SIGGRAPH2004

- 1. Absorption / survival test according to albedo

We do a survival test on the hit surface: the survival probability is chosen equal to the reflectivity of the surface: so, much chance of survival on a bright surface, much chance of absorption on a dark surface.

Further transitions (2)

SIGGRAPH2004



- 2. Sample direction according to brdf

If survival was sampled, we choose a ray direction, according to the brdf this time, rather than the directional emission characteristics. For a diffuse surface, we use a cosine distributed ray direction, illustrated by the semi-transparent sphere in this image.

Further transitions (3)

And we shoot the just constructed ray. The hit point density of that ray on the left surface is shown here.

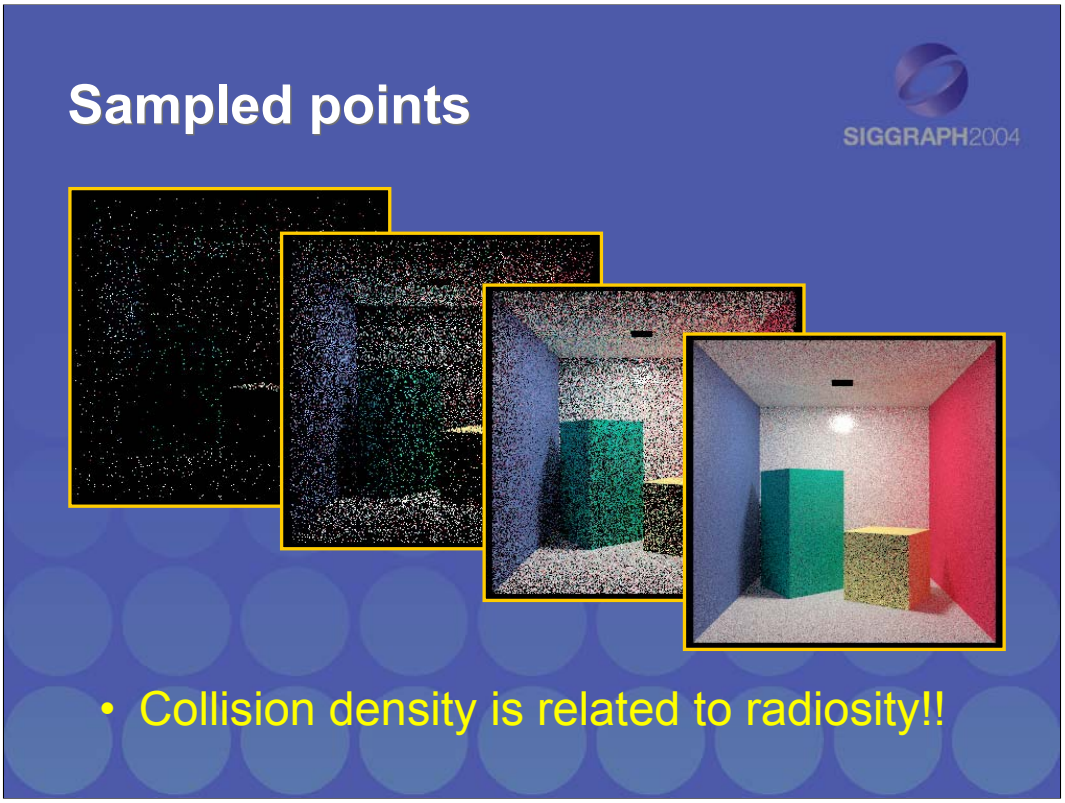
And the full transition density is the product of both.

Once more …

Illustrated here for non-diffuse reflection.

The transition density is zero on the lower-back part of the red wall: it is occluded from the ray origin.

So we continue until absorption.



Let's now have a look at the sampled locations: the hit points of 1000, 10,000, 100,000 and 1,000,000 light paths are shown here.

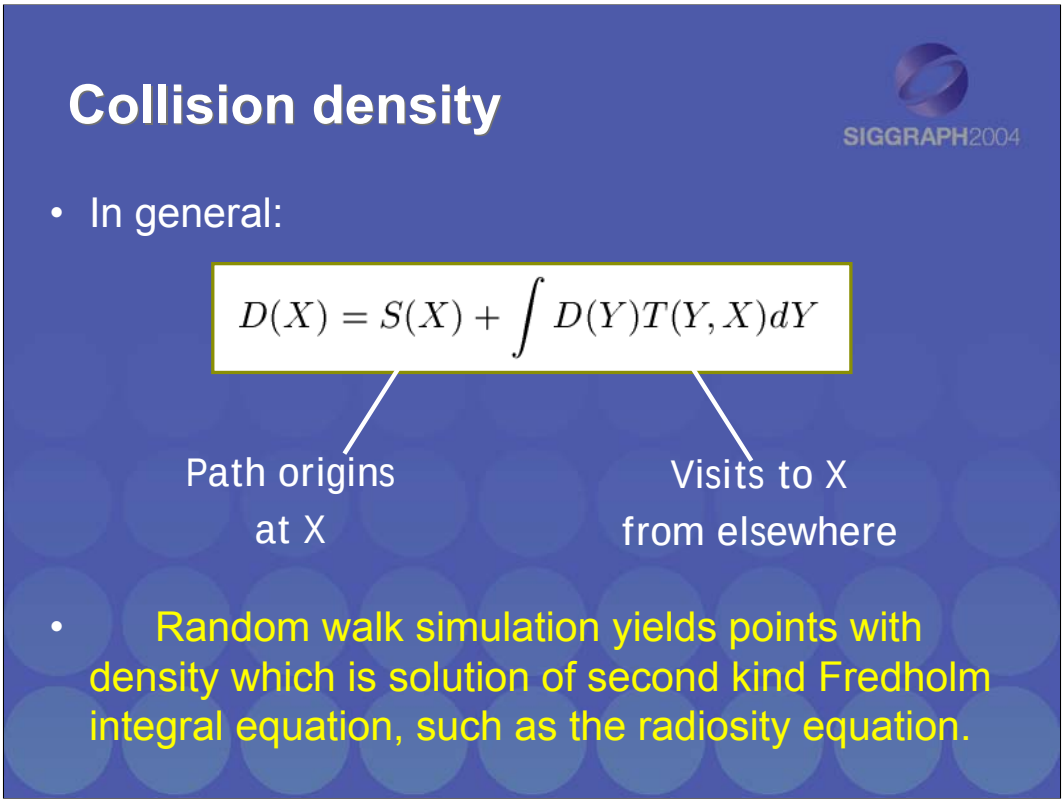
It turns out that the density of the sampled locations is proportional to the radiosity.

This should not surprise us too much: it's simply the way we think nature works: Photons originate at light sources, with more photons originating on bright light sources, and fewer on less bright sources. These photons bounce around in the scene according to the laws of light reflection and refraction and eventually get absorbed. Each photon carries a fixed amount of energy (corresponding with each wavelength, which is perceived as color). Where we find many photons close to each other, there's a lot of energy per unit of surface area. Other areas are less frequently visited by our photons, so we will find less energy per square meter there. Radiosity is light enery per unit of surface area, remember?

Note that this is also true in non-diffuse scenes!

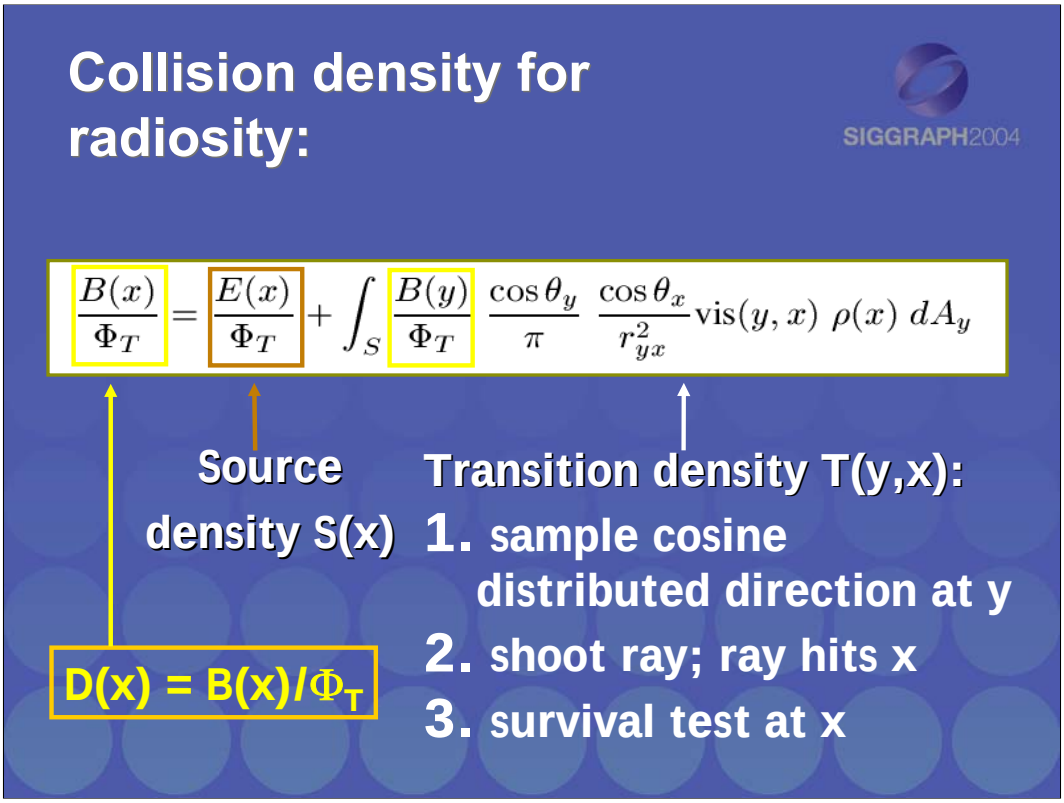
We will however only estimate the radiosity and not try to estimate the directional distribution of the illumination at each location (the radiance). We could do that along the same lines as will be developed here, but compact storage of the directional distribution is problematic in practice, and requires much more samples than for radiosity, except for near-diffuse surfaces.

Collision density



So, in short: tracing random walks is a way to sample surface points with a density that is proportional to the light intensity (the radiosity).

We thus have reduced the radiosity problem to a density estimation problem. Density estimation is a well studied discipline in statistics, and a good introductory text to it is Silvermans 1986 book, cited at the end of these slides. We will present a few density estimation techniques soon.



Here, we make the analogy with the radiosity equation more clear.

Radiosity reconstruction

- Need to compute scalar products

$$\tilde{B} = \int_S M(x) B(x) dA_x$$

Measurement function

- Estimates look like:

$$\frac{1}{N^{RW}} \sum_s \frac{M(x_s) B(x_s)}{B(x_s)/\Phi_T} = \frac{\Phi_T}{N^{RW}} \sum_s M(x_s)$$

Here's a different interpretation of the same result:

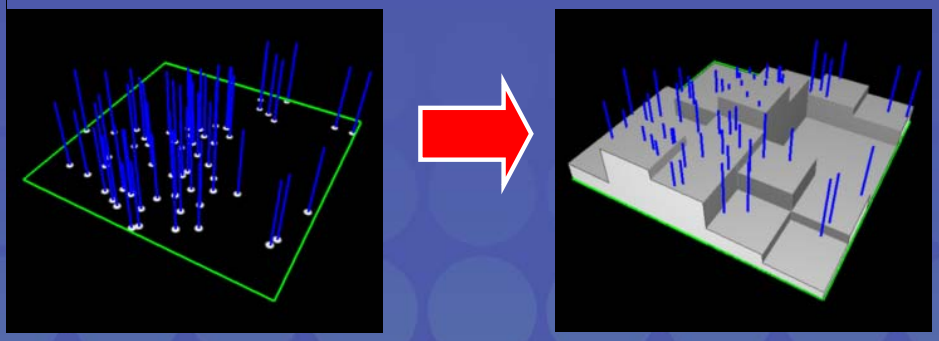
Representation of radiosity in a view independent way requires that scalar product integrals as shown in the top formula box are computed, e.g. the average radiosity on a patch is described by such an integral (next slide). Random walks are a way to sample points x distributed with density proportional with the radiosity B(x). That means we can estimate the scalar product integral, without having to know radiosity. Indeed: "if you can sample it, you don't have to know it". The same was true for the form factors in the stochastic Jacobi radiosity method.

On the next slides, we will illustrate how the two points of view (Monte Carlo integration, and density estimation) are closely related: different density estimation methods are characterized by different measurement functions M(x). We will show the measurement functions for the histogram method, orthogonal series estimation and the kernel method. Also other density estimation methods can be formulated in terms of measurement functions, but the resulting measurement functions are not always practical to use directly. This is the case, for instance, for nearest neighbor density estimation, which lays at the basis of the photon mapping method. (Photon mapping is dealt with extensively in another part of this course.) Still, the measurement function allows theoretical analysis of these methods, as well as may play a role in the development of future methods. See the advanced global illumination book, cited at the end of these slides, for more details on the topic.

Histogram method

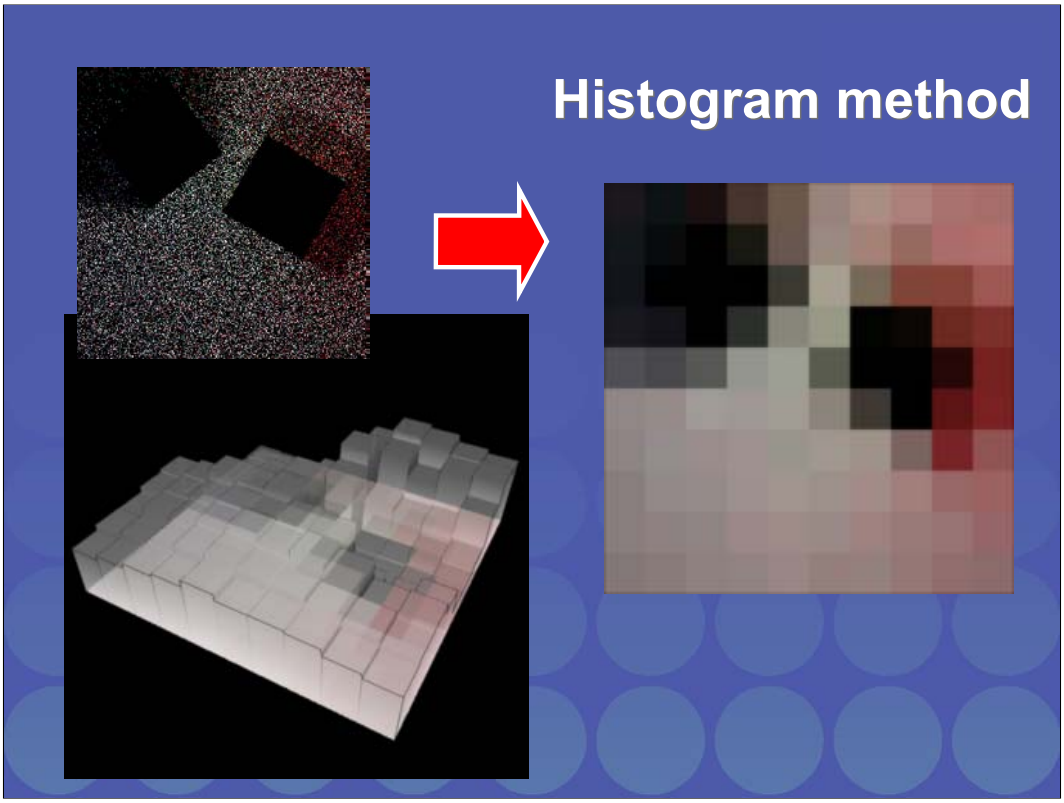
SIGGRAPH2004

- Break surfaces in small elements. Count photons hitting each element:



$$\tilde{B}_i = \frac{1}{A_i}\int_{S_i} B(x)dA_x \quad \Rightarrow \quad M_i(x) = \frac{1}{A_i}\chi_i(x) \quad ; \quad \sum_s M_i(x_s) = \frac{1}{A_i}\sum_s \chi_i(x_s) = \frac{N_i}{A_i}$$

Histogram method: simple and fast and often used.



This slide shows:

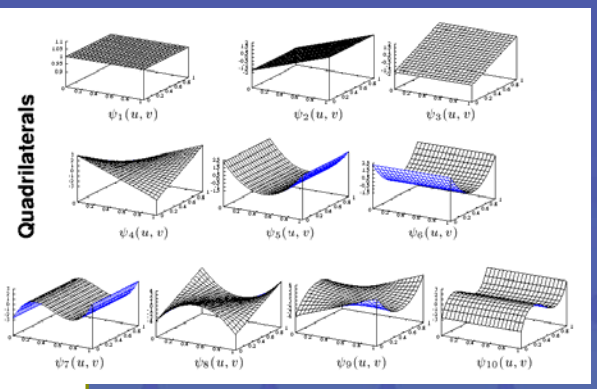
-Top left: the photon hits on the bottom plane of the simple cube like environment of the previous examples

-Bottom left: histogram of these hit points on a 10 by 10 grid

-Right: more familiar visualisation of the histogram

Orthogonal series density estimation

- Linear, bi-linear, quadratic, cubic, … approximations

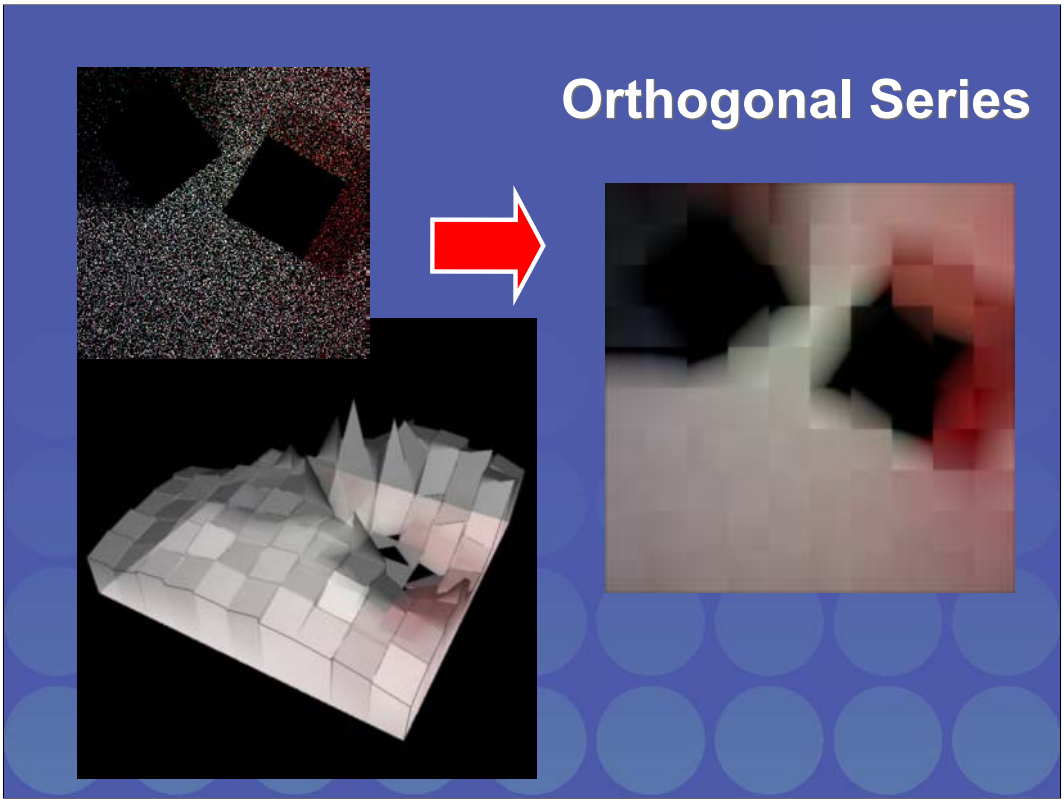


$$\tilde{B}(z) = \sum_{\alpha} B_\alpha \psi_\alpha(z)$$

$$B_\alpha = \int_S B(x)\tilde{\psi}_\alpha(x) dA_x$$

$$\Rightarrow M_z(x) = \sum_{\alpha} \tilde{\psi}_\alpha(x)\psi_\alpha(z)$$

The histogram method can be generalised to non-constant approximations, by introducing basis functions and dual basis functions.

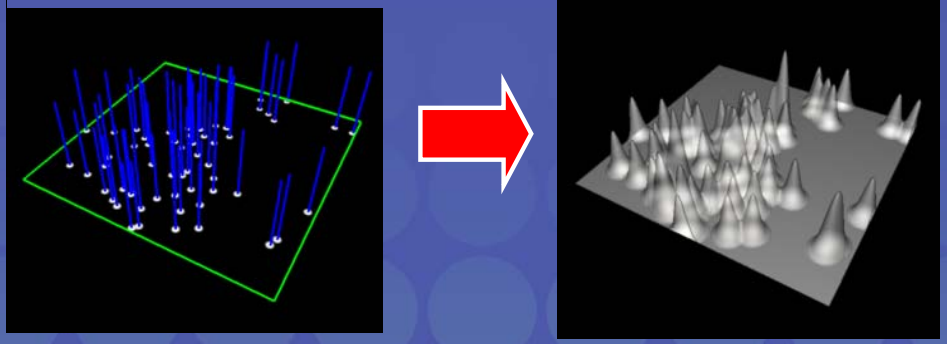


Orthogonal series estimation allows very smooth radiosity solutions in absence of discontinuities. The computation cost is proportional to the number of basis functions taken into account: a bilinear approximation is four times more expensive to calculate than a constant approximation, for instance.

Kernel density estimation

SIGGRAPH2004

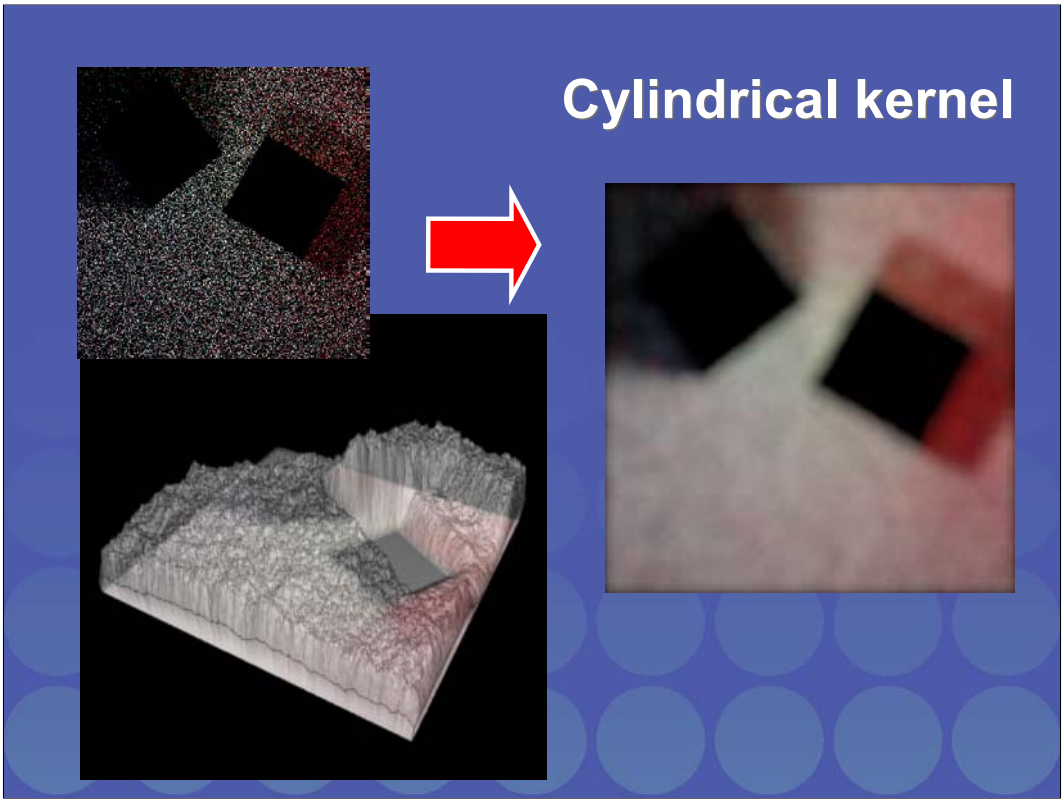
- Place finite-width density kernel at each sample point.



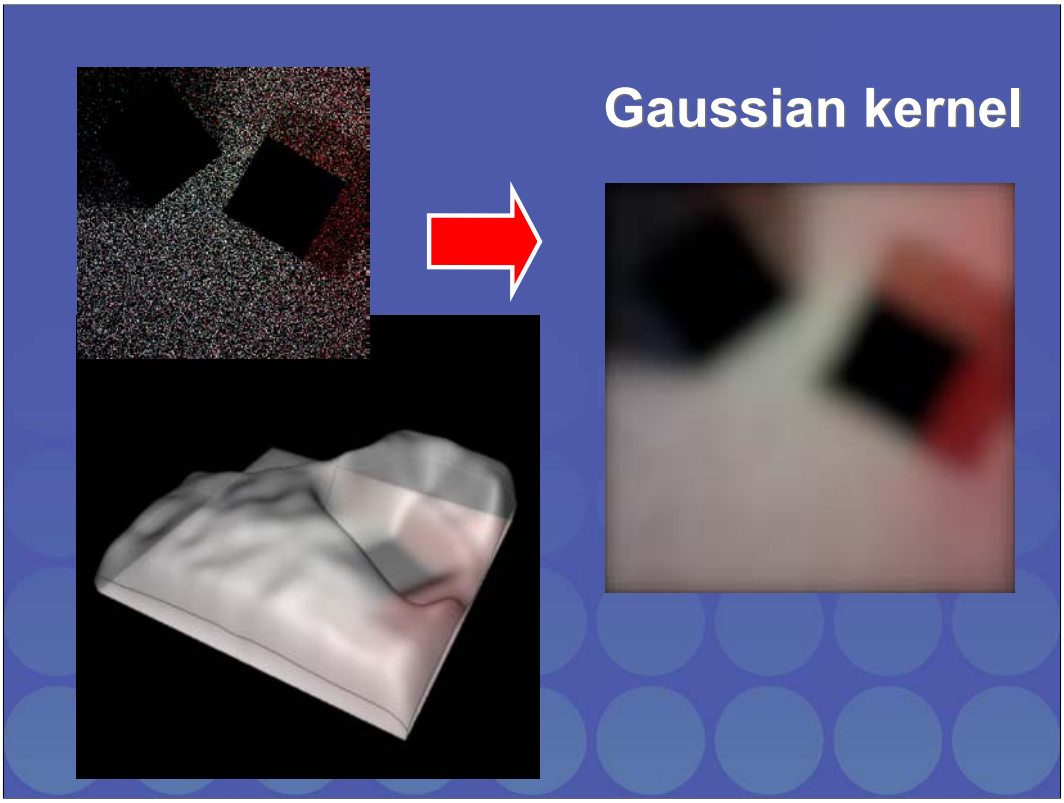
$$\tilde{B}(z) = \int_S K(z-x)B(x)dA_x \quad \Rightarrow \quad M_z(x) = K(z-x) \quad ; \quad \tilde{B}(z) \approx \frac{\Phi_T}{N^{RW}} \sum_s K(z-x_s)$$

Unlike the histogram method and orthogonal series estimation, the kernel density estimation method does not require that surfaces are broken into patches in advance. (You can do so a-posteriori, to get a nicer mesh with properly resolved shadow boundaries, for instance).

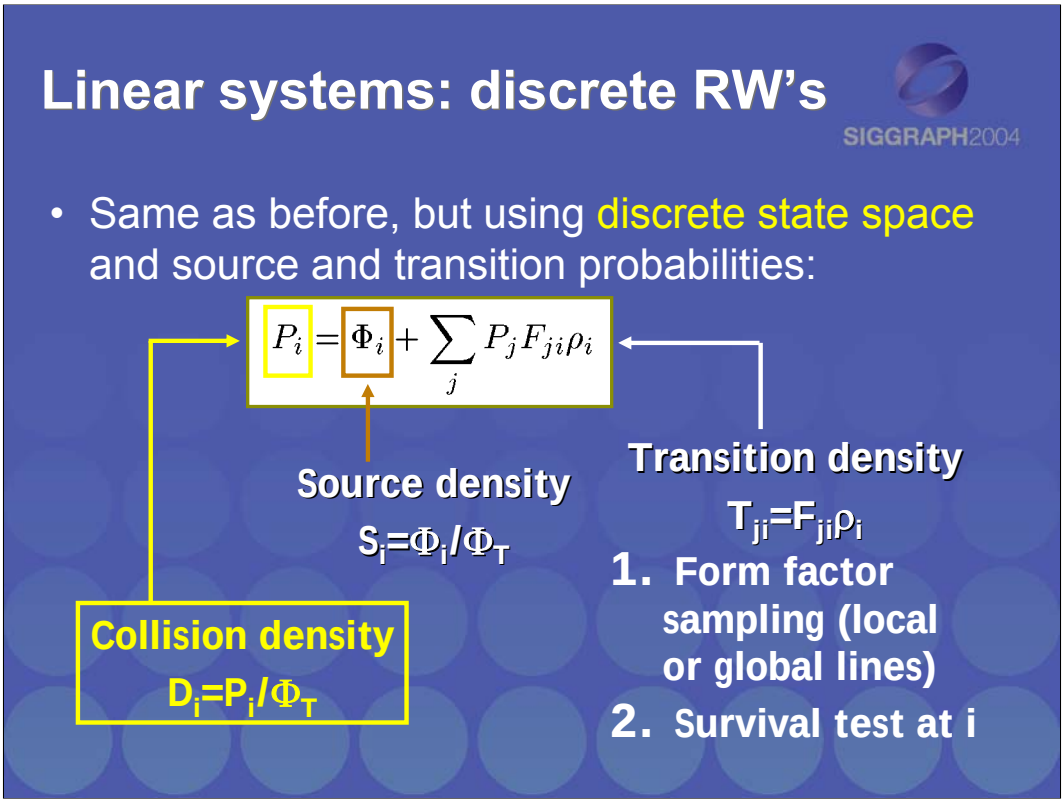
The idea is to place a fixed kernel function, such as a normalized Gaussian of given width, at each hit point. The sum of the kernels, at any location, yields a representation of radiosity. The measurement function of the kernel method, is equal to the kernel.



Example with cylindrical kernels (cheap kernel function).

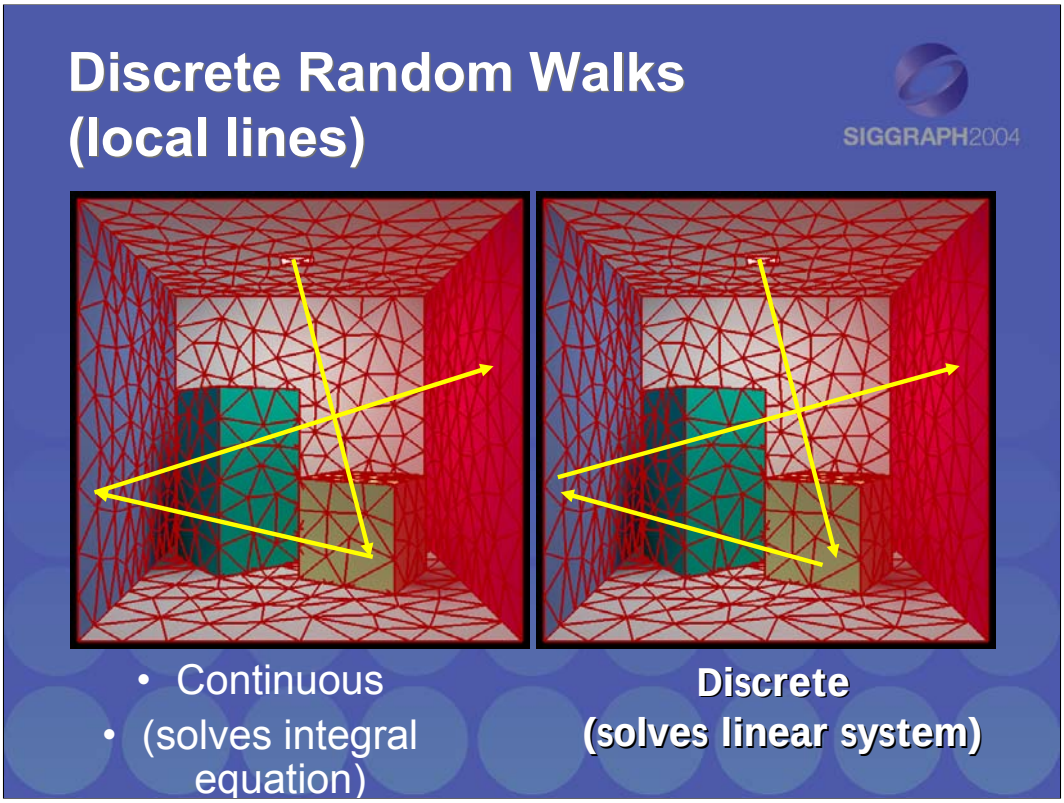


Gaussian kernels of the same with as the cylindrical kernels in the previous example. Note that the solution is much smoother. But Gaussians are also more costly to evaluate.



We now turn back to the radiosity system of equations.

Indeed: Random walks can also be used to solve linear systems, such as the radiosity or power system of equation.



The difference between continuous and discrete random walks is small in practice:

-Continuous random walk: particle reflects of the location of incidence

-Discrete random walk: particle reflects of a random, uniformly chosen, location on the hit patch.

Ad-joint systems: gathering RW's

- With each scalar product <V,B> with TB=E corresponds a scalar product <I,E> with I the solution of an ad-joint equation T*I=V:

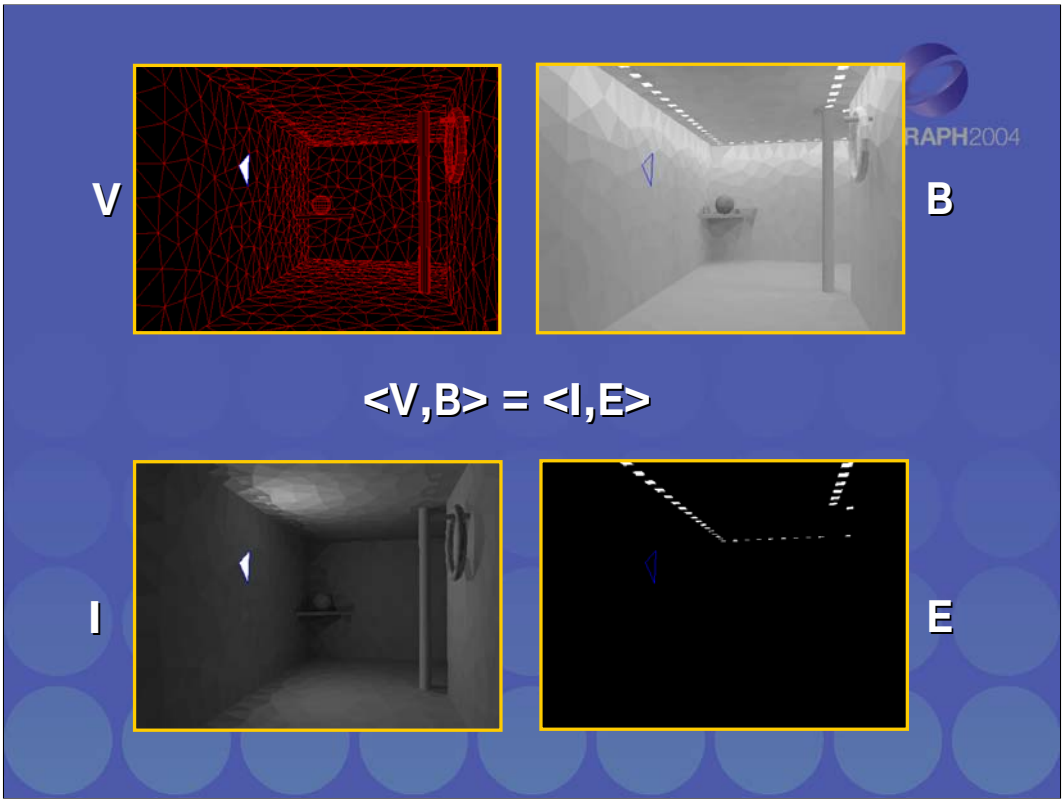
  Proof: <V,B>=<T*I,B>=<I,TB>=<I,E>

- Ad-joint radiosity systems:

$$I_i = V_i + \sum_j F_{ij}\rho_j I_j.$$

- Random walks start at "region of interest" and yield scores when hitting light sources

So far, we have dealt with random walks originating at light sources. It is also possible to compute radiosity by means of random walks that originate at the patches, which we want to compute the radiosity of. Compare it with ray tracing: ray tracing exploits random walks that originate at the observer position. Ray tracing has a dual algorithm, often called light path tracing, that exploits random walks originating at a light source. The random walk radiosity methods describes so far correspond with light tracing and here we describe the ray tracing equivalent. The underlying theory is the same: the theory of dual, or ad-joint, linear systems or integral equations.



Consider the triangle with the blue outline. Suppose we're interested in computing the flux of that triangle. The theorem on the previous slide says that we can do so in two different ways:

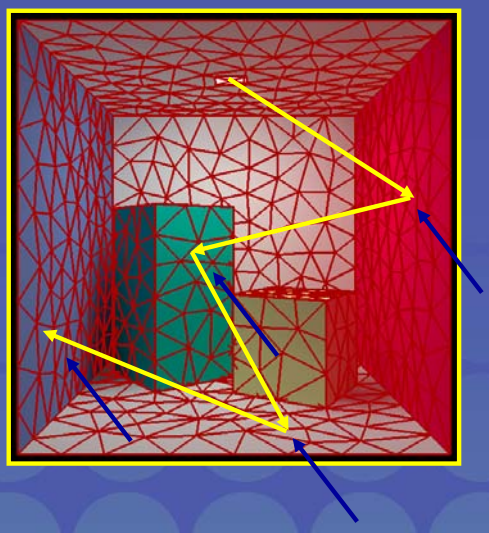
-By calculating the radiosity B in the scene and making a scalar product with the direct importance V of the patch. The direct importance is zero for all patches, except the one under consideration, where it equals 1: flux = area times radiosity. (We absorb multiplication with the patches surface area into our definition of scalar product here.)

-Or we can calculate the total importance I of the light sources in the scene, and make a scalar product with self-emitted radiosity. The importance of each light source tells us to what extent the light source contributes to the illumination of the patch under consideration, either directly or via interreflections: suppose we switch the light source off, how much would the patch under consideration become darker?

Calculating the total importance is a very similar problem as calculating radiosity itself. The idea is to view the patch under consideration as the sole source of some kind of imaginary illumination (the importance or potential) in the scene. Importance propagates through the scene in the reverse way of light.

Since radiosity and importance satisfy such similar equations, similar methods can be used to calculate it. When using random walks, the random walks originate on the sources of importance, thus, the patch under consideration.

Scoring (1)

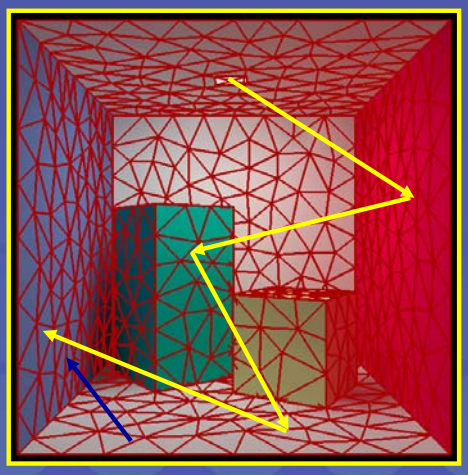


- Collision Estimator: scores everywhere

Here's yet another difference between random walk radiosity algorithms, determined by at what locations we let a random walk contribute a non-zero score. So far, we have discussed random walks that contribute a score at all locations where they hit surfaces.

Scoring (2)

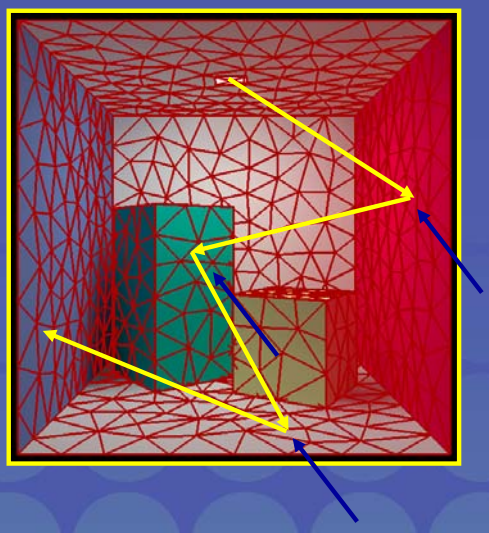
SIGGRAPH2004



- Absorption estimator: scores only where path terminates

Another common type of random walk estimators in literature contributes a score only at the location where the random walk is terminated.

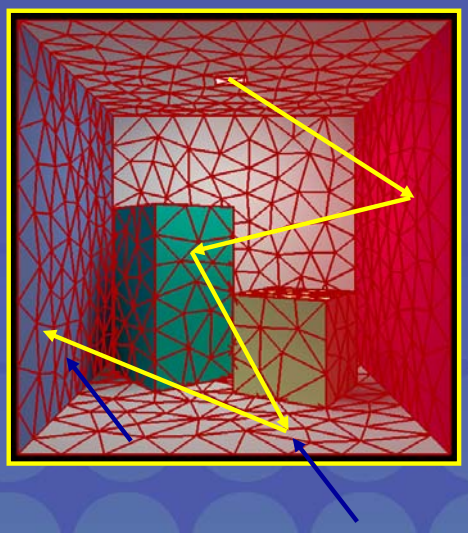
Scoring (3)



- Survival estimator: scores everywhere except when absorbed

Or you can have random walks scoring at all locations where they are not terminated.

Scoring (4)



- Many more
- Possibilities!

And there are many more possibilities indeed.

Comparison

1. Discrete versus continuous RW (histogram):
   Slightly higher discretisation error, but QMC sampling is much more effective for discrete RW.
2. Scoring:
   Collision RW most often preferable
3. Gathering versus shooting:
   Shooting more effective, except on small patches
4. Discrete collision shooting RW versus stochastic iterations:
   Basic estimators equally efficient, but variance reduction more easy and effective for stochastic iterations.

The question now is how all these random walk strategies compare with each other, and with the stochastic Jacobi method. Here's just a very short summary. Details can be found in the advanced global illumination book cited at the end of these slides.

Efficiency improvements:

1. View-importance driven sampling
2. Control variates
3. Combining gathering and shooting
4. Weighted importance sampling
5. Metropolis sampling
6. Low-discrepancy sampling

- Topic of ongoing research!
- Our experience: often easier and more effective for stochastic iterative methods

We now turn to the one but last topic of this course section: techniques to improve the efficiency of the Monte Carlo approaches discussed so far. There are many such techniques, too many to discuss all of them here. Many of these techniques are quite straightforward applications of variance reduction techniques described in basic Monte Carlo text books such as Kalos and Whitlock (ref. at the end of these slides). Other's aren't that obvious. Not all possibilities have been fully explored yet.

View-Importance driven sampling.

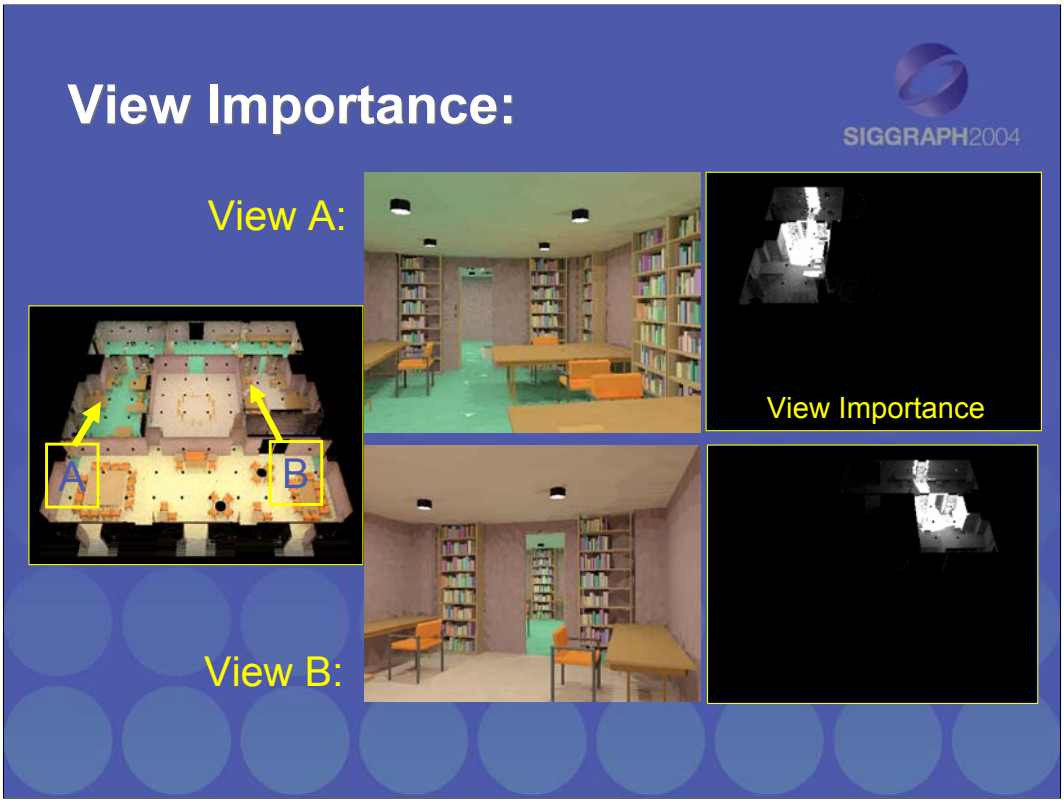
- Goal: focus computations on “important” parts of a scene
- Measure for “importance”:
  - Solve adjoint radiosity equations:

$$I_i = V_i + \sum_j F_{ij}\rho_j I_j.$$

  - Use this “importance” in order to shoot more rays originating at/towards important regions

We will give just one particular, but useful and interesting, example.

We already discussed the ad-joint radiosity equations in order to explain and introduce gathering random walk methods for radiosity. With a slightly different definition of direct view importance V, the ad-joint system also allows to derive algorithms that focus computation effort on a part of a scene to be rendered. For instance, suppose you want to render a room in a complex multi-floor building with many rooms on each floor. Usually, the illumination perceived in a room will hardly depend on the illumination in other floors, as well as many other rooms on the same floor. We would proceed by assigning a high direct view importance to the part of the room visible in a view. The solution of the adjoint radiosity equation then tells us to what extent the illumination elsewhere in the building would contribute to the flux emitted by the directly visible surfaces in our view. We can use that knowledge in order to tune our stochastic Jacobi sampling probabilities, or to tune our source, survival and transition random walk probabilities. Calculating the importance I is however a similar problem as calculating radiosity. In short, we do some work twice, in order to avoid a lot of unnecessary work.



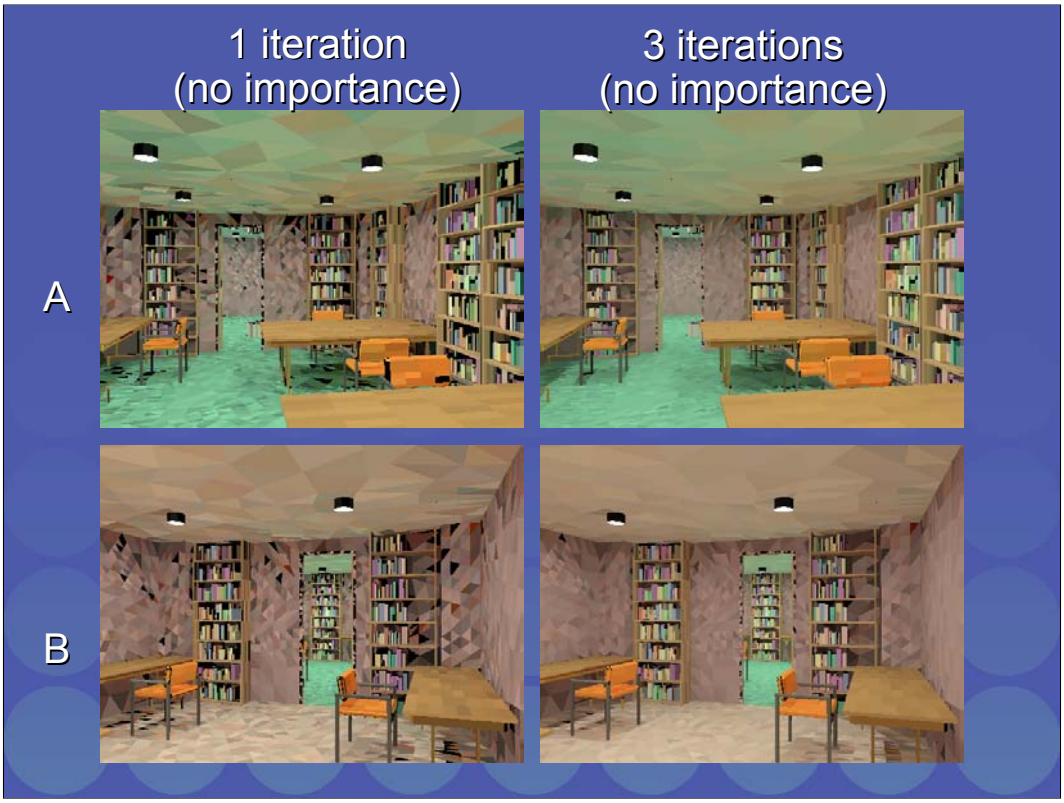
Here's an overview of an example scene, with two views A and B. The right column shows the importance corresponding to each view. A bright surface has high influence on the illumination perceived in a view. A dark surface hardly has any impact.

View-importance driven stochastic iterations

$$P_i I_i = \Phi_i I_i + \sum_j P_j I_j F_{ji} \rho_i \frac{I_i}{I_j}$$

- A: Sample more rays **originating** at patches with high importance
- B: Sample more rays **originating** at patches with high **indirect** importance
- C: B + also aim rays **towards** regions with high indirect importance

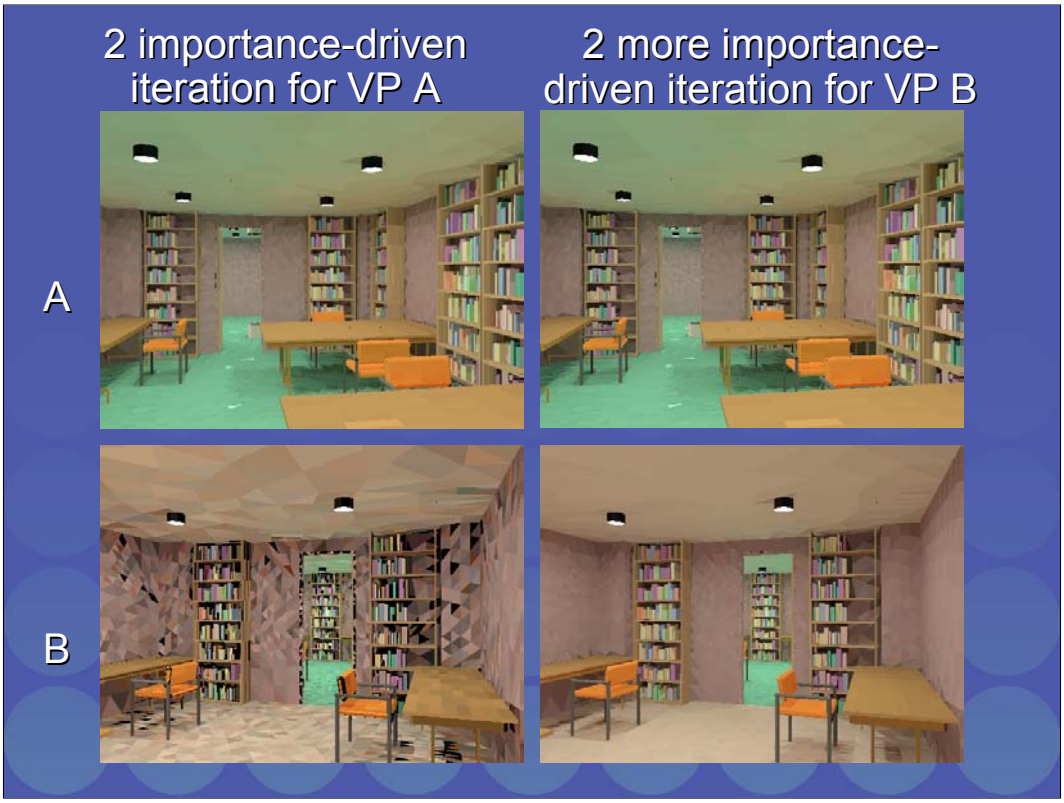
We can derive several view-importance driven stochastic Jacobi algorithms by modifying our power system. For instance, we multiply left and right with importance, and interpret the resulting factors in a similar way for sampling, as sketched before. Method B in this list, is the most practical method. Method C would be better, if we had better methods for sampling the required transitions, which are no longer according to the form factor only. It's one example where further research would be beneficial.



Here are some results.

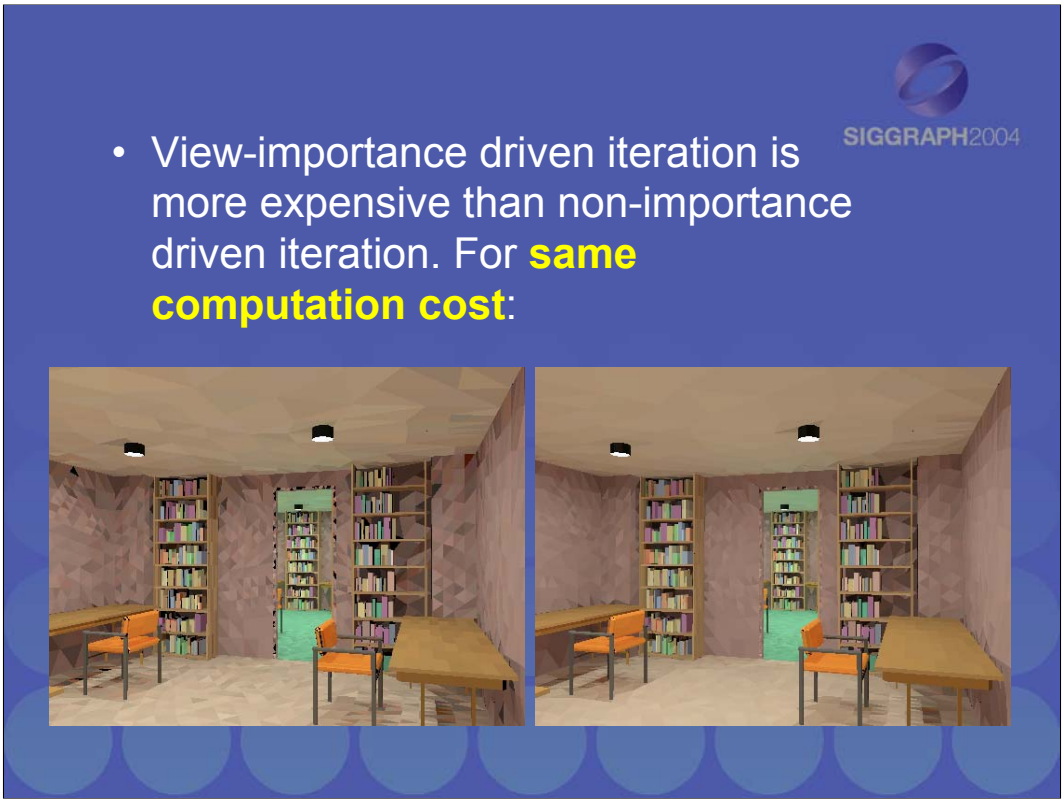
If you don’t use view importance sampling, the quality of the solution improves in a uniform way throughout the scene to be rendered.

(The model shown is part of the Berkeley SODA hall building model, available from the university of California at Berkeley.)



The left column shows what happens after two view-importance iterations for view A: noise in view A is rapidly reduced, while the quality in other locations (e.g. view B on the bottom) doesn't improve at all: we focused our computation efforts on view A.

The right column shows what happens if we next move to view point B and further refine the radiosity solution using the importance solution for B instead of A: view B improves quickly, in turn. By averaging partial solutions in a proper way, the quality of view A does not degrade.



View importance driven iterations are however typically about twice as expensive as non-view importance driven iterations, in largest part because importance needs to be propagated in addition to radiosity. Even for the same computation cost, however, view importance driven sampling clearly is better in this example.

Hierarchical Refinement

SIGGRAPH2004

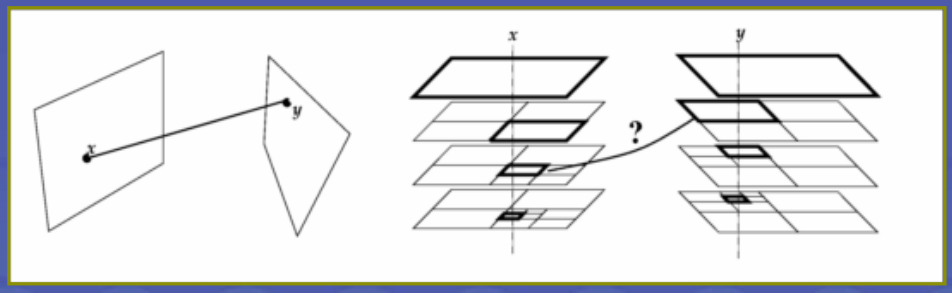
- Problem:
  - Noisy artifacts on small patches: group them!
  - Large patches are too “flat”: subdivide them!

So far, we’ve dealt with algorithms that avoid explicit form factor computation and storage in radiosity. Form factor computation and storage was however but one of the two problems of the radiosity method discussed at the beginning of this course section. What about the discretisation error? We will discuss how to incorporate hierarchical meshing in stochastic Jacobi radiosity here, as a means to reduce discretisation artefacts to a certain extent.

Per-ray refinement

SIGGRAPH2004

- Principle:

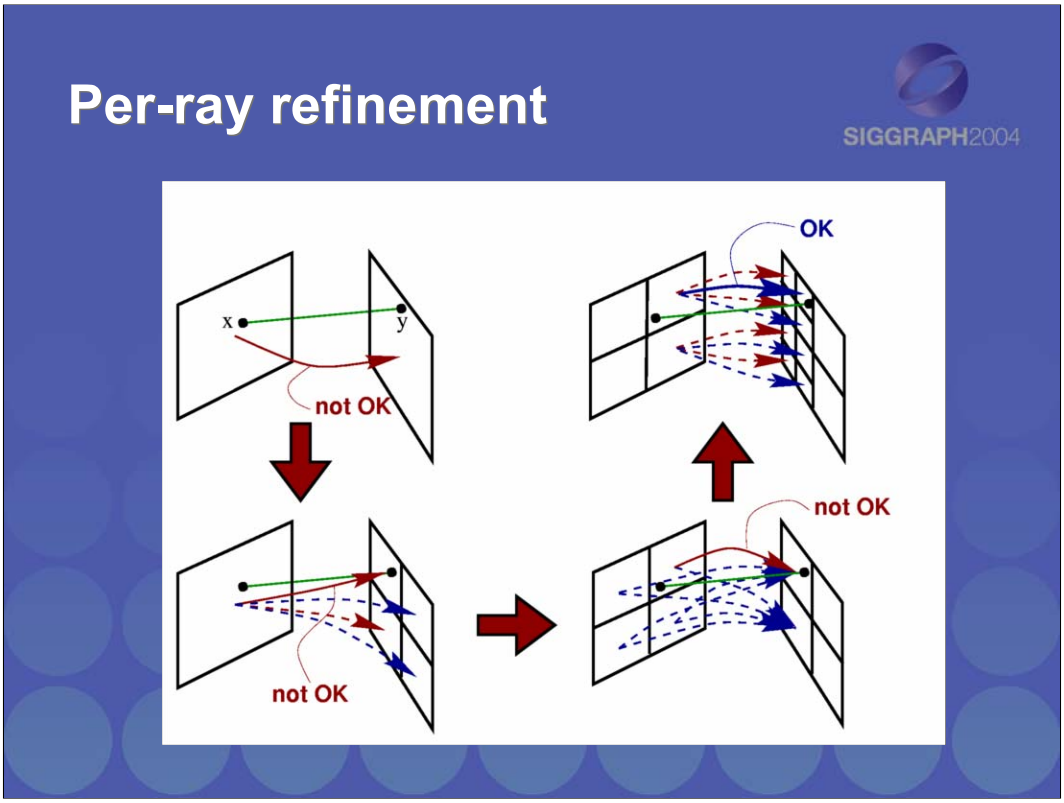


- Related with Monte Carlo solution of wavelet-preconditioned linear systems.

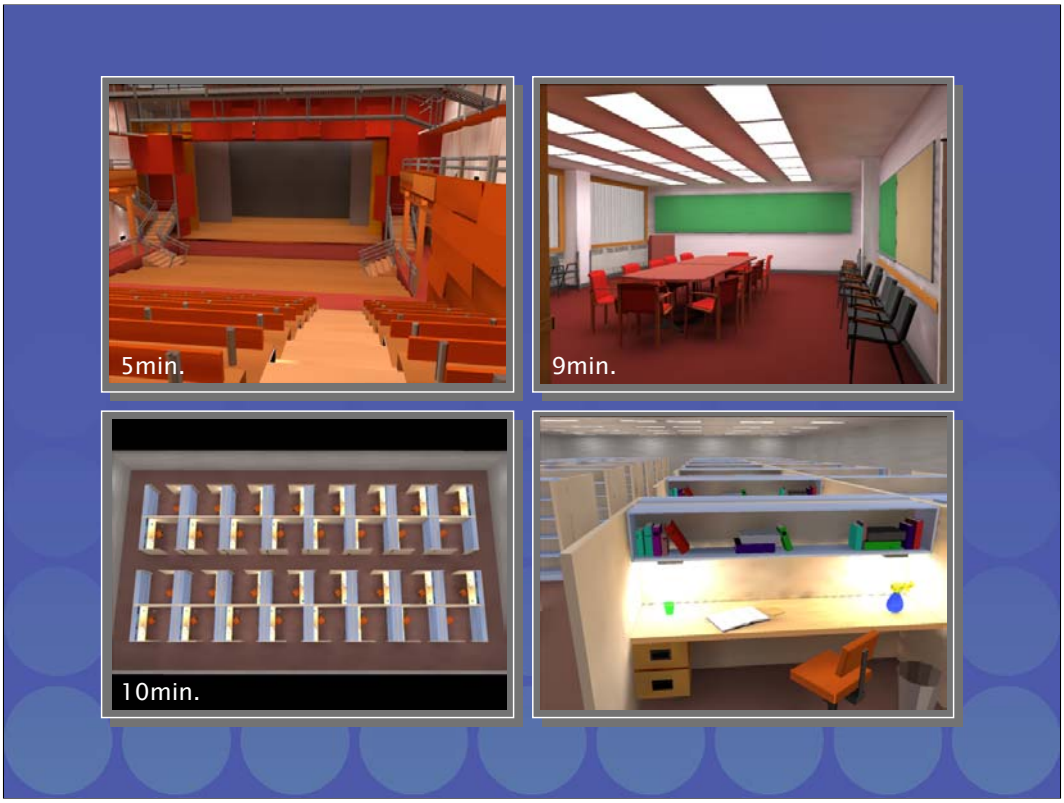
The basic idea is as follows: in all non-hierarchical radiosity algorithms described so far, rays were traced from one surface location to another. The rays carried some amount of power from their origin to their destination. The power was deposited and distributed over the patch that contained the destination point. The key idea of per-ray refinement is to subdivide or group patches on the fly, in order to deposit the power quantum carried by the rays on a more appropriate surface area. We can do that in a similar way as in deterministic hierarchical radiosity, by means of a

-oracle function that predict whether or not a candidate link between two given surface or cluster elements would allow sufficiently precise light transport

-Refinement strategy that subdivides either the source of destination element in order to obtain new candidate sub-links whenever the oracle deems a candidate link not appropriate.



Per ray refinement is illustrated here: we start out with some top-level element pair containing the ray end-points x and y. Without clustering, we would take the model polygons as they are input to the radiosity system. With clustering, the top-level elements would correspond to a single cluster element that contains the whole model. If the oracle deems a candidate link between the given element pair as inappropriate, one of both elements will be subdivided in smaller sub-elements. In deterministic hierarchical radiosity, each of the candidate sub-links resulting from that subdivision would be examined in turn by the oracle function, and recursively subdivided if necessary. In per-ray refinement, we will subject only a single candidate sub-link to the oracle function: the one with the sub-element that contains the end-point of the ray. We continue to do so until the oracle deems the candidate OK. The power carried by the ray is deposited on the resulting destination sub-element. The links do not need to be stored in memory, resulting in a major storage requirement reduction.



Here are some results. The models contain between 80,000 (top left) en 500,000 (bottom) elements after refinement. The timings are obtained on a good old 200MHz SGI machine. The bottom model takes less than one minute to render on my 1.6GHz laptop computer nowadays. A decent, production quality, implementation would do perhaps 5 times faster.

Here are some more examples. The bottom right one is a radiosity rendering of the so called Berkeley SODA hall model, available from www.cs.berkeley.edu . It is a 7-floor building model, each with a large number of rooms, and fully furnished with desks, flowers, pens, books and canary birds. The radiosity rendering contains 8 million elements after refinement. It took 1 night to render on a 400 MHz Onyx 2 machine with 2GB of RAM. Once computed, the radiosity solution can be viewed interactively using graphics hardware or using a fast ray casting engine. The upper left model, as well as the models shown on the previous slide are by Greg Ward and collaborators at UC Berkeley and Laurence Berkeley Labs.

Summary

- Radiosity measurement
- Solution by means of stochastic iteration or random walks (= point sampling)
- Many kinds of random walks:
  - Continuous versus discrete
  - Shooting versus gathering (adjoint equation)
  - Scoring: all collisions, at absorption, survival, …
- Plenty of ways to improve efficiency
- Viable alternative for deterministic methods

And that's basically it.

More information

- Related books:
  - Kalos and Whitlock, The Monte Carlo Method, 1986
  - Silverman, Density Estimation for Statistics and Data Analysis, 1986
  - Dutre, Bekaert, Bala, Advanced Global Illumination, A.K.Peters
- RenderPark: test bed system for global illumination
  - www.renderpark.be

Here's a minimal reading list for those who want to know more.

The renderings on these slides have all been done with RenderPark.

Interactive Global Illumination

SIGGRAPH 2004 Course

Kavita Bala
Cornell University

Overview

- Fast Ray Tracing
- Reusing shading samples for GI

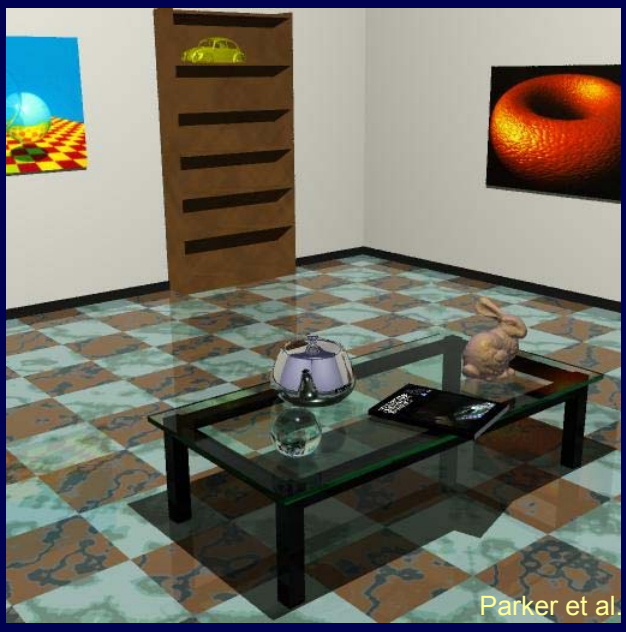
Motivation

- Ray casting well-suited for global illumination
  - Flexible
  - Efficient for large models when using an acceleration structure (grids, bsp, etc)
  - Parallelizable
- But, usually the largest computational bottleneck

Interactive RT [Parker et al.]

- SGI Origin 2000
  - 64 processors
  - Shared memory
- Whitted-style ray tracing
  - Shadow, reflection, and refraction rays
- Non-polygonal primitives
  - Spheres and splines

15 fps



4 fps

Coherent RT [Wald et. al.]

- Create highly optimized ray tracing engine for Intel-based PCs
  - Hand-crafted and tuned
  - Discovered ray casting is memory bound
    - Need better memory coherence
    - Created compact and cache friendly data structures
    - Optimized for SIMD (SSE)

Test Scenes I

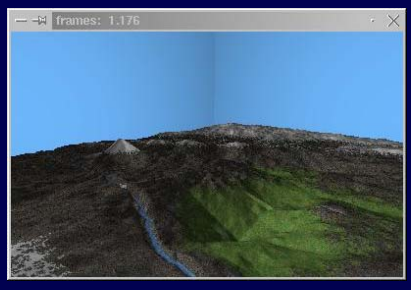
- Test scenes: 800 triangles to >8 million

Office:
34,000 triangles, 3 lights

Conference Room:
280,000 triangles, 2 lights

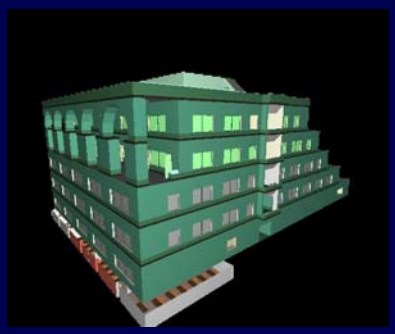
Test Scenes II

- Test scenes: 800 triangles to >8 million



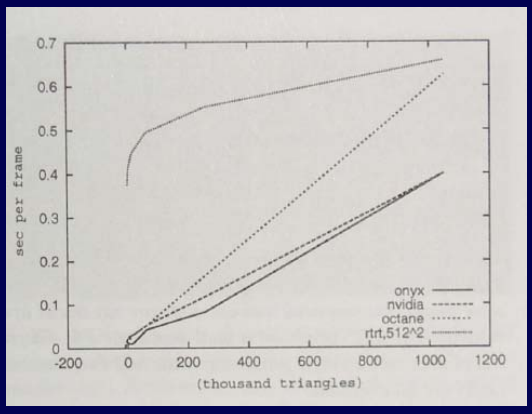
Terrain:
1 million triangles
(textured)

Berkeley Soda Hall:
1.5 to 8 million triangles



Performance Results III

- Comparison to Rasterization-Hardware
  - Ray tracing scales well for large environments

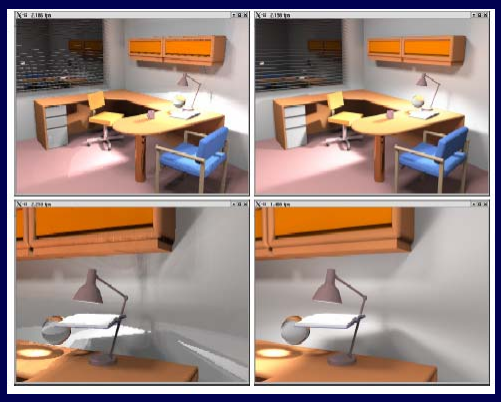


Recent Results

- Billion polygons, uses instantiation

Applied to GI[Wald et al.]

- Use instant radiosity and caustic photon maps with the fast ray tracing engine
- Interleaved sampling and filtering



Ray Tracing on GPUs [Purcell]

- Map ray tracer onto GPU
- Multi-pass
- Abstract programmable fragment processor as a stream processor

Fast RT

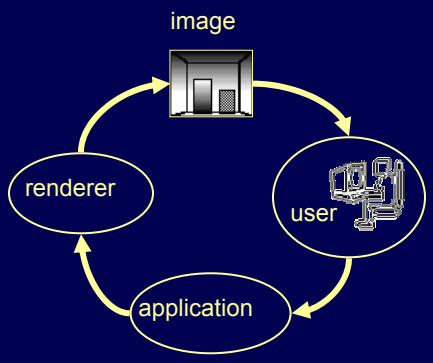
- Impressive performance
- Still too slow to compute global illumination

Reusing Shading

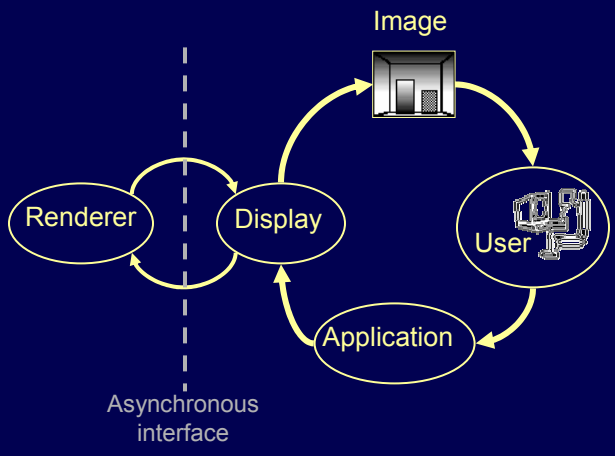
- Full simulation of GI can take minutes to hours
- Reuse shading to achieve interactive performance
  - Exploit spatial and temporal coherence if possible

Standard Display Loop

- Synchronous



Modified Display Loop



Approaches

- Image based
  - Post-rendering Warp *(Mark97)*
  - Corrective texturing *(Stamminger00)*
- Point based
  - Render Cache *(Walter99,02)*
  - Edge-and-Point Rendering (Bala03)
- 4D
  - Radiance Interpolants (Bala99)
  - Holodeck (Ward99)
- Mesh based
  - Tapestry *(Simmons00)*
  - Shading Cache *(Tole02)*

Frameless Rendering

- Update pixels as they are computed
  - Don't wait for full frame to finish

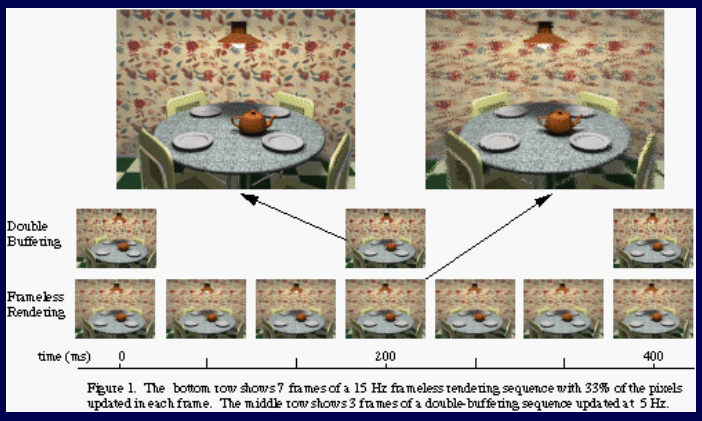


Figure 1. The bottom row shows 7 frames of a 15 Hz frameless rendering sequence with 33% of the pixels updated in each frame. The middle row shows 3 frames of a double-buffering sequence updated at 5 Hz.

Post-Rendering 3D Warp

- A subset of frames are rendered
- Image warping computes intermediate frames
- Fill in holes and missing data from past and future reference frames

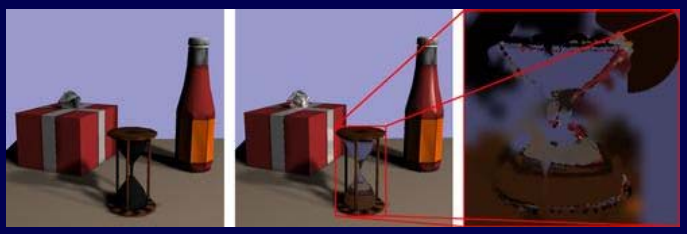
Reference frame Warped frame

Post-Rendering 3D Warp

Prior reference Warped frame Next reference

Corrective Texturing

- Exploit hardware rendering capabilities
- Use radiosity solution
- Do corrective texturing where radiosity solution differs from samples



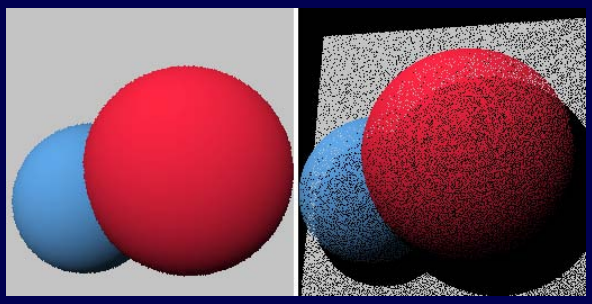
Radiosity solution | Corrected image | Corrective texture

Corrective Texturing

- Sparse shading samples compared to hardware results
  - Splat differences into textures
  - More samples where larger differences

Render Cache

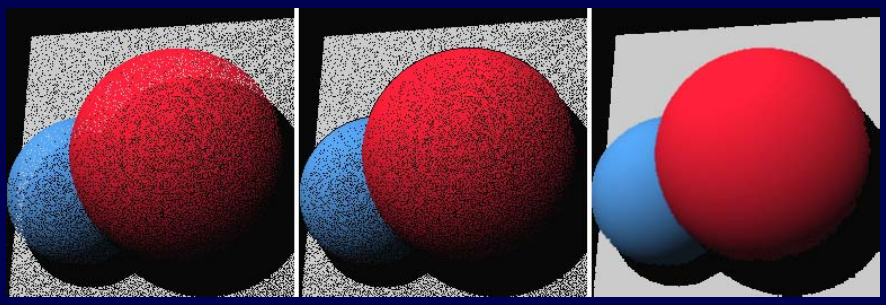
- Results stored as cloud of unordered points with: 3D position, color, age, etc.
- Points reprojected from frame-to-frame



Initial view After reprojection

Render Cache

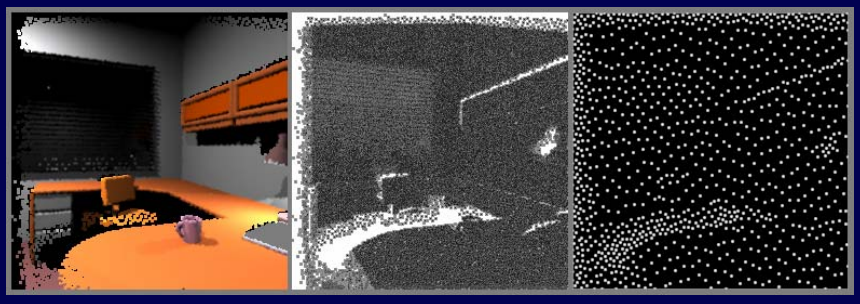
- Use occlusion culling heuristic
- Interpolation to fill holes
  - Fixed size kernels: 3x3 and 7x7



Reprojection Occlusion cull Interpolation

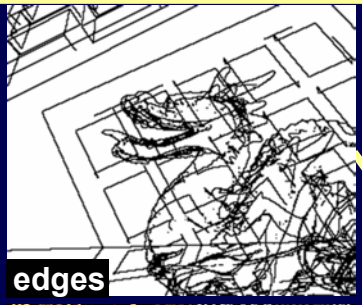
Render Cache

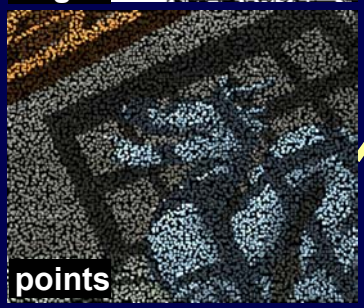
- Priority image for sampling
  - High priority for old points, sparse regions
- Error-diffusion dither on priority image determines sample locations



Displayed image Priority image Requested pixels

Edge-and-Point Rendering

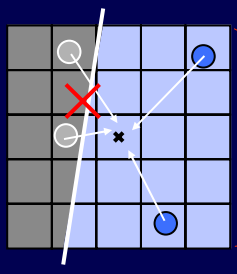




- Edges: important discontinuities
  - Silhouettes and shadows
- Points: sparse shading samples

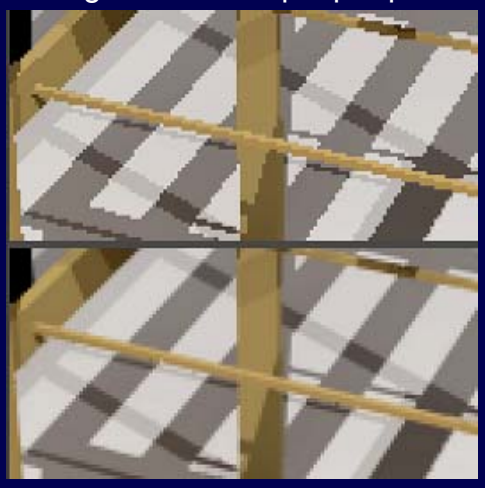
Edge-and-Point Image

- Alternative display representation
- Edge-constrained interpolation preserves sharp features
- Fast anti-aliasing



Fast Anti-aliasing

Magnified view of a ray traced image with 1 sample per pixel

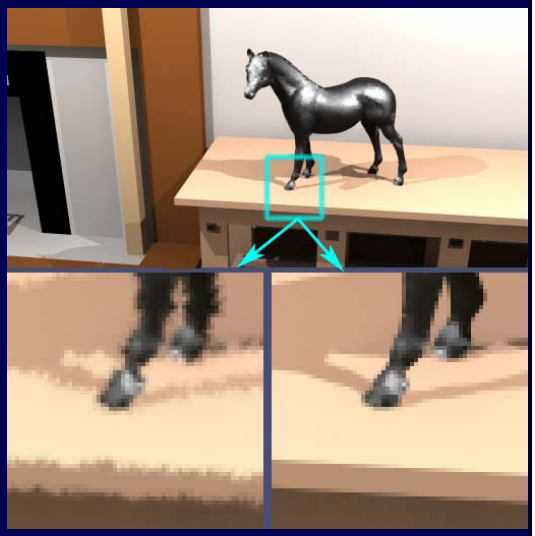


Result using <1 sample per pixel

Magnified view of E&P renderer

Results: Quality

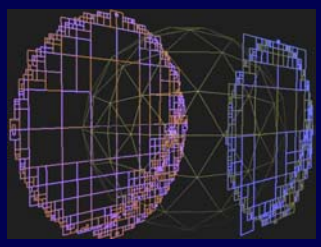
- Global Illumination
  - 3 lights
  - 150k polygons
- Sparseness Ratio
  - 100: 1
- Performance
  - 8-14 fps



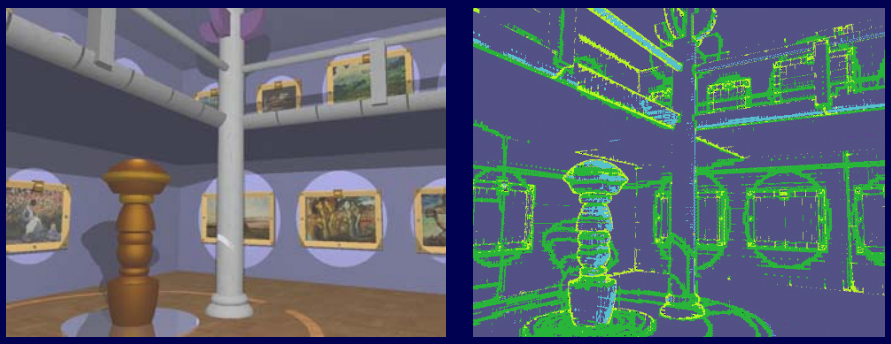
Without Edges With Edges

Radiance Interpolants Bala[96,99]

- Radiance interpolant
  - Set of sparse samples (in 4D line space) of radiance function
  - 4D linetree stores samples
  - Bounded-error, accurate reconstruction



Results: Museum Scene



gray: interpolation success
yellow: silhouettes; green: shadows

Holodeck [Ward et al.]

- 4D samples are lazily evaluated and reused
  - Radiance used to compute samples
- Shading samples stored in 4D data structure
  - Similar to Light Field or Lumigraph
  - Paged to disk if necessary
- Display uses hardware for Gouraud-shaded triangle mesh
  - Lookup samples near current viewpoint
  - Samples become vertices in a mesh
  - Delaunay triangulation of samples in direction space

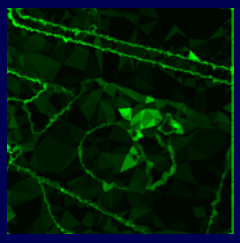
Tapestry [Simmons et al.]

- Based on Holodeck system
- With enhancements:
  - Prioritized sampling
  - Incremental “recentering” of spherical Delaunay mesh as viewpoint moves
  - Fixed cache size: max vertices = pixels
  - Sample invalidation: occlusion and color change heuristics

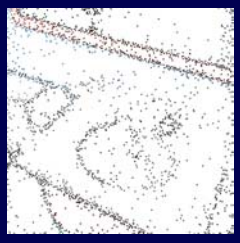
Tapestry

- Triangle priorities use
  - Color differences, depth differences and age
  - Hardware rasterizes priority
  - Quasi-random sampling with rejection

Image



Priority

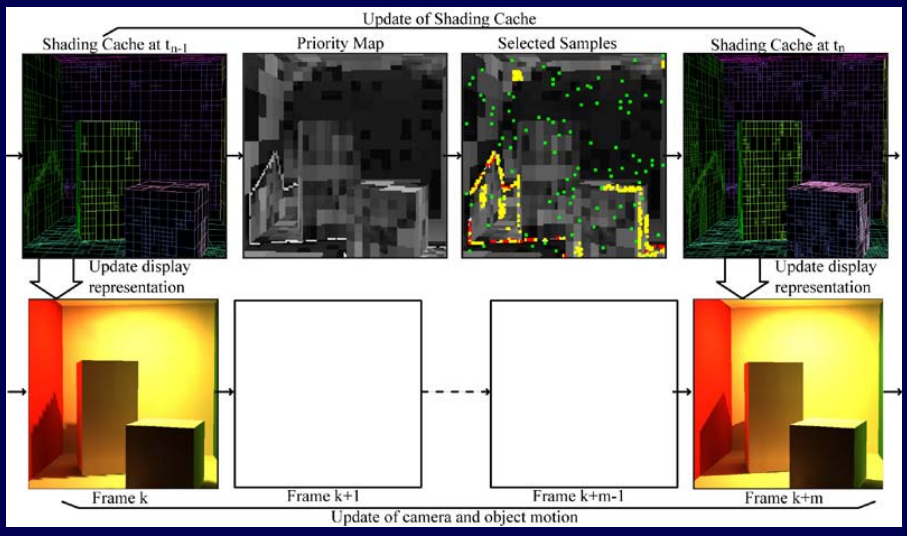


Samples

Shading Cache [Tole et al.]

- Sparse samples combined with display mesh
- Use hardware to render mesh and handle textures
- Display mesh is refinement of original scene mesh
  - No occlusion errors
  - Display mesh >= original mesh
- Decouples frame update from mesh update
  - Mesh refinement over multiple frames
- Heuristic to determine where to sample
  - Random sampling and flood filling
  - More samples near discontinuities

Shading Cache



Conclusions

- Ray tracing scales well with scene complexity
- Reusing shading for global illumination
  - Interactive feedback
  - Allows use of flexible, slow shaders
  - Various promising approaches
- Fast ray tracing and reusing shading could be useful tools

Bibliography

- Bala, Dorsey, Teller: **Radiance Interpolants for Accelerated Bounded-Error Ray Tracing**, TOG 1999
- Bala, Walter, **Reusing Shading for Global Illumination**, Game Developer's Conference, April 2004
- Bala, Walter, Greenberg, **Combining edges and points for Interactive high quality rendering**, SIGGRAPH 2003
- Parker, Martin, Sloan, Shirley, Smits, Hansen, **Interactive Ray Tracing**, Symposium on Interactive 3D Graphics, April 1999
- Simmons, Séquin, **Tapestry: A Dynamic Mesh-based Display Representation for Interactive Rendering**, Rendering Techniques 2000

Bibliography

- Stamminger, Haber, Schirmacher, Seidel, **Walthroughs with Corrective Texturing**, Rendering Techniques 2000
- Wald, Benthin, Wagner, Slusallek, **Interactive Rendering with Coherent Ray Tracing**, Proceedings of Eurographics 2001
- Wald, Kollig, Benthin, Keller, Slusallek, **Interactive Global Illumination using Fast Ray Tracing**, Rendering Techniques 2002
- Walter, Drettakis, Parker, **Interactive Rendering using the Render Cache**, Rendering Techniques 1999
- Walter, **GI for Interactive Applications and High-Quality Animations**, SIGGRAPH 2003 course notes
- Ward, Simmons, **Holodeck ray cache**, TOG 1999