Practical Global Illumination with Irradiance Caching

SIGGRAPH 2008 Class

Organizers

Jaroslav Křivánek
Czech Technical University in Prague

Pascal Gautron
Thomson Corporate Research France

Lecturers

Greg Ward
Dolby Canada

Henrik Wann Jensen
University of California, San Diego

Eric Tabellion
PDI/DreamWorks

Per H. Christensen
Pixar Animation Studios

Abstract

Since its invention 20 years ago, irradiance caching has been successfully used to accelerate global illumination computation in the *Radiance* lighting simulation system. Its widespread use had to wait until computers became fast enough to consider global illumination in production rendering. Since then, its use is ubiquitous. Virtually all commercial and open-source rendering software base the global illumination computation upon irradiance caching. Although elegant and powerful, the algorithm often fails to produce artifact-free images. Unfortunately, practical information on implementing the algorithm is scarce.

The objective of the class is twofold. The first and main objective is to expose the irradiance caching algorithm along with all the details and tricks upon which the success of its practical implementation is dependent. Various image artifacts that the basic algorithm can produce will be shown along with a recipe to suppress them. We will also put strong emphasis on practical aspects of irradiance caching integration in production environments and discuss the particularities used in two big production houses, namely PDI/DreamWorks and Pixar.

The second objective is to acquaint the audience with the recent research results that increase the speed and extend the functionality of basic irradiance caching. Those include: exploiting temporal coherence to suppress temporal flickering; extending the caching mechanism to rendering glossy surfaces; accelerating the algorithm by porting it to the GPU. Advantages and disadvantages of those methods will be discussed.

Prerequisites

A basic understanding of rendering, ray tracing in particular, is expected. Familiarity with global illumination concepts is useful.

Level of difficulty

Intermediate

Intended audience

Anyone interested in making a production-ready implementation of irradiance caching that reliably renders artifact-free images. Researchers and industrial developers interested in recent speed and quality improvements of global illumination with irradiance caching.

Syllabus

1. Introduction (*Křivánek*) 12
(8:30 – 8:35 / 5 min)
    - What is global illumination, why compute it?
    - Overview

2. Monte Carlo Ray Tracing (*Křivánek*) 21
(8:35 – 8:55 / 20 min)
    - Diffuse interreflections
    - Gathering approach to computing diffuse interreflections
    - Final gathering for photon mapping

3. Irradiance Caching Algorithm (*Ward*) 46
(8:55 – 9:20 / 25 min)
    - Spatial coherence of indirect illumination
    - Caching strategy
    - Weighting function
    - Data structure
    - Irradiance Gradients

4. Irradiance Caching in RADIANCE (*Ward*) 74
(9:20 – 9:35 / 15 min)
    - Computation of ambient "constant"
    - Adaptive super-sampling on hemisphere
    - Maximum and minimum record spacing
    - Gradient limit on record spacing
    - Bump maps using rotation gradient
    - Options for excluding surfaces/materials
    - Record sharing for multiprocessors

5. Problems & Solutions: Implementation Details (*Křivánek*) 85
(9:35 – 10:00 / 25 min)
    - Improved cache data structure
    - Image-space caps on record spacing
    - Removing artifacts due to missing small geometry & leaking rays: the neighbor clamping heuristic
    - Weighting function revisited
    - Image sampling order, two-pass rendering
    - Motion blur

6. Irradiance Caching and Photon Maps (*Jensen*) 137
(10:00 – 10:15 / 15 min)

Break
(10:15 – 10:30 / 15 min)

7. Extension to Glossy Surfaces: Radiance Caching (*Křivánek*)....................................... 174
(10:30 – 10:45 / 15 min)
- Indirect illumination on glossy surfaces
- Incoming radiance representation, spherical harmonics
- Radiance interpolation
- Adaptive radiance caching

8. Hardware Implementation (*Gautron*).. 223
(10:45 – 11:05 / 20 min)
- Reformulation for efficient GPU implementation
- Irradiance splatting
- GPU-Based hemisphere sampling

9. Temporal Caching (*Gautron*) 284
(11:05 – 11:20 / 15 min)
- Flickering in animation
- Temporal coherence of indirect lighting
- Temporal weighting function and gradients

10. Irradiance Caching at PDI/DreamWorks (*Tabellion*)... 333
(11:20 – 11:55 / 35 min)
- Global Illumination at PDI/DreamWorks
- IC integration
- Practical problems with irradiance caching: proprietary solutions
- Fur and Grass
- Irradiance caching files and tools

11. Irradiance Caching in Pixar's RenderMan (*Christensen*) 380
(11:55 – 12:15 / 15 min)
- Global illumination in RenderMan
- Irradiance caching in the REYES image rendering architecture
- Occlusion gradients for ambient occlusion

12. Discussion (*All*)
(12:15 – 12:30 / 15 min)

Class presenter information

***Jaroslav Křivánek**, Czech Technical University in Prague*
xkrivanj@fel.cvut.cz
Jaroslav Křivánek is an assistant professor at the Czech Technical University in Prague. He received his Ph.D. from IRISA/INRIA Rennes and the Czech Technical University (joint degree) in 2005 for extending the irradiance caching algorithm to rendering global illumination on glossy surfaces. In 2003 and 2004 he was a research associate at the University of Central Florida. He received a Masters in computer science from the Czech Technical University in Prague in 2001.

***Pascal Gautron**, Thomson Corporate Research France*
pascal.gautron@thomson.net
Pascal Gautron is a R&D engineer at Thomson Corporate Research, Rennes, France. During his Ph.D. in collaboration with the IRISA/INRIA Rennes and the University of Central Florida, his research focused on fast, GPU-based global illumination computation using the irradiance caching algorithm. His current research work considers real-time 3D visual effects for live video production.

***Greg Ward**, Dolby Canada*
gward@lmi.net
Greg Ward has been involved with computer graphics since 1985, when he began development of the RADIANCE lighting simulation and rendering system at the Lawrence Berkeley National Lab. Since then, he has worked on rendering algorithms, reflectance models and measurement systems, tone reproduction operators, and HDR image processing and display techniques. His past employers include the Lawrence Berkeley National Laboratory, EPFL Switzerland, SGI, Shutterfly, and Exponent. Greg holds a bachelor's in Physics from UC Berkeley and a master's in Computer Science from SF State University. He is currently working as an independent consultant in Albany, California <www.anyhere.com>.

***Henrik Wann Jensen**, University of California, San Diego*
henrik@cs.ucsd.edu
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. His contributions to computer graphics include the photon mapping algorithm for global illumination, and the first technique for efficiently simulating subsurface scattering in translucent materials. He is the author of "Realistic Image Synthesis using Photon Mapping," AK Peters 2001. Prior to coming to UCSD in 2002, he was a research associate at Stanford University from 1999-2002, a postdoctoral researcher at the Massachusetts Institute of Technology (MIT) from 1998-1999, and a research scientist in industry working on commercial rendering software from 1996-1998. 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.
In 2004, Professor Jensen received an Academy Award (Technical Achievement Award) from the Academy of Motion Picture Arts and Sciences for pioneering research in rendering translucent materials, and he became a Sloan Fellow.

***Eric Tabellion**, PDI/DreamWorks*
et@pdi.com
Eric Tabellion is a senior software engineer in the rendering group at DreamWorks Animation, where he has been working on film production rendering since 1999. Eric's work has been primarily focused on practical global illumination using irradiance caching, as well as raytracing techniques. He also worked on various aspects of computer graphics, such as efficient post-process motion-blur and depth-of-field rendering algorithms, fluid surface reconstruction techniques, and particle and crowd simulation systems. Eric received a Masters in computer science from the Université de Marne-la-Vallée in 1996. Eric's movie credits include Shrek, Shrek2, Madagascar, Over the Hedge, Flushed Away, Shrek the Third, and Bee-Movie.

***Per H. Christensen**, Pixar Animation Studios*
per@pixar.com
Per Christensen is a senior software developer in Pixar's RenderMan group in Seattle. His main research interest is efficient ray tracing and global illumination in very complex scenes. Before joining Pixar, he worked at Mental Images in Berlin and at Square USA in Honolulu. He received an M.Sc. degree in electrical engineering from the Technical University of Denmark and a Ph.D. in computer science from the University of Washington in Seattle. His movie credits include Finding Nemo, The Incredibles, Cars, Ratatouille, and Wall-E.

Reprinted papers

The following papers are reprinted in the course notes.

[Page 406] Gregory J. Ward, Francis M. Rubinstein, and Robert M. Clear. A Ray Tracing Solution for Diffuse Interreflections. In *SIGGRAPH '88*.

The first paper on irradiance caching. Describes the overall algorithm (the "lazy evaluation" scheme), derives the weight function and the sample spacing from the split sphere model, and proposes the use of an octree to index the cache records.

***Course sections**: all*

[Page 414] Gregory J. Ward and Paul Heckbert. Irradiance Gradients. In *Eurographics Symposium on Rendering*. 1992.

This paper introduces the translational and rotational gradients. It describes how the gradients can be estimated from the stratified hemisphere samples and how they are applied in interpolation to obtain smoother indirect illumination.

***Course sections**: all*

[Page 431] Greg Ward Larson and Rob A. Shakespere. Indirect Calculation. Chapter 12 from "Rendering with Radiance: The Art and Science of Lighting Visualization". Morgan Kaufmann Publishers. 1998.

This book chapter summarizes the contents for the previous two papers with the focus on irradiance caching implementation in the Radiance lighting simulation system. The chapter also gives a summary of Radiance's parameters for irradiance caching.

***Course sections**: all*

[Page 471] Eric Tabellion and Arnaud Lamorlette. Approximate Global Illumination System for Computer Generated Films. In SIGGRAPH 2004.

This paper gives a high-level description of various tools used at PDI/Dremaworks to support global illumination computation, including irradiance caching. Various modifications of the original irradiance caching algorithms are described, such as a new weight function. Integration of irradiance caching in the proprietary relighting tools is also sketched.

***Course sections:** 5 (Problems and Solutions: Implementation Details), 10 (Irradiance Caching at PDI/DreamWorks)*

[Page 479] Jaroslav Křivánek, Pascal Gautron , Sumanta Pattanaik, and Kadi Bouatouch. Radiance Caching for Efficient Global Illumination Computation. Technical Report no. 1623. IRISA, Rennes, France. June 2004.

Introduces radiance caching, an extension of irradiance caching to glossy surfaces. A novel gradient computation method is proposed. The final version was published in IEEE Transactions on Visualization and Computer Graphics, Vol. 11, No. 5, September/October 2005.

***Course sections**: 7 (Extension to Glossy Surfaces: Radiance Caching)*

[Page 498] Jaroslav Křivánek, Pascal Gautron, Kadi Bouatouch, and Sumanta Pattanaik. Improved Radiance Gradient Computation. In SCCG 2005**:** *Proceedings of Spring Conference on Computer Graphics*, 2005.

Describes a generalization of Ward and Heckbert's gradient computation [1992] that can be used for radiance caching. Specifically, the assumption of cosine-proportional hemisphere sampling is lifted. The technique gives better results than the gradient computation described by the original radiance caching paper [Křivánek et al. 2005].

***Course sections:** 7 (Extension to Glossy Surfaces: Radiance Caching)*

[Page 503] Jaroslav Křivánek, Kadi Bouatouch, Sumanta Pattanaik, and Jiří Žára. Making Radiance and Irradiance Caching Practical: Adaptive Caching and Neighbor Clamping. In *Rendering Techniques, Proceedings Eurographics Symposium on Rendering*, 2006.

Two independent contributions are described. First an adaptive method for radiance and irradiance caching that takes into account the difference in illumination from neighboring records to adapt the record density to actual illumination conditions. This technique is somewhat similar to the irradiance caching implementation described by Per Christensen in Section 11 of the course, but it works in screen space rather than in the parametric surface space. The second contribution is the neighbor clamping heuristic used to improve robustness of irradiance caching.

***Course sections**: 5 (Problems and Solutions: Implementation Details), 7 (Extension to Glossy Surfaces: Radiance Caching)*

[Page 515] Pascal Gautron , Jaroslav Křivánek, Kadi Bouatouch, and Sumanta Pattanaik. Radiance Cache Splatting: A GPU-Friendly Global Illumination Algorithm. In *Proceedings of Eurographics Symposium on Rendering*, 2005.

The paper reformulates irradiance caching to make it amenable to GPU implementation. Octree lookups are replaced by splatting the record contributions to the screen. GPU rasterization is used instead of hemisphere sampling. The algorithm gives up to 40x speedup compared to Radiance and affords for interactive walkthroughs with global illumination.

***Course section**: 8 (Hardware Implementation)*

[Page 525] Pascal Gautron, Kadi Bouatouch, and Sumanta Pattanaik. Temporal Radiance Caching. Technical Report no. 1796, IRISA, Rennes, France

This paper deals with the problem of illumination flickering animations. In addition to spatial interpolation, irradiance (and radiance) is interpolated in time using temporal *gradients. The result is a flicker-free animation computed in shorter time than if the the frames were computed independently. The final version was published in IEEE Transactions on Visualization and Computer Graphics, Vol. 13, No. 5, September/October 2007.*

***Course section**: 9 (Temporal Caching)*

Further reading

The following papers are closely related but not reprinted in the class notes.

Okan Arikan, David Forsyth, and James O'Brien. Fast and Detailed Approximate Global Illumination by Irradiance Decomposition. SIGGRAPH 2005.

This paper reduces the computation time of irradiance caching by reducing the number of cache records (and consequently hemisphere samplings). The indirect illumination field is divided into near and far field. Far field is computed with classical hemisphere sampling but the records can be much more sparsely spaced. The near field is then quickly computed by ignoring local visibility.

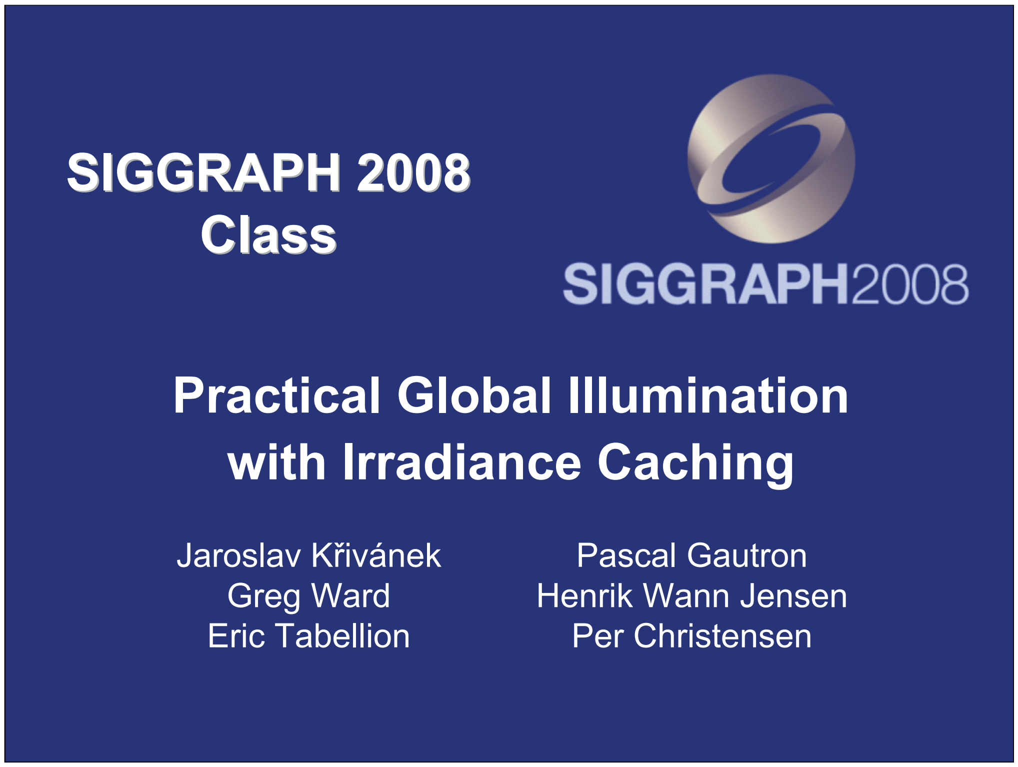
Wojciech Jarosz, Craig Donner, Matthias Zwicker, and Henrik Wann Jensen. Radiance caching for participating media. ACM Trans. Graph. Vol. 27, No. 1 (March 2008).

In this paper, irradiance and radiance caching are applied to accelerate global illumination computation in participating media. Gradients for single and multiple scattering terms are derived. The algorithm results in superior quality than photon mapping and faster rendering times than path tracing.

Wojciech Jarosz, Matthias Zwicker Henrik, and Wann Jensen. Irradiance Gradients in the Presence of Participating Media and Occlusions. Eurographics Symposium on Rendering 2008.

This paper shows that some of the assumptions of the common irradiance gradient computation techniques are not valid in scenes with participating media. An irradiance gradient calculation algorithm is presented that takes into account the participating media and yields smoother interpolation. In addition to the gradients derived in the TOG paper [Jarosz et al. 2008], the gradient computation presented here takes visibility changes into account.

SIGGRAPH2008



SIGGRAPH 2008
Class

SIGGRAPH2008

Practical Global Illumination with Irradiance Caching

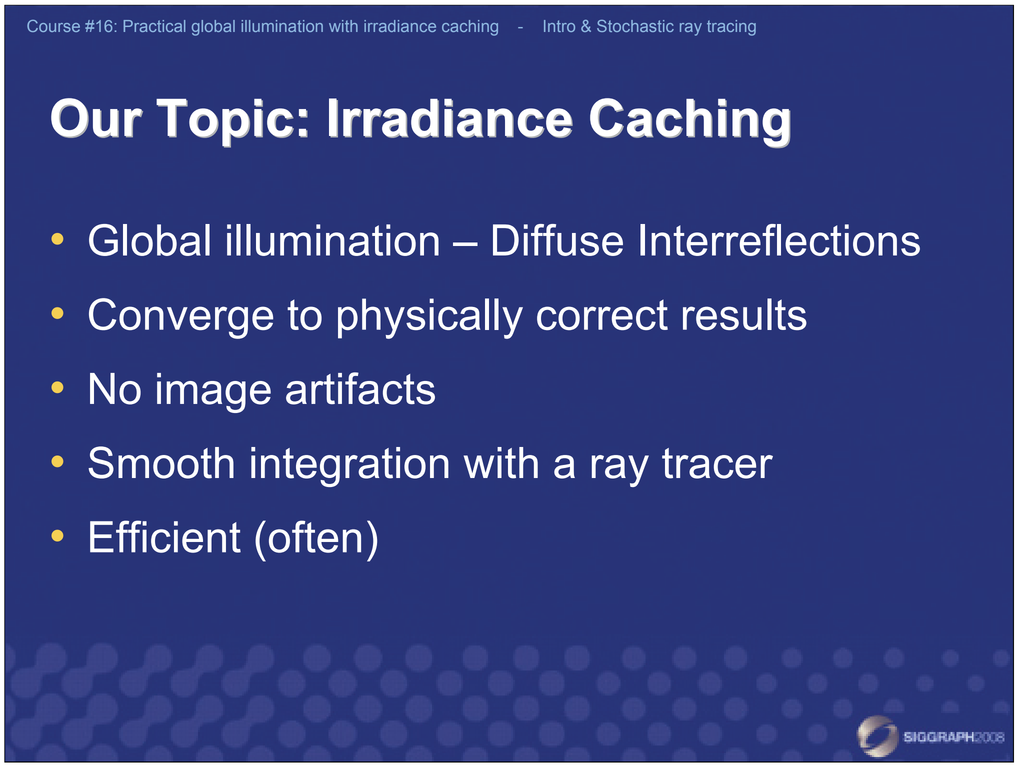
Jaroslav Křivánek
Greg Ward
Eric Tabellion
Pascal Gautron
Henrik Wann Jensen
Per Christensen

Introduction

SIGGRAPH2008

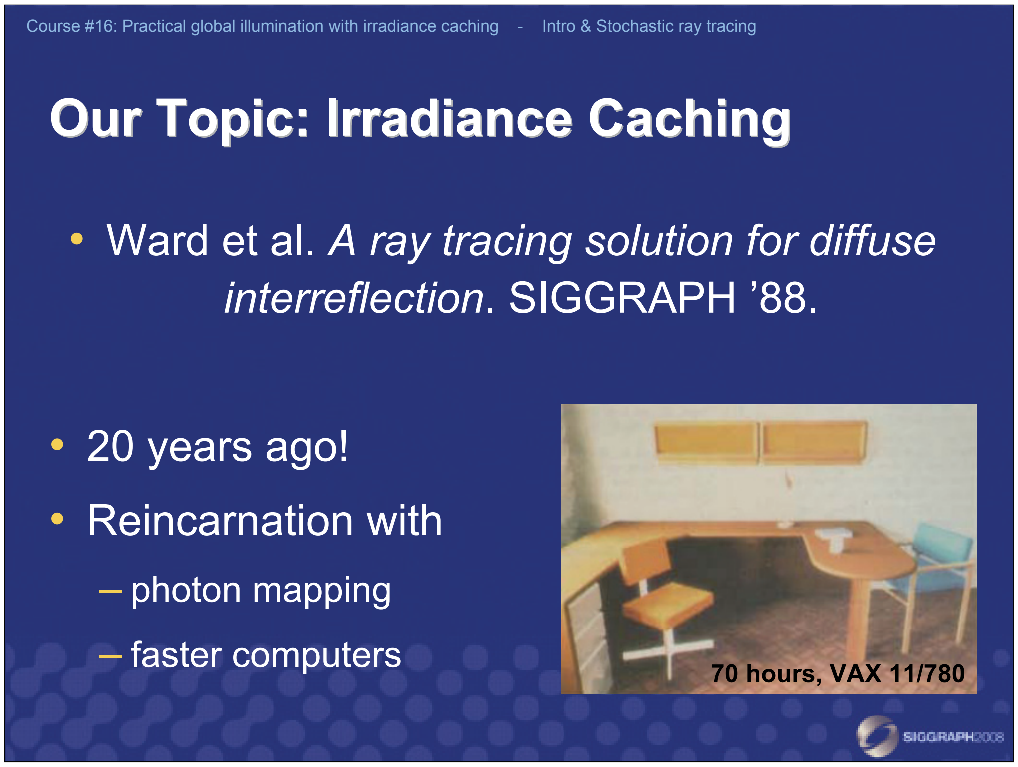
Jaroslav Křivánek

Czech Technical University in Prague



Our Topic: Irradiance Caching

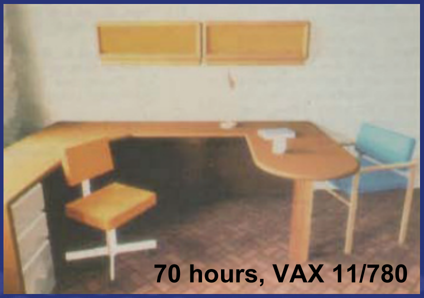
- Global illumination – Diffuse Interreflections
- Converge to physically correct results
- No image artifacts
- Smooth integration with a ray tracer
- Efficient (often)

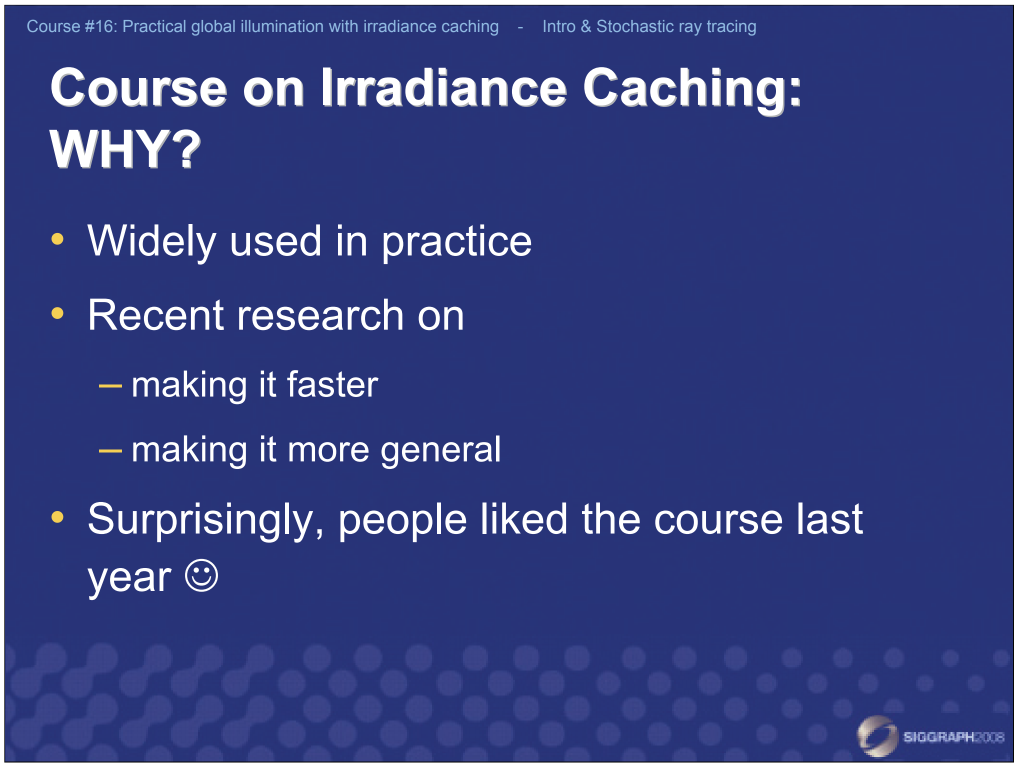


Our Topic: Irradiance Caching

- Ward et al. *A ray tracing solution for diffuse interreflection*. SIGGRAPH '88.

- 20 years ago!
- Reincarnation with
  - photon mapping
  - faster computers


70 hours, VAX 11/780

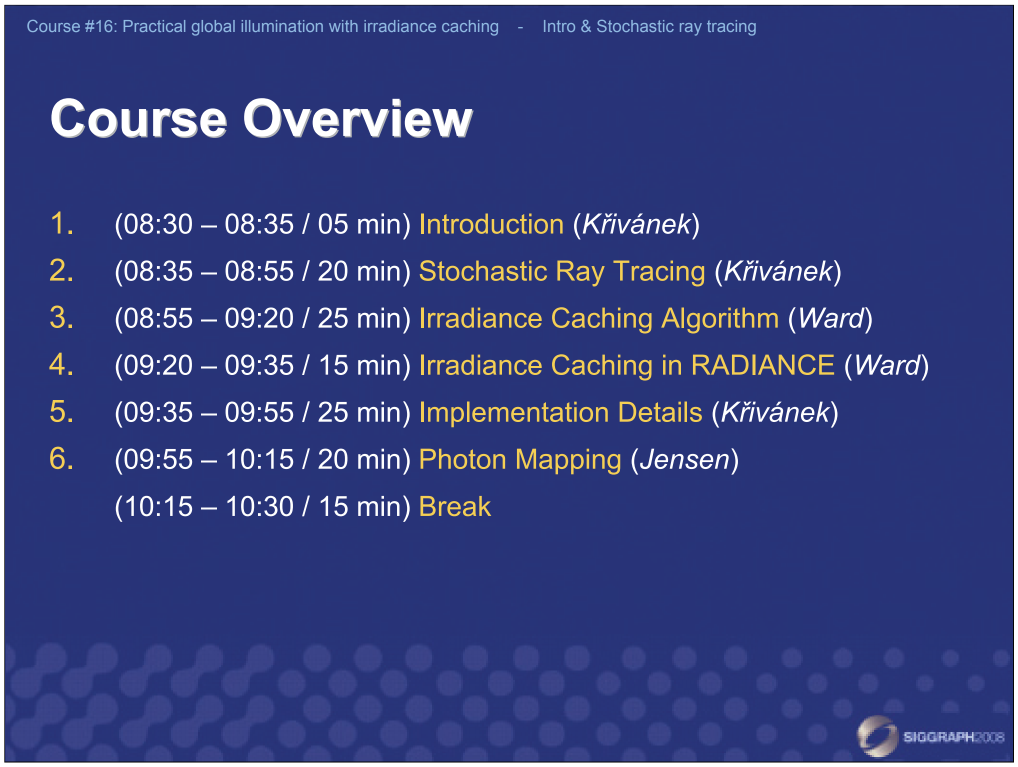


Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Course on Irradiance Caching: WHY?

- Widely used in practice
- Recent research on
  - making it faster
  - making it more general
- Surprisingly, people liked the course last year ☺

SIGGRAPH2008



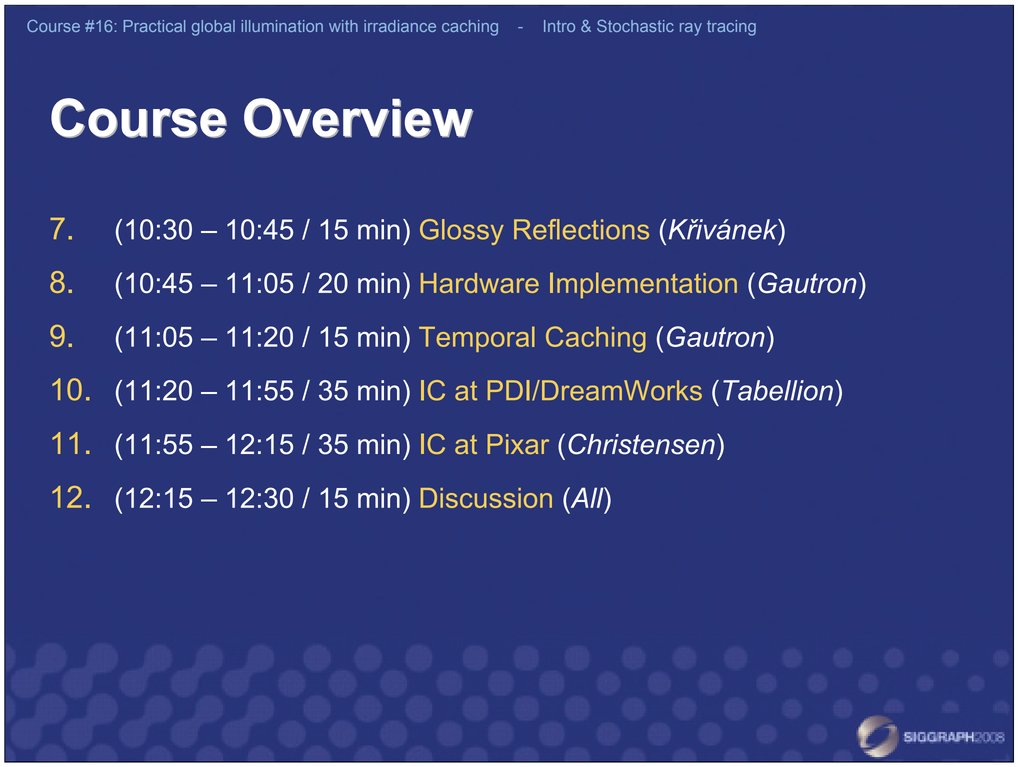
Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Course Overview

1. (08:30 – 08:35 / 05 min) Introduction (*Křivánek*)
2. (08:35 – 08:55 / 20 min) Stochastic Ray Tracing (*Křivánek*)
3. (08:55 – 09:20 / 25 min) Irradiance Caching Algorithm (*Ward*)
4. (09:20 – 09:35 / 15 min) Irradiance Caching in RADIANCE (*Ward*)
5. (09:35 – 09:55 / 25 min) Implementation Details (*Křivánek*)
6. (09:55 – 10:15 / 20 min) Photon Mapping (*Jensen*)

(10:15 – 10:30 / 15 min) Break

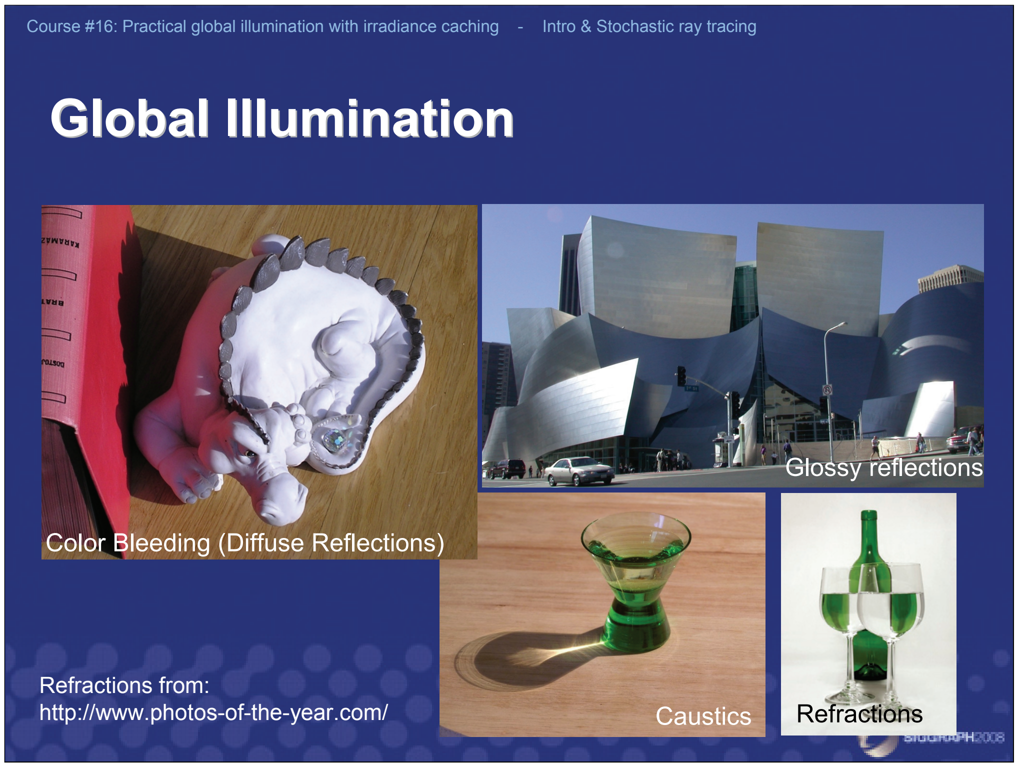
SIGGRAPH2008



Course Overview

7. (10:30 – 10:45 / 15 min) Glossy Reflections (*Křivánek*)
8. (10:45 – 11:05 / 20 min) Hardware Implementation (*Gautron*)
9. (11:05 – 11:20 / 15 min) Temporal Caching (*Gautron*)
10. (11:20 – 11:55 / 35 min) IC at PDI/DreamWorks (*Tabellion*)
11. (11:55 – 12:15 / 35 min) IC at Pixar (*Christensen*)
12. (12:15 – 12:30 / 15 min) Discussion (*All*)

SIGGRAPH2008



Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Global Illumination

Color Bleeding (Diffuse Reflections)

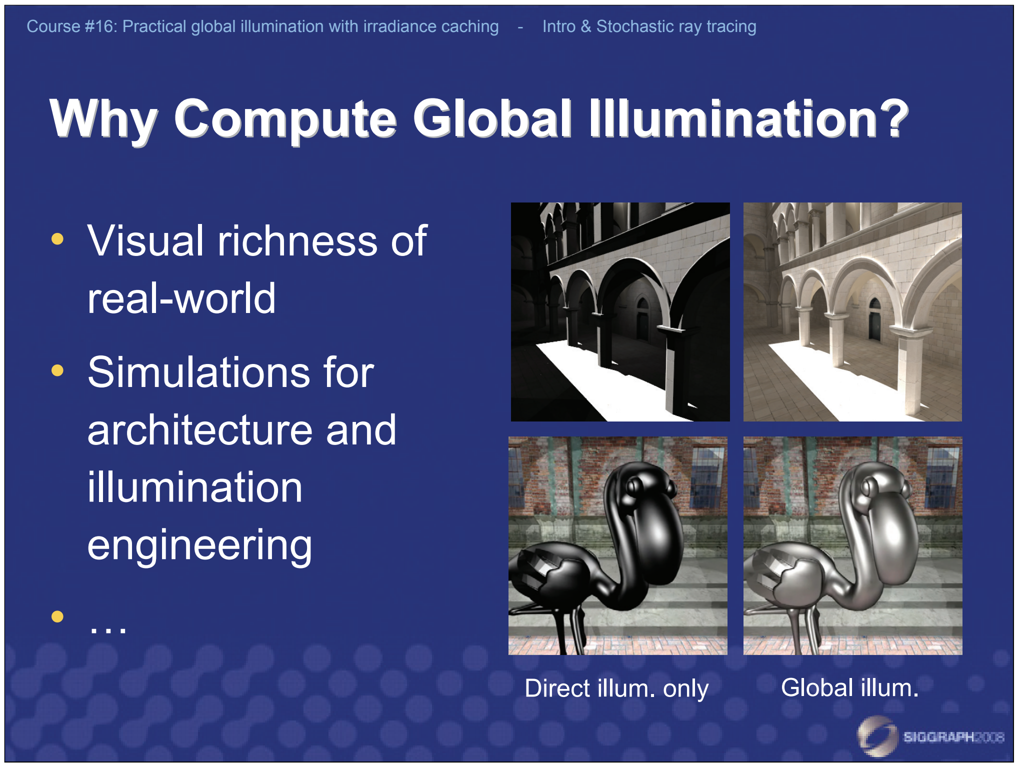
Glossy reflections

Caustics

Refractions

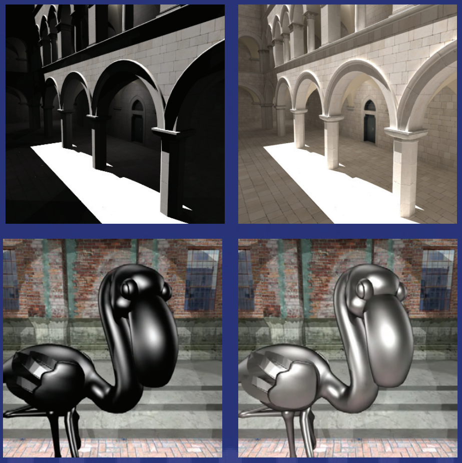
Refractions from:
http://www.photos-of-the-year.com/

The term “global illumination” embraces many lighting effects encountered in real world. Some of them are shown on this slide.



Why Compute Global Illumination?

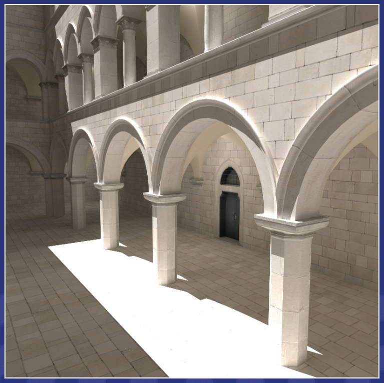
- Visual richness of real-world
- Simulations for architecture and illumination engineering
- …

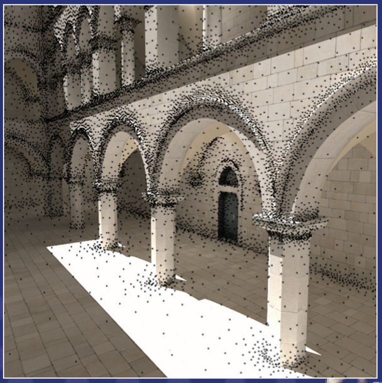


Simulating global illumination can reproduce the visual richness of real world. It is also useful for predictive rendering, such as architecture or illumination engineering. In these applications, it is essential to simulate light transport accurately so that the simulation results can be predictive of what the actual construction would look like.

Diffuse Interreflections with Irradiance Caching

- Purpose: faster computation
- Means: sparse sampling & interpolation



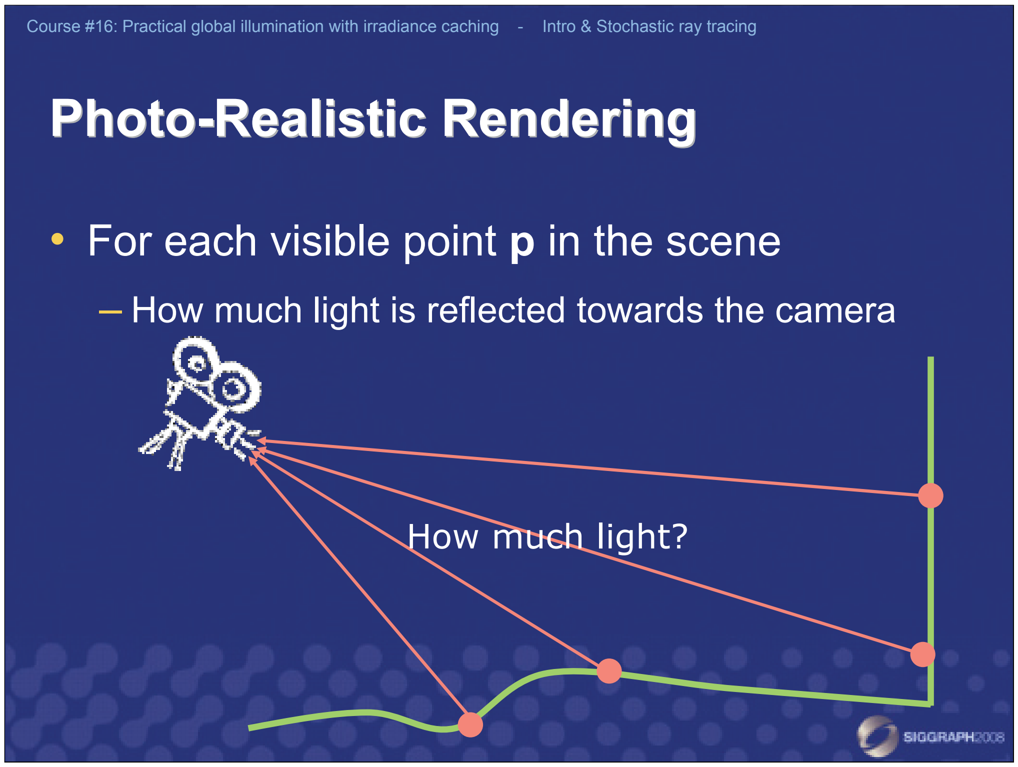


The focus of this course is on irradiance caching – an algorithm for fast computation of global illumination on diffuse surfaces. The algorithm gains its efficiency by performing the costly global illumination computation on a sparse set of locations, caching the results and using them to interpolate illumination elsewhere.

Global Illumination with Monte Carlo Ray Tracing

SIGGRAPH2008

Jaroslav Křivánek
ČVUT v Praze – CTU Prague



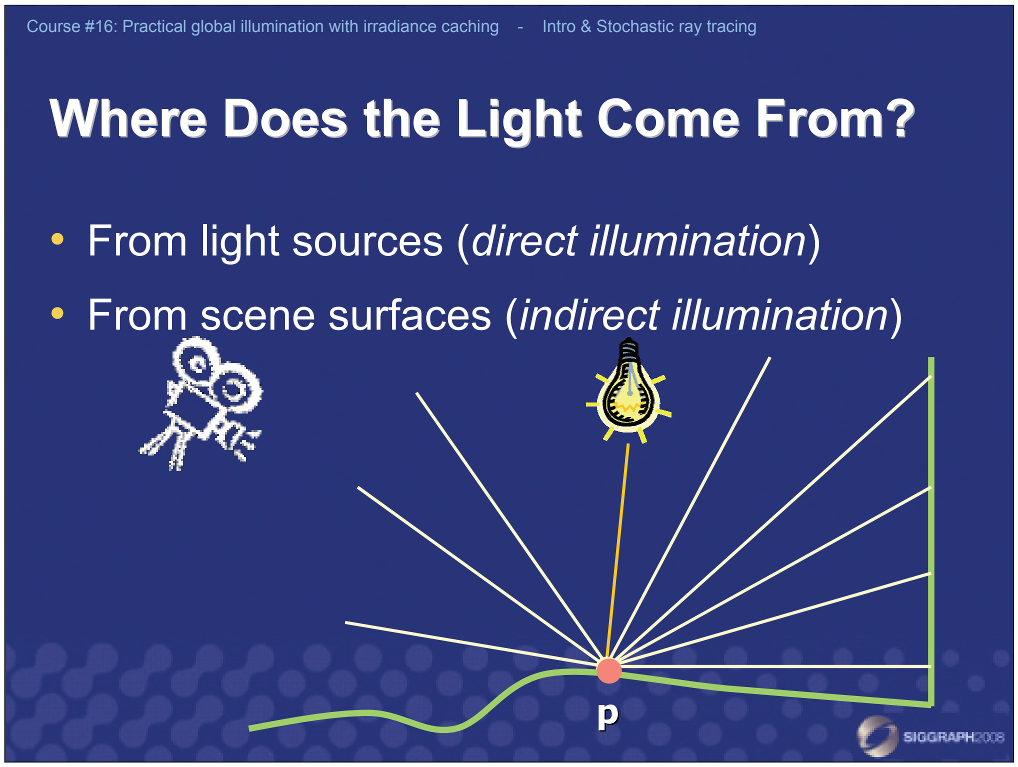
Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Photo-Realistic Rendering

- For each visible point **p** in the scene
  - How much light is reflected towards the camera

How much light?
SIGGRAPH2008

Given the description of a scene, the goal of photo-realistic rendering is to compute the amount of light reflected from visible scene surfaces that arrives to the virtual camera through image pixels. This light determines the color of image pixels.

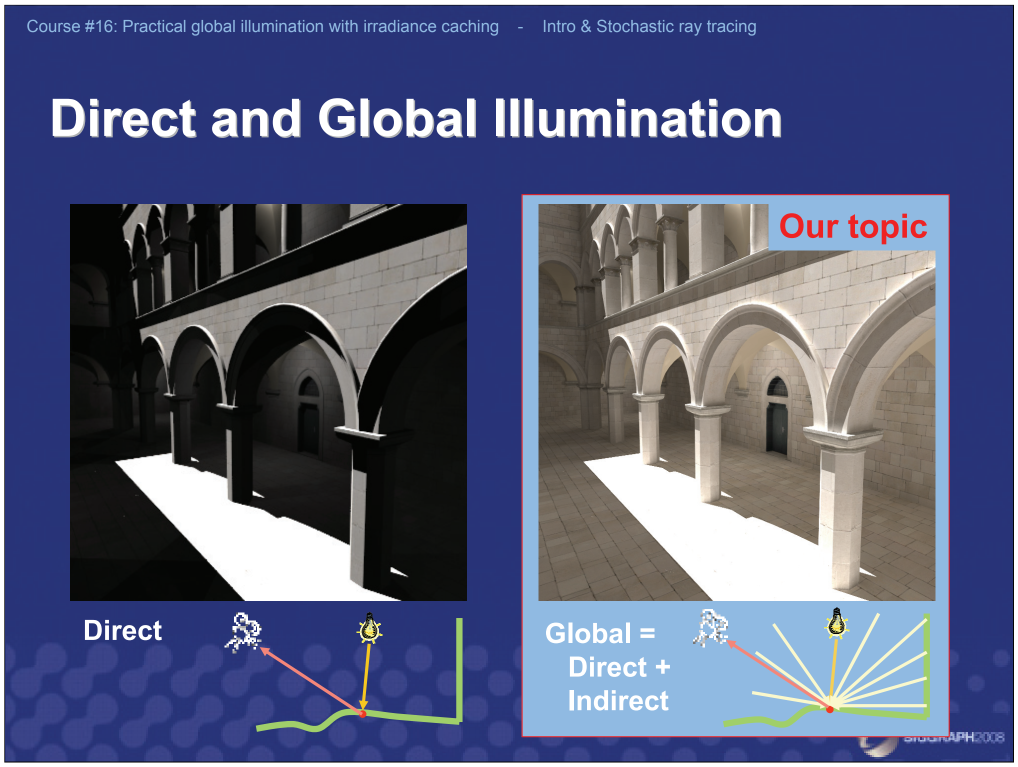


Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Where Does the Light Come From?

- From light sources (*direct illumination*)
- From scene surfaces (*indirect illumination*)

Focusing on one such point, where does the light come from? Surely, it comes directly from the light sources placed in the scene – this is called the *direct illumination*. But light also comes after being reflected from other scene surfaces. This is the *indirect illumination*.



Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

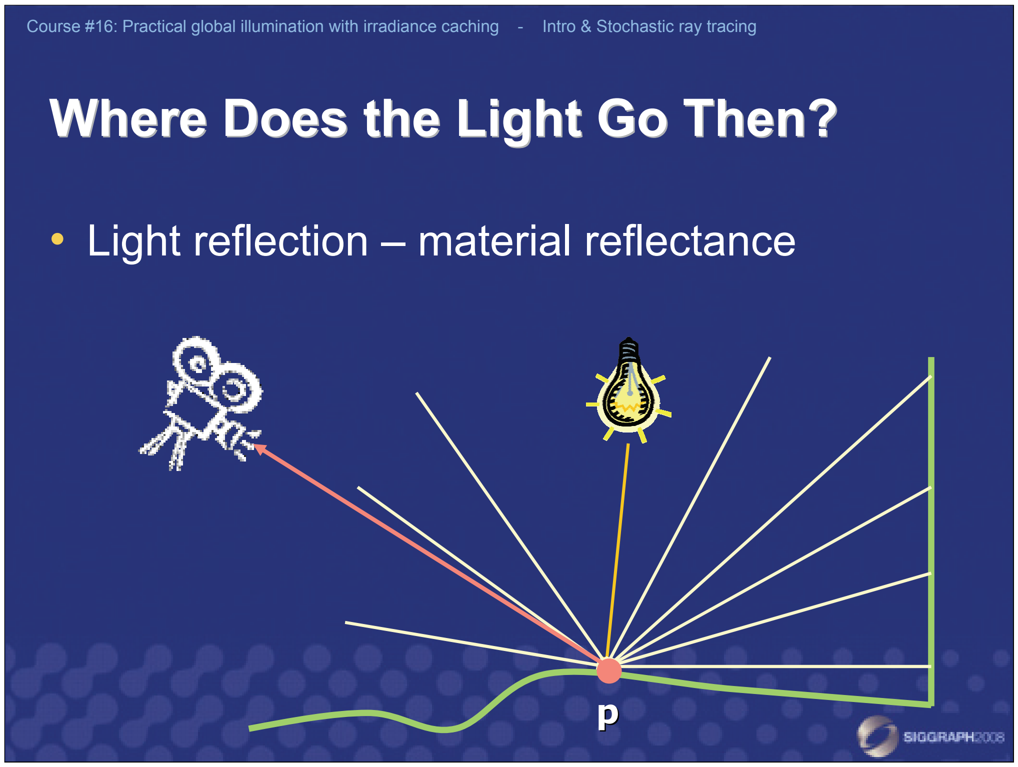
Direct and Global Illumination

Our topic

Direct

Global = Direct + Indirect

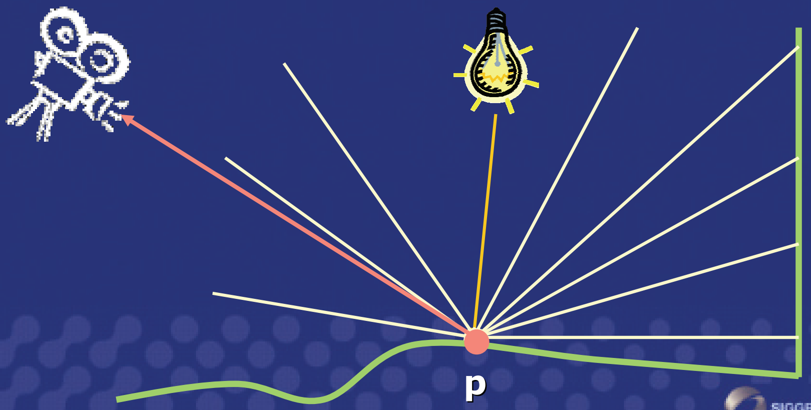
On the left is an image generated by taking into account only direct illumination. Shadows are completely black because, obviously, there is no direct illumination in the shadowed areas. On the right is the result of global illumination computation where light is allowed to bounce around in the scene before it is finally reflected towards the camera.



Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

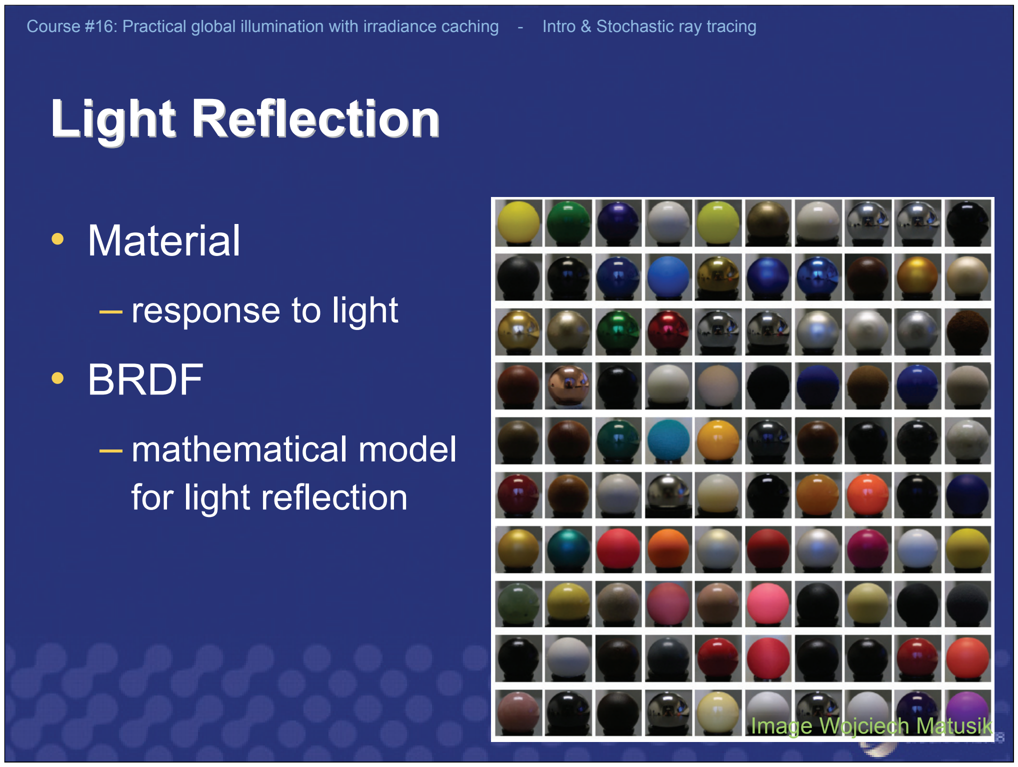
Where Does the Light Go Then?

- Light reflection – material reflectance



SIGGRAPH2008

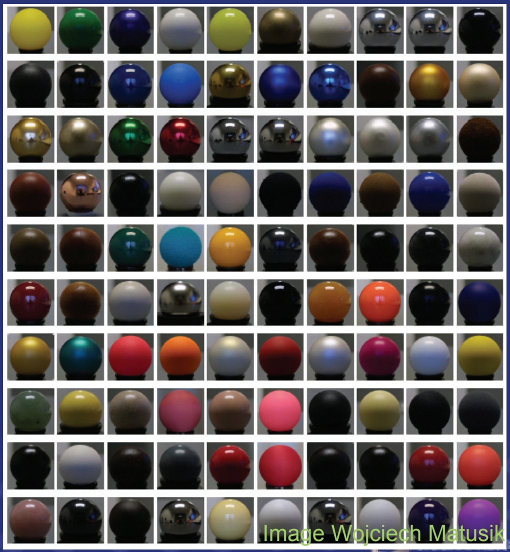
So far, we know where the light comes from at **p**. But we want to compute how much of it is reflected towards the camera. This is determined by the surface material reflectance properties.



Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

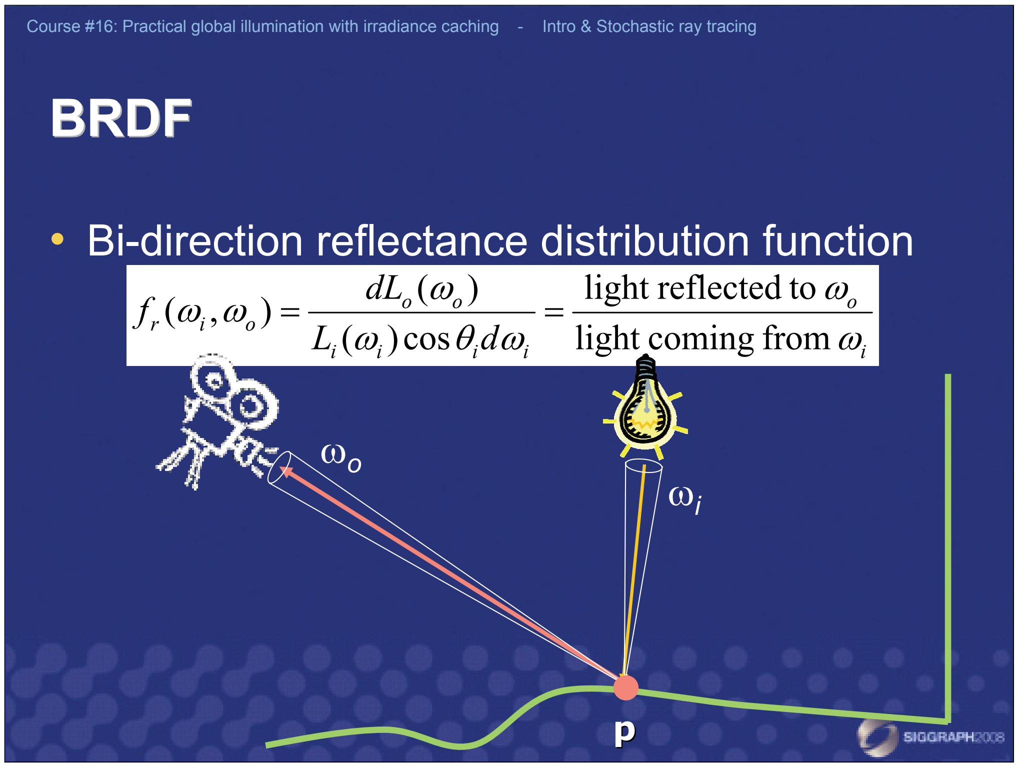
Light Reflection

- Material
  - response to light
- BRDF
  - mathematical model for light reflection



From the point of view of light simulation, the material can be looked upon as the surface's response to incoming light. The spheres on the right are all illuminated by the same light – their varying appearance is only determined by their different material properties.

In physically-based image synthesis (rendering), the most common way to mathematically describe material reflectance characteristics is the BRDF – bidirectional reflectance distribution function.



Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

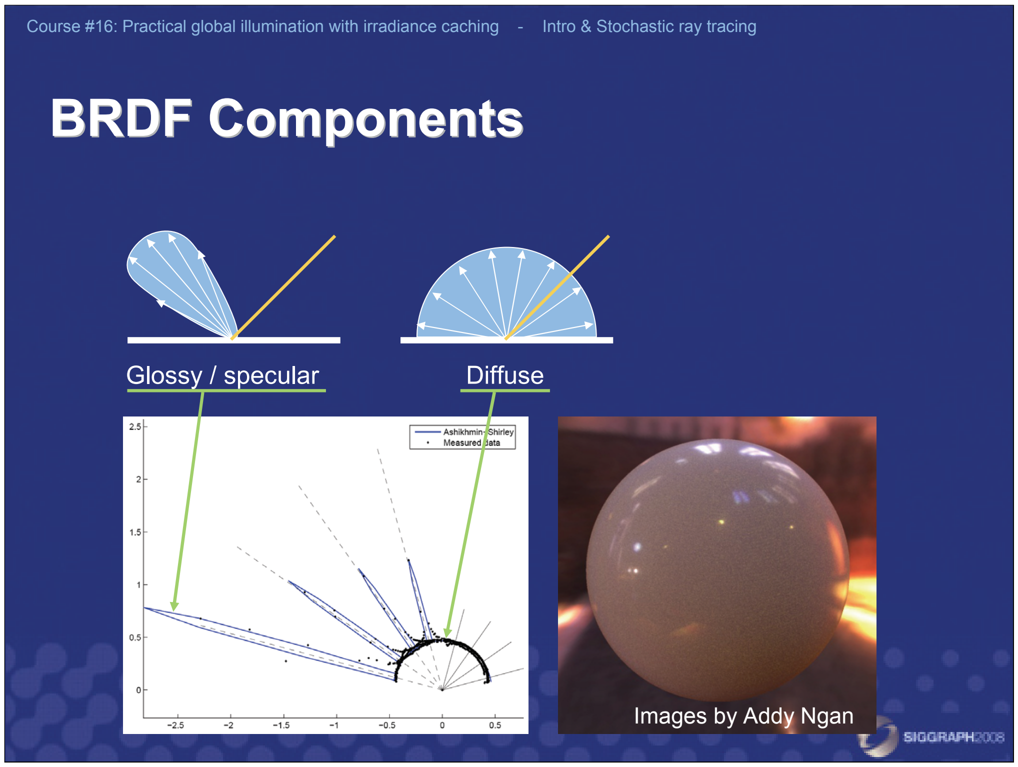
BRDF

- Bi-direction reflectance distribution function

$$f_r(\omega_i,\omega_o) = \frac{dL_o(\omega_o)}{L_i(\omega_i)\cos\theta_i d\omega_i} = \frac{\text{light reflected to } \omega_o}{\text{light coming from } \omega_i}$$

ωo
ωi
p
SIGGRAPH2008

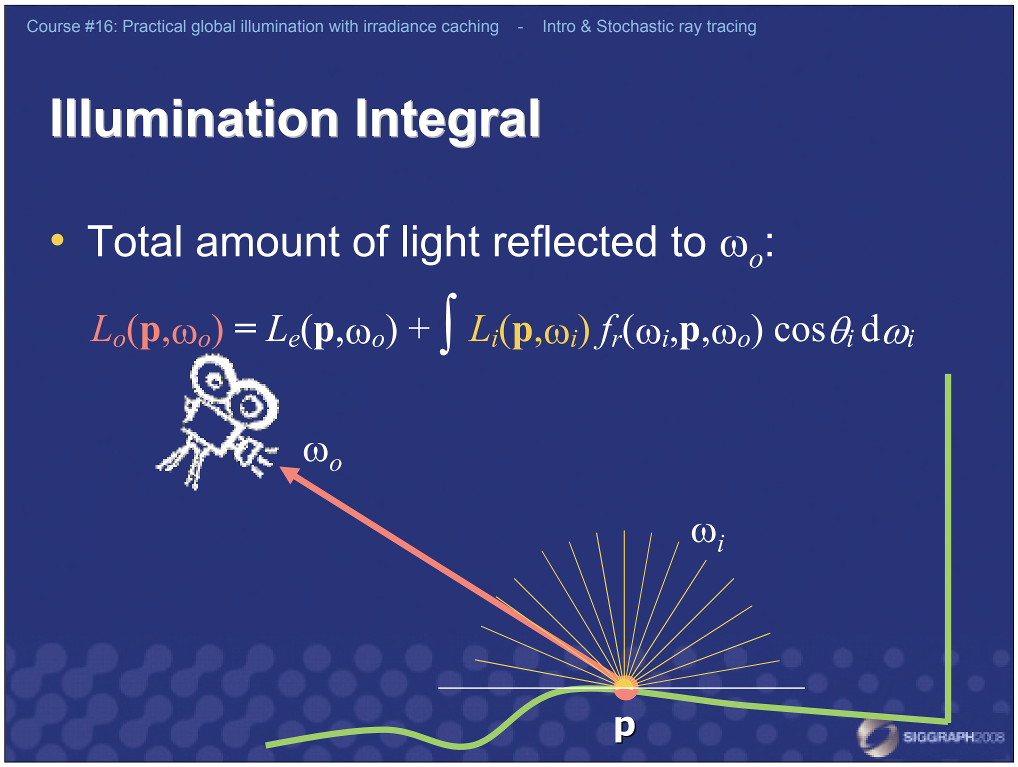
The BRDF at a single point has two parameters: the incoming and the outgoing light directions, ω_i and ω_o, respectively. For our purposes, it is sufficient to say that the BRDF value is the ratio of the light reflected in the outgoing direction to the light coming from the incoming direction. In other words, the BRDF tells us how much of the light energy coming from ω_i gets reflected towards ω_o. In radiometry, the quantity associated with the amount of light reaching a point **p** along a direction w is the **radiance** $L(\mathbf{p},w)$.



Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

BRDF Components

For practical purposes, we express a general BRDF as a sum of several terms, or components. On this slide you see a plot of the BRDF of an acrylic white paint (from the supplemental material for [Ngan 2002]). The BRDF is plotted for four different outgoing directions. For each fixed outgoing direction, the BRDF becomes a function of only the incoming direction – this is the BRDF lobe. In our example, there is a prominent diffuse component and clearly visible specular component on top of that. The diffuse BRDF component is a constant function, independent of the incoming and outgoing direction. It corresponds to the ‘color’ of an object. The specular BRDF component has significant values only in a narrow cone, often around the direction of perfect mirror reflection. The specular component adds the highlight on the sphere.



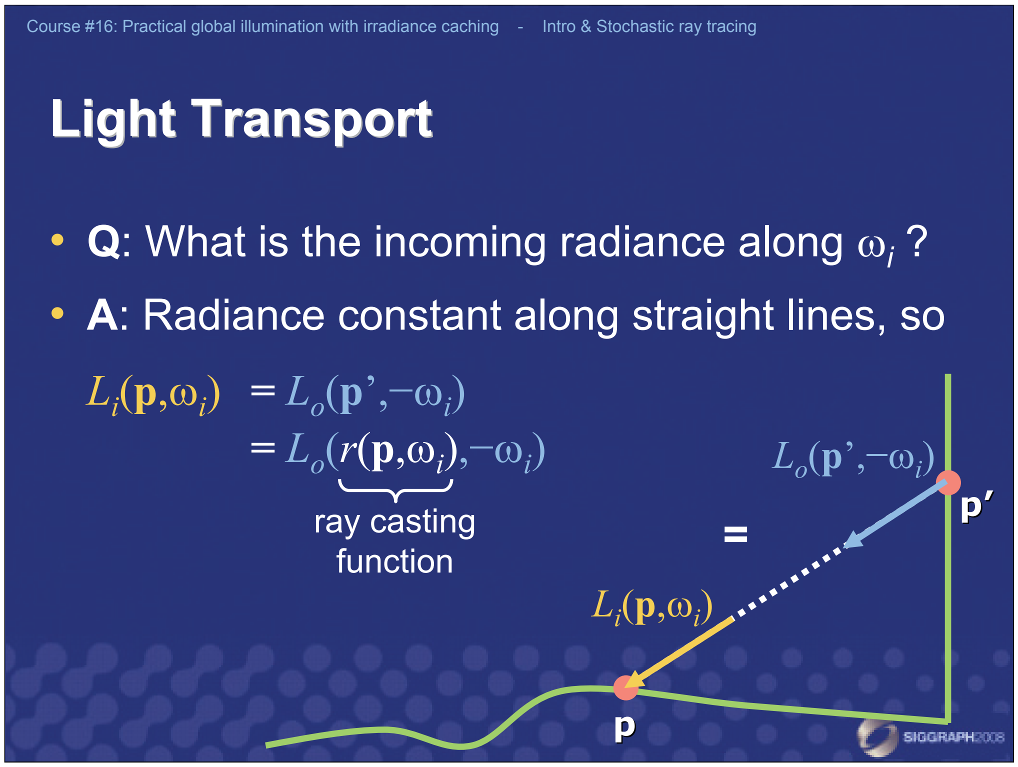
Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Illumination Integral

- Total amount of light reflected to ω_o:

$$L_o(\mathbf{p},\omega_o) = L_e(\mathbf{p},\omega_o) + \int L_i(\mathbf{p},\omega_i)\, f_r(\omega_i,\mathbf{p},\omega_o)\, \cos\theta_i \, d\omega_i$$

Let us go back to image rendering. We want to compute the amount of light going from **p** towards the camera. Assuming we know how much light is coming to **p** from all possible directions (the incoming hemisphere), the reflected light can be computed by evaluating the illumination integral: For each incoming direction, we multiply the radiance from that direction by the BRDF and the cosine term. To get the total amount of reflected light, we sum (integrate) these contributions over all incoming directions. Then we add the self emitted light at **p** (in case **p** is on the light source) and we have the total amount of light going from **p** towards the camera.



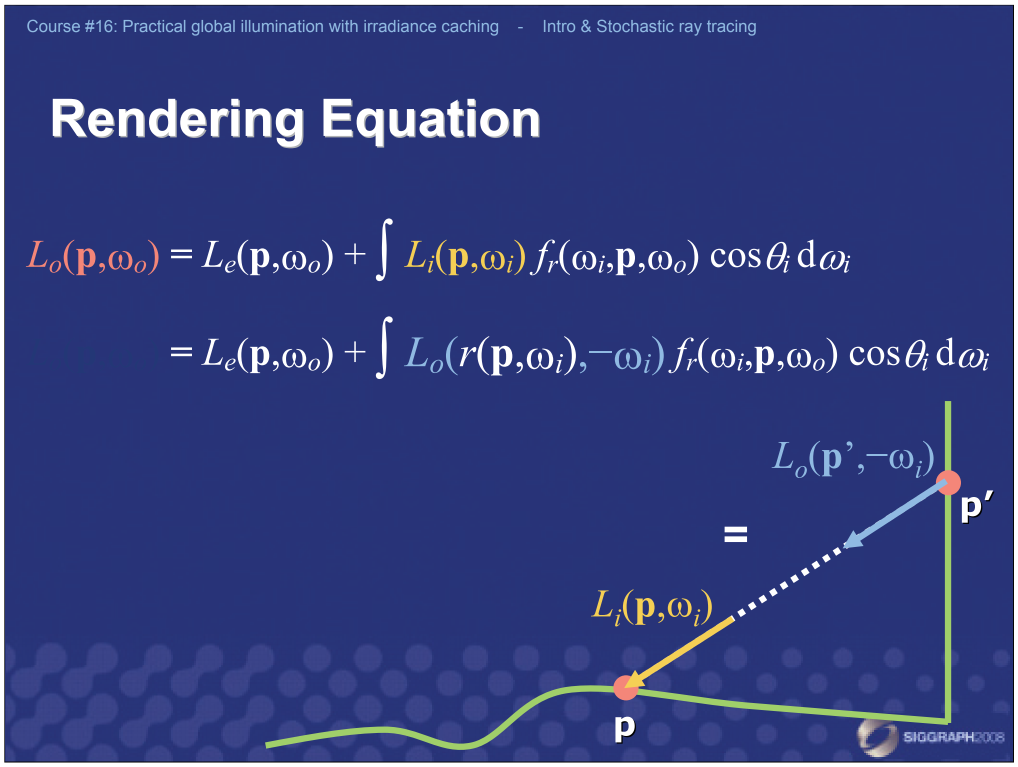
Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Light Transport

- **Q**: What is the incoming radiance along ω_i ?
- **A**: Radiance constant along straight lines, so

$$L_i(\mathbf{p},\omega_i) = L_o(\mathbf{p}',-\omega_i) = L_o(\underbrace{r(\mathbf{p},\omega_i)}_{\text{ray casting function}},-\omega_i)$$

We made an important assumption in computing the illumination integral. We assumed to know how much light is coming to **p** from each direction. But how do we find this out? Taking advantage of the fact that *radiance does not change along straight lines*, we see that the incoming radiance from direction w_i is equal to the outgoing radiance at a point **p**’ visible from **p** along w_i.



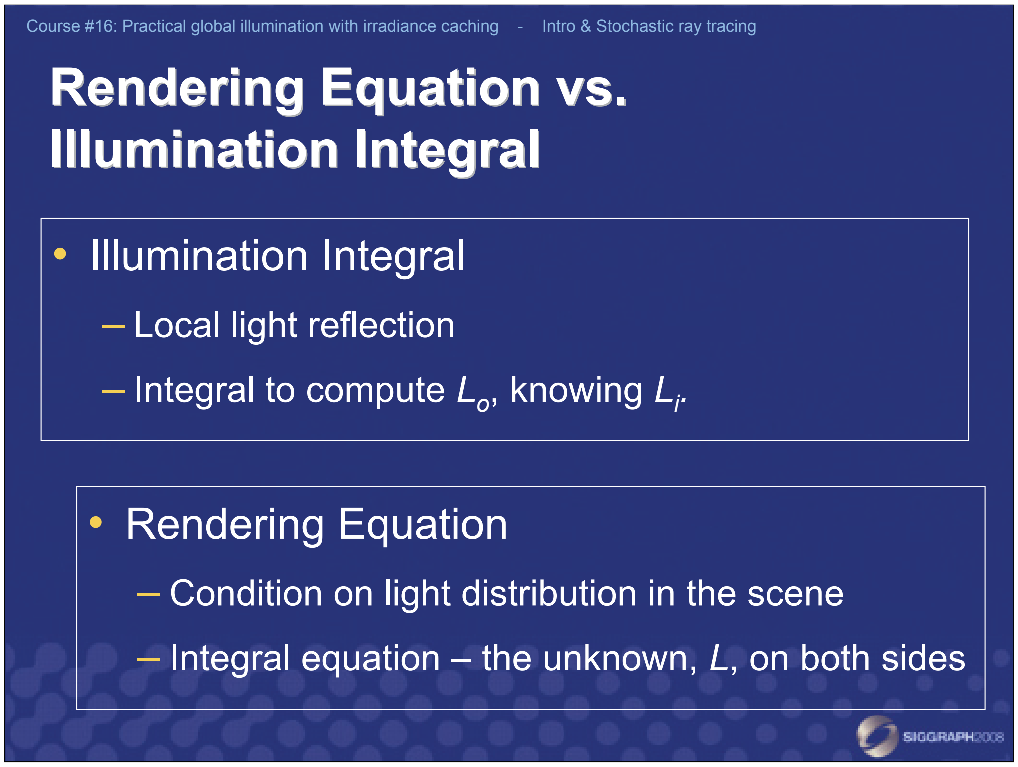
Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Rendering Equation

$$L_o(\mathbf{p},\omega_o) = L_e(\mathbf{p},\omega_o) + \int L_i(\mathbf{p},\omega_i)\, f_r(\omega_i,\mathbf{p},\omega_o) \cos\theta_i \, d\omega_i$$

$$= L_e(\mathbf{p},\omega_o) + \int L_o(r(\mathbf{p},\omega_i),-\omega_i)\, f_r(\omega_i,\mathbf{p},\omega_o) \cos\theta_i \, d\omega_i$$

Plugging our knowledge of light transport to the illumination integral yields *the rendering equation* [Kajiya 1986].



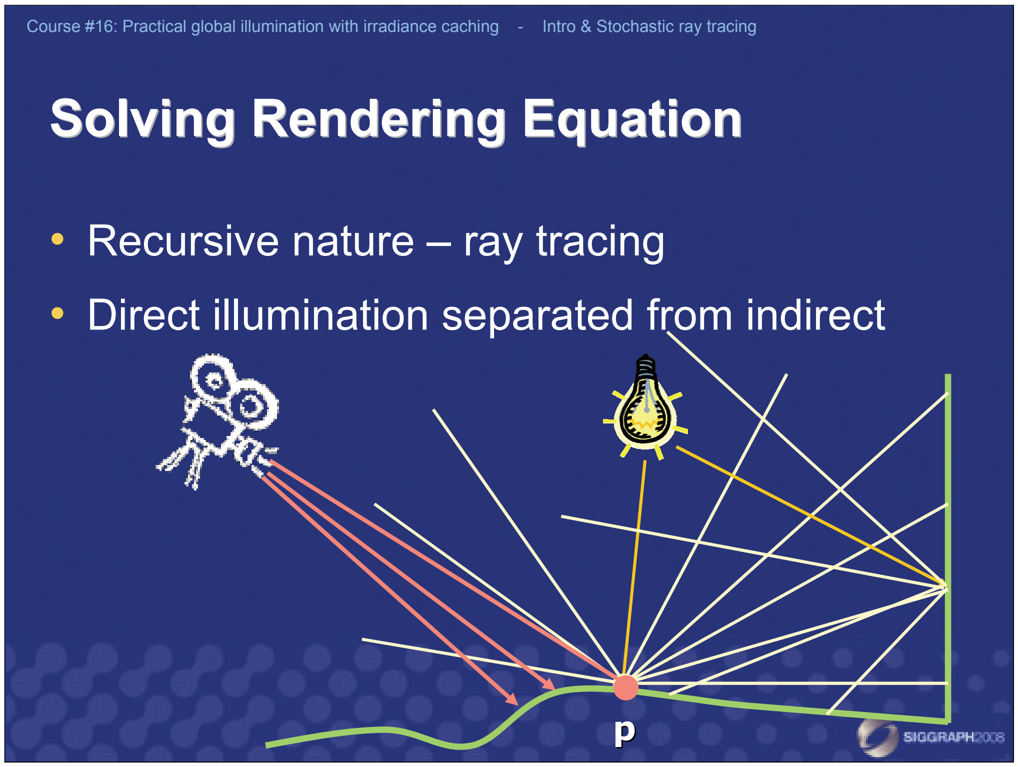
Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Rendering Equation vs. Illumination Integral

- Illumination Integral
  - Local light reflection
  - Integral to compute L_o, knowing L_i.
- Rendering Equation
  - Condition on light distribution in the scene
  - Integral equation – the unknown, L, on both sides

SIGGRAPH2008

Unlike the illumination integral, which is simply a formula that computes reflected light from incident light and therefore has local character, the rendering equation has a more profound meaning. It is truly an ‘equation’ – the unknown quantity, radiance L, – appears on both sides. The rendering equation expresses a condition on equilibrium light distribution in the whole scene.



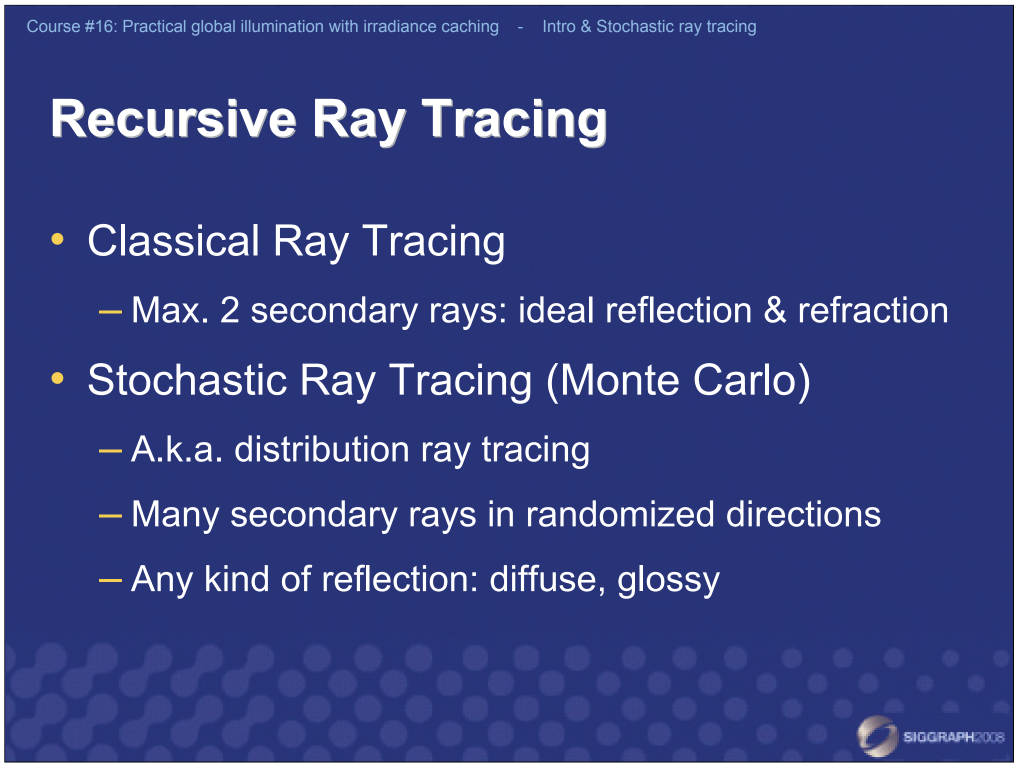
Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Solving Rendering Equation

- Recursive nature – ray tracing
- Direct illumination separated from indirect

Although the rendering equation is at the heart of all theoretical analyses of global illumination, our solution strategy – recursive ray tracing – is much more reminiscent of the illumination integral. At each point of the scene, we want to estimate the outgoing light. In order to do this, we want to numerically evaluate the illumination integral. So we shoot secondary rays, which are ‘samples’ of the incoming radiance. These samples are then multiplied by the BRDF and averaged to numerically estimate the outgoing radiance.

For each of these secondary rays, we need to evaluated the outgoing radiance at a point where they intersect the scene. So we use the same procedure recursively again and again. The recursion can be limited since each light reflection takes away some of the transported light energy. So after couple of reflections, the contribution of light to the image becomes insignificant.



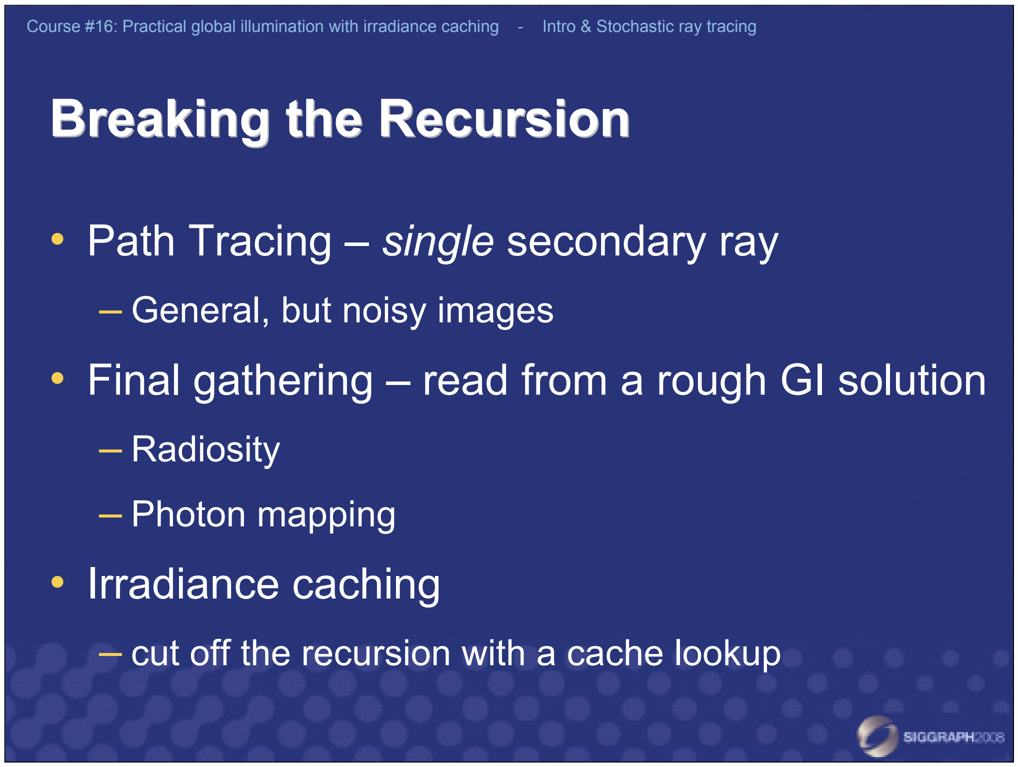
Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Recursive Ray Tracing

- Classical Ray Tracing
  - Max. 2 secondary rays: ideal reflection & refraction
- Stochastic Ray Tracing (Monte Carlo)
  - A.k.a. distribution ray tracing
  - Many secondary rays in randomized directions
  - Any kind of reflection: diffuse, glossy

SIGGRAPH2008

In the classical ray tracing, we shoot at most two secondary rays – in the direction of ideal specular reflection and refraction. That means we disregard indirect illumination on anything but ideal specular reflectors or refractors (such as water, glass etc.) If we want to add indirect light on surfaces of arbitrary reflective characteristics (glossy, diffuse), we need to send many rays to estimate the illumination integral – this is most often referred to as *distribution ray tracing*.



Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

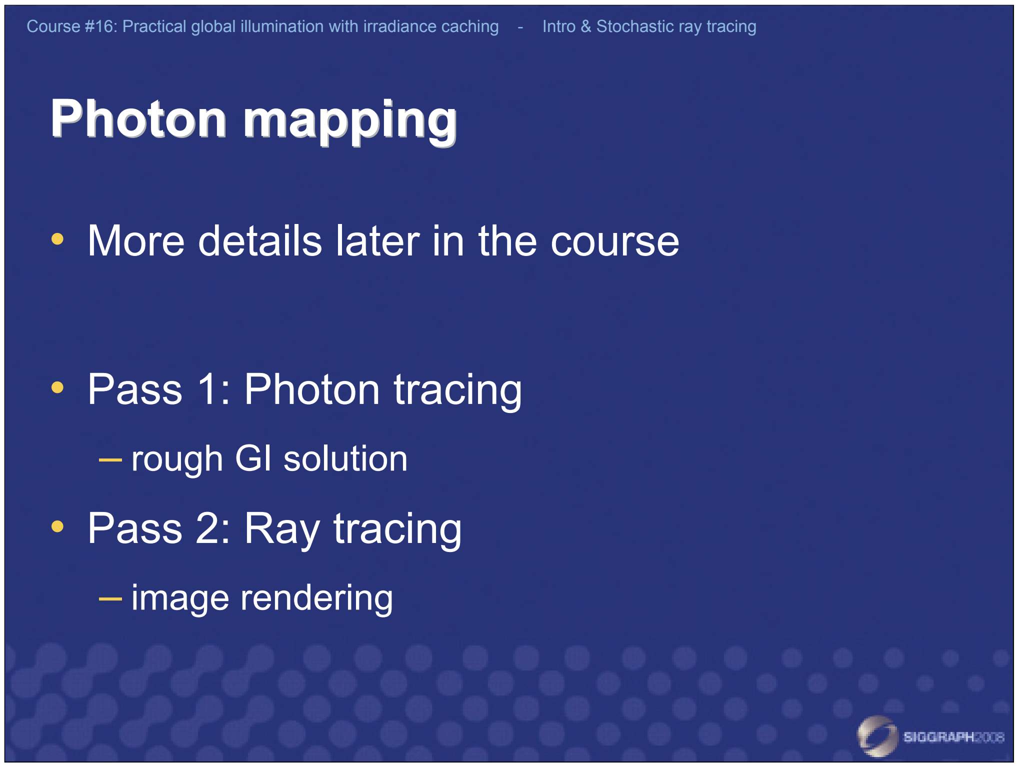
Breaking the Recursion

- Path Tracing – *single* secondary ray
  - General, but noisy images
- Final gathering – read from a rough GI solution
  - Radiosity
  - Photon mapping
- Irradiance caching
  - cut off the recursion with a cache lookup

SIGGRAPH2008

One serious problem of distribution ray tracing is the explosion of the number of rays due to the recursion, where each ray is split into many rays upon each reflection. One way to combat this problem is path tracing where only one single ray is generated at each reflection. The rays then form linear ‘paths’ instead of the ray tree in distribution ray tracing. Path tracing is the original strategy that Jim Kajiya [1986] proposed for solving the rendering equation. It is very general and simple to implement. However, thousands of paths have to be traced through each image pixel in order to compute images without visible noise, which is time-consuming.

Another way to break the recursion is to compute a rough global illumination solution in a pre-process and store it in the scene. Then, as we trace the camera rays, we can read the rough GI solution instead of continuing the recursion. In practice, the most common way of doing this is to use photon mapping.



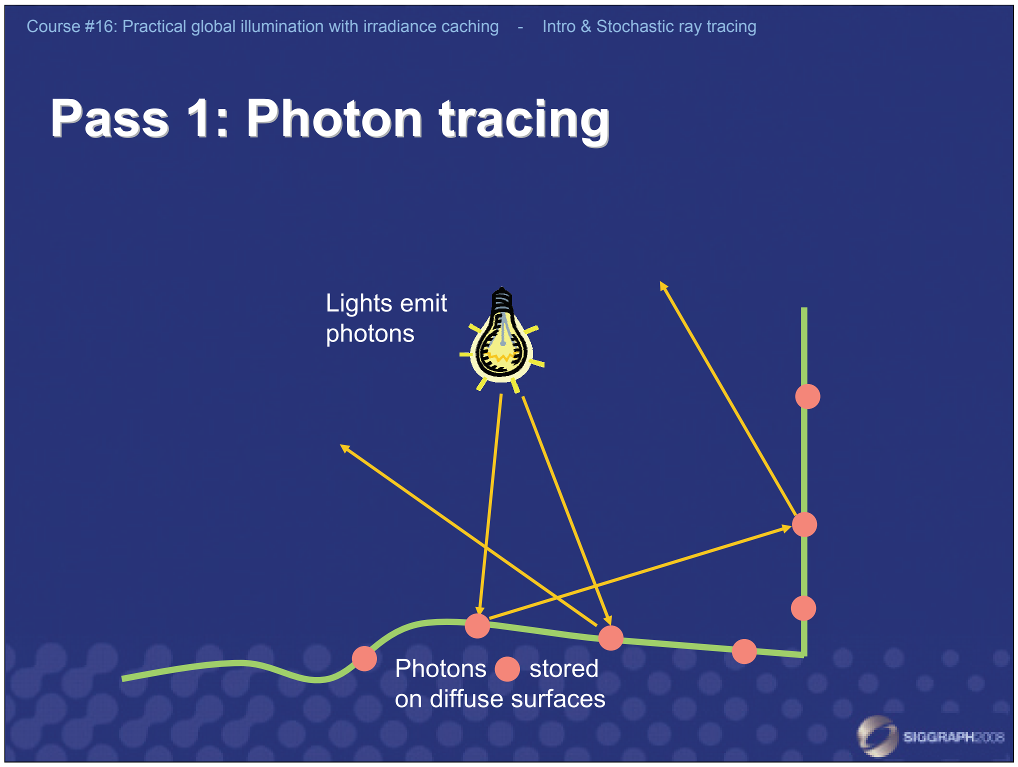
Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Photon mapping

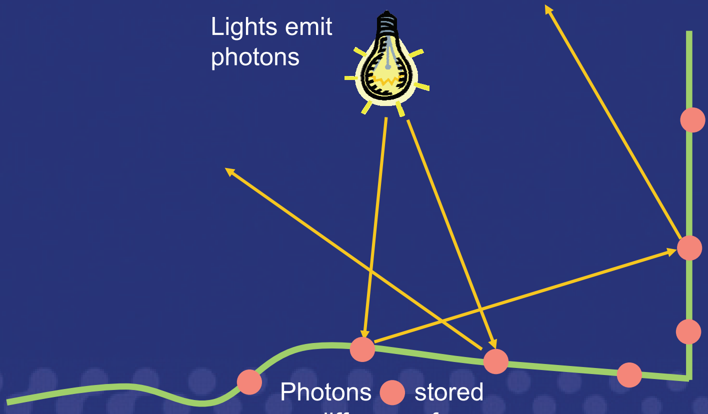
- More details later in the course
- Pass 1: Photon tracing
  - rough GI solution
- Pass 2: Ray tracing
  - image rendering

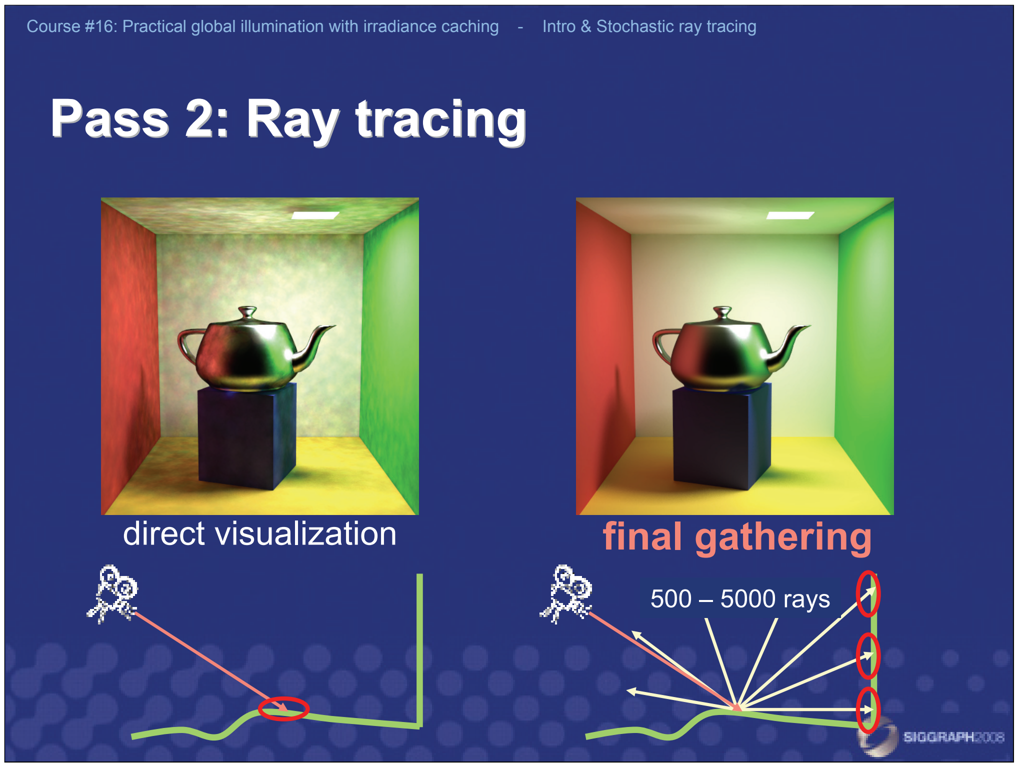
SIGGRAPH2008

Photon mapping works in two passes. In the first pass – photon tracing – a rough GI solution is created in form of ‘photons’ distributed in the scene. The second pass is distribution ray tracing as described previously, where recursion is limited by reading the rough GI solution from the photon map.



Pass 1: Photon tracing

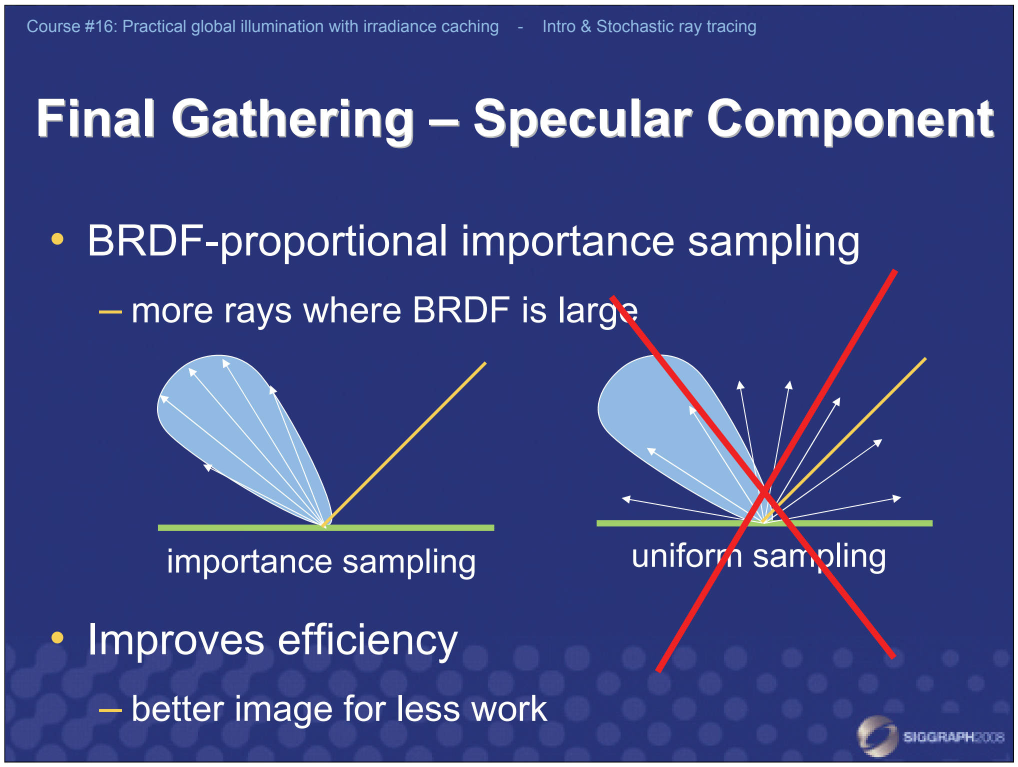




Pass 2: Ray tracing

In ray tracing with photon maps, one possibility is to read the rough GI solution from the photon map immediately as a primary ray hits a surface. However, this gives a ‘splotchy’ artifacts in the images, precisely because the GI solution in the photon map is very rough.

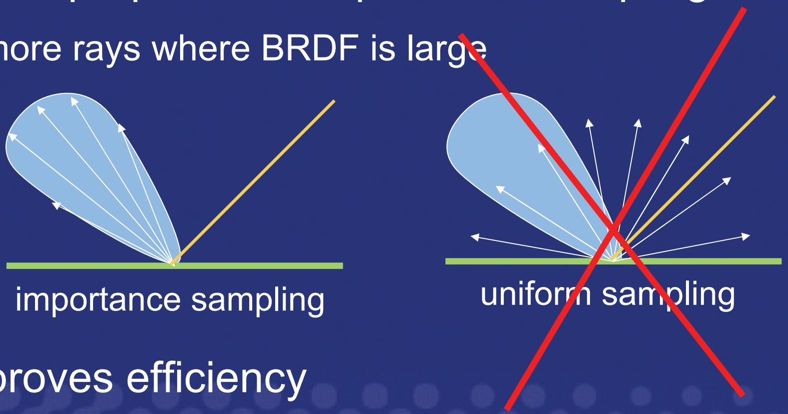
A better way is to perform one level of distribution ray tracing and read the photon map after the first reflection. Shooting many secondary rays averages out any artifacts and the result is a clean image with global illumination. This technique is called *final gathering*. Final gathering as described so far is *slow*: for each of the thousands or millions primary rays, hundreds or thousands gather rays have to be shot.



Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Final Gathering – Specular Component

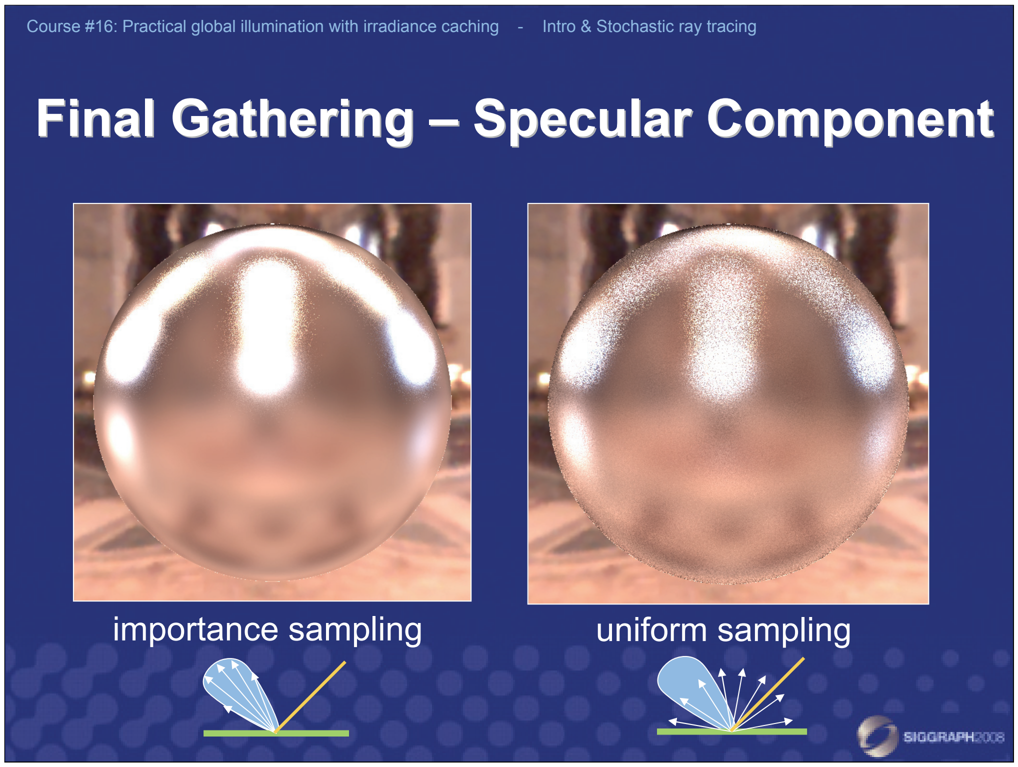
- BRDF-proportional importance sampling
  - more rays where BRDF is large



- Improves efficiency
  - better image for less work

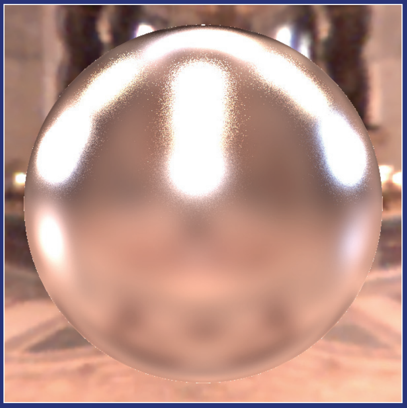
On specular surfaces, an extremely powerful way to reduce the cost of distribution ray tracing – and final gathering – is *importance sampling*. Since the BRDF lobe corresponding to the (known) outgoing direction has significant values only in a narrow cone, there is no use wasting time in sampling the whole hemisphere uniformly. It is much better to concentrate the rays within that cone.

Importance sampling improves efficiency – we get the same quality results with much fewer rays (or much better quality with the same number of rays). The more specular surface, the more effective importance sampling.

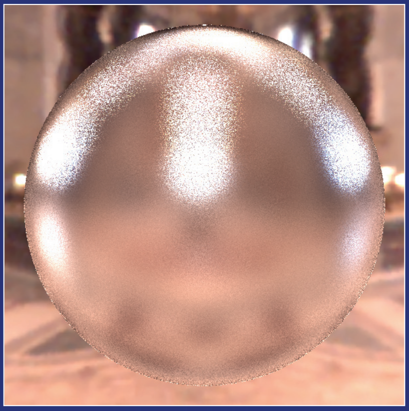


Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

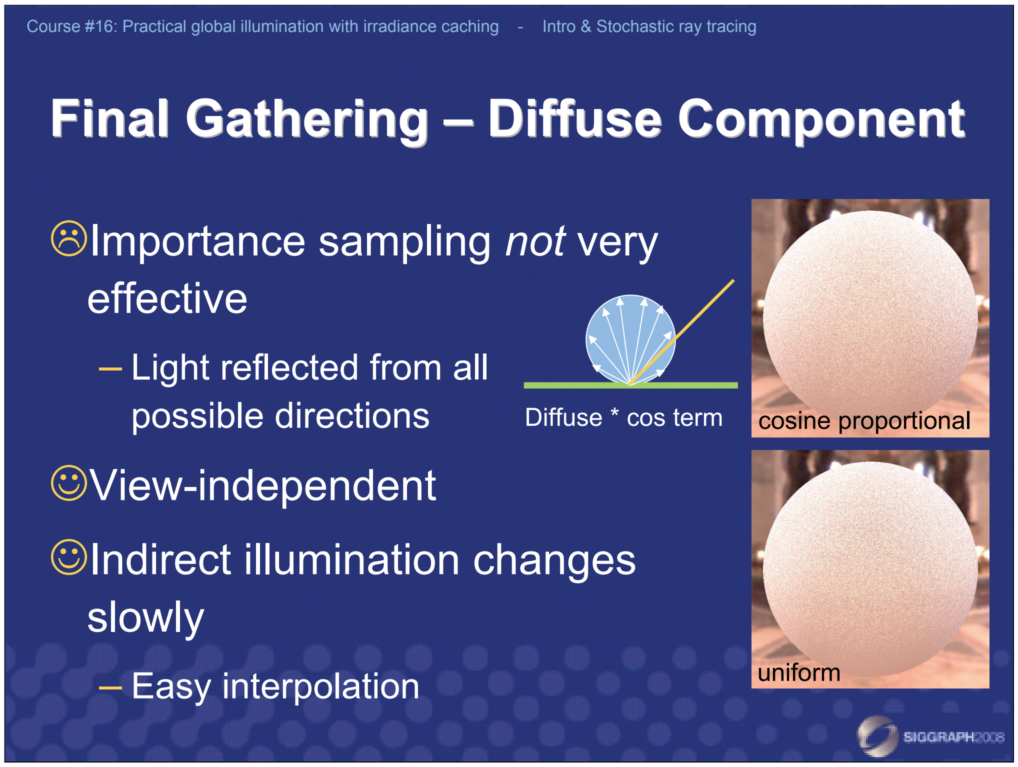
Final Gathering – Specular Component



importance sampling



uniform sampling



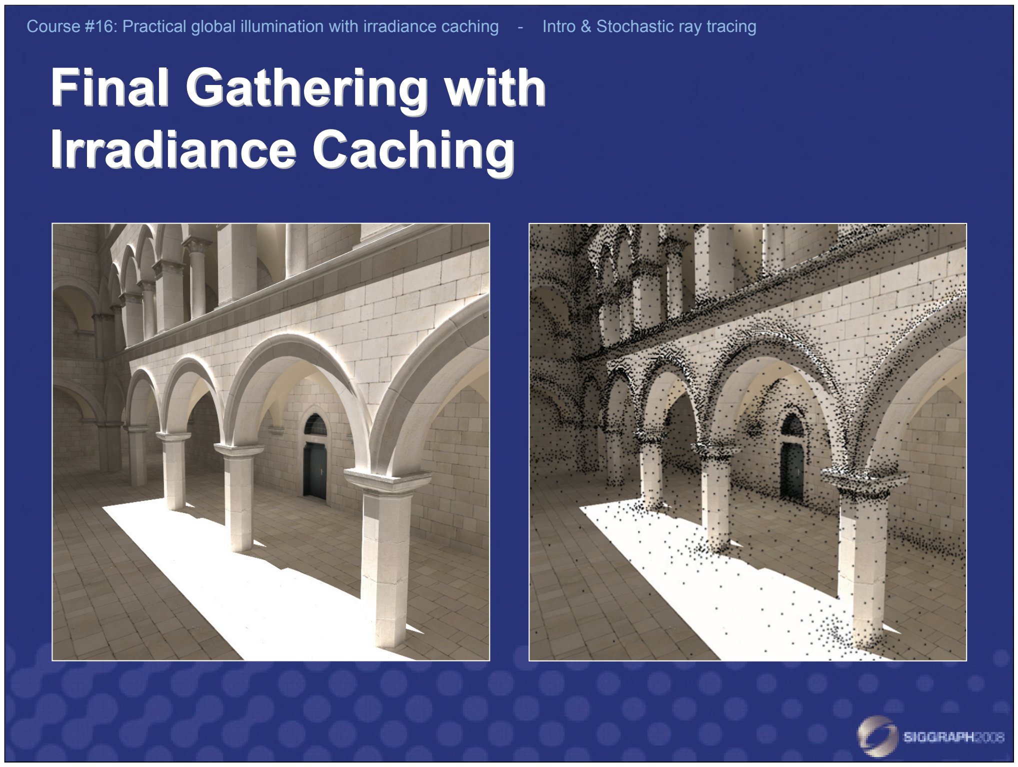
Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Final Gathering – Diffuse Component

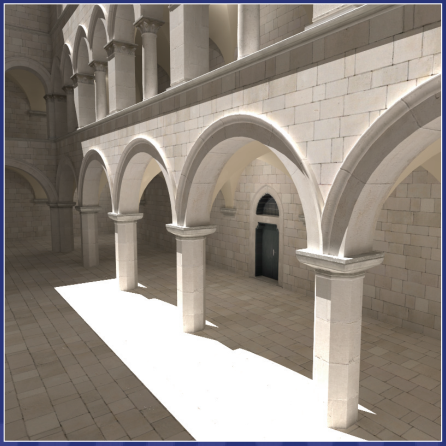
- ☹Importance sampling *not* very effective
  - Light reflected from all possible directions
- ☺View-independent
- ☺Indirect illumination changes slowly
  - Easy interpolation

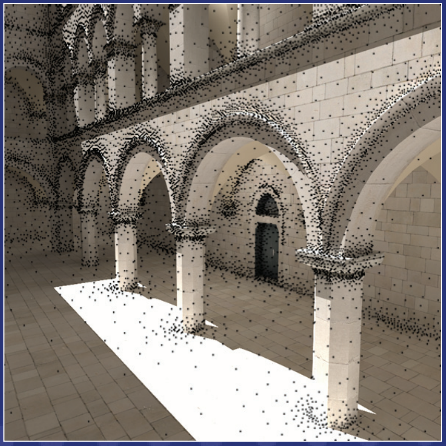
Diffuse * cos term
cosine proportional
uniform
SIGGRAPH2008

Unfortunately, importance sampling isn't very effective on diffuse surfaces, since light is reflected nearly equally from all incoming directions. Two good news help us make distribution ray tracing on diffuse surfaces efficient: 1) indirect illumination is view-independent and 2) it varies slowly over surfaces. This makes it relatively easy to compute indirect illumination sparsely in the scene and interpolate.

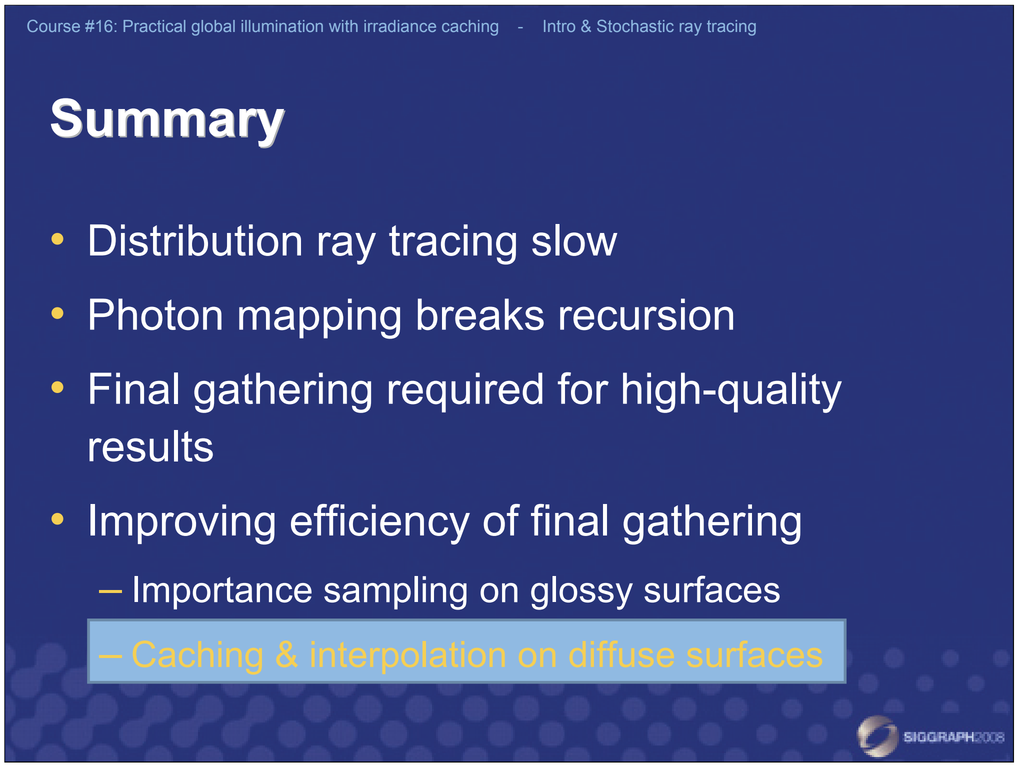


Final Gathering with Irradiance Caching





Shown on the left are the positions at which the costly final gathering computation was performed. The computed indirect illumination values were then used for interpolation for the remaining positions.

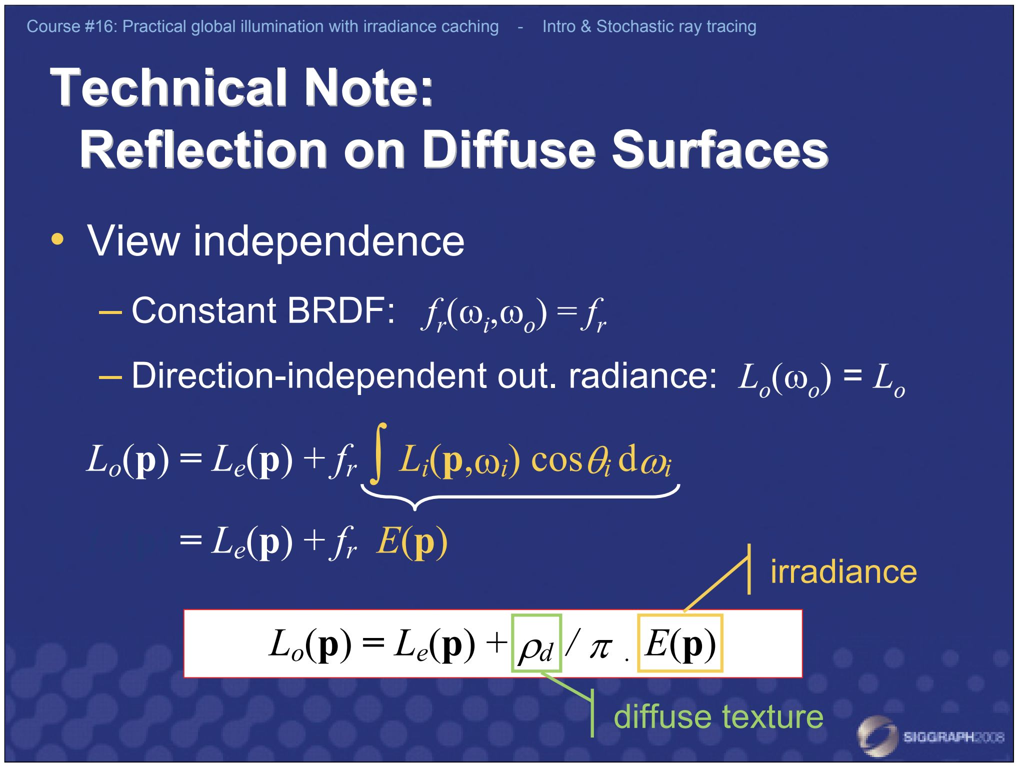


Course #16: Practical global illumination with irradiance caching - Intro & Stochastic ray tracing

Summary

- Distribution ray tracing slow
- Photon mapping breaks recursion
- Final gathering required for high-quality results
- Improving efficiency of final gathering
  - Importance sampling on glossy surfaces
  - Caching & interpolation on diffuse surfaces

SIGGRAPH2008



Technical Note: Reflection on Diffuse Surfaces

- View independence
  - Constant BRDF: $f_r(\omega_i,\omega_o) = f_r$
  - Direction-independent out. radiance: $L_o(\omega_o) = L_o$

$$L_o(\mathbf{p}) = L_e(\mathbf{p}) + f_r \underbrace{\int L_i(\mathbf{p},\omega_i)\cos\theta_i \,\mathrm{d}\omega_i}$$

$$= L_e(\mathbf{p}) + f_r\; E(\mathbf{p})$$

$$\boxed{L_o(\mathbf{p}) = L_e(\mathbf{p}) + \boxed{\rho_d} / \pi \,.\, \boxed{E(\mathbf{p})}}$$

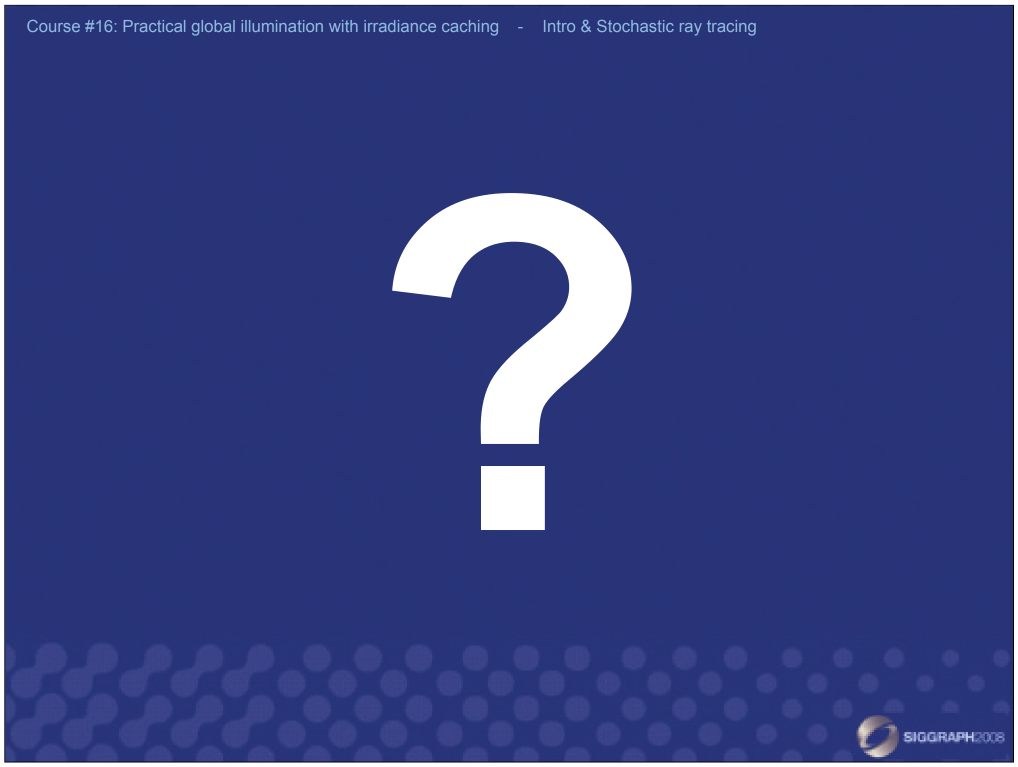
irradiance

diffuse texture

The view independence of diffuse materials means that the outgoing radiance $L_o(\omega_o)$ is independent of the direction, ω_o. This implies that the BRDF is constant in both the incoming and the outgoing directions and, in the illumination integral, it can be taken out of the integration. What remains is the integral of the incoming radiance multiplied by the cosine term – this is the irradiance *E*. Irradiance expresses the area density of light energy and is measured in Watts per square meter.

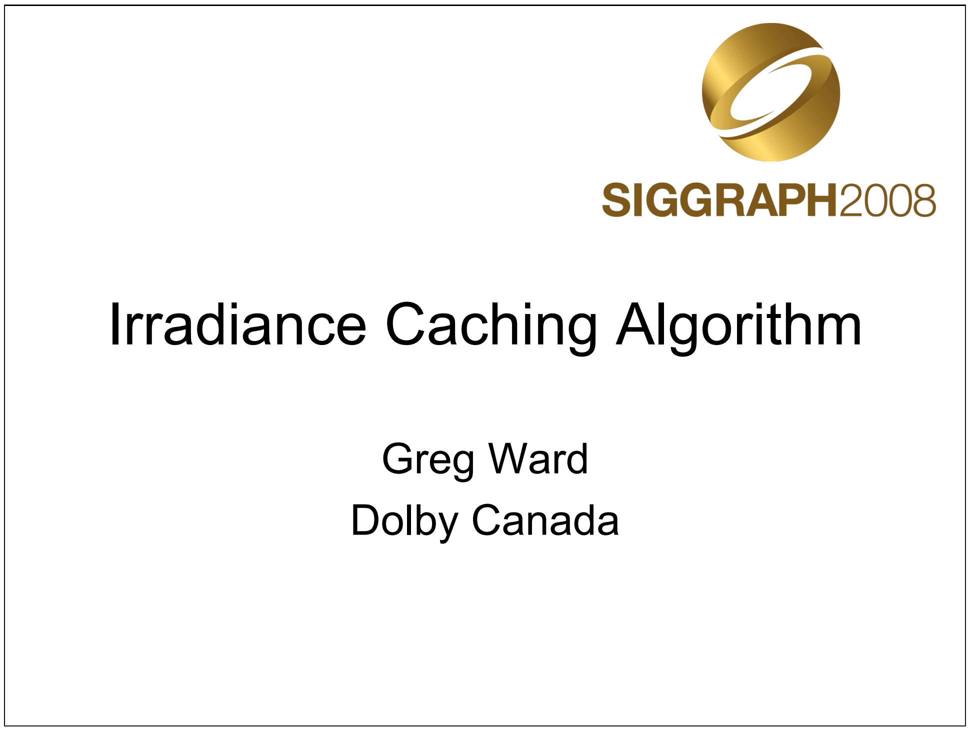
The BRDF of the diffuse surface is uniquely determined by one color triple – the reflectance ρ_d. Reflectance of a real surface must be between zero and one so that reflection conserves energy. This is why the radiometric quantity that corresponds to the diffuse texture is the reflectance.

Beware! Reflectance (=texture value) must be divided by π to get the BRDF value.



?

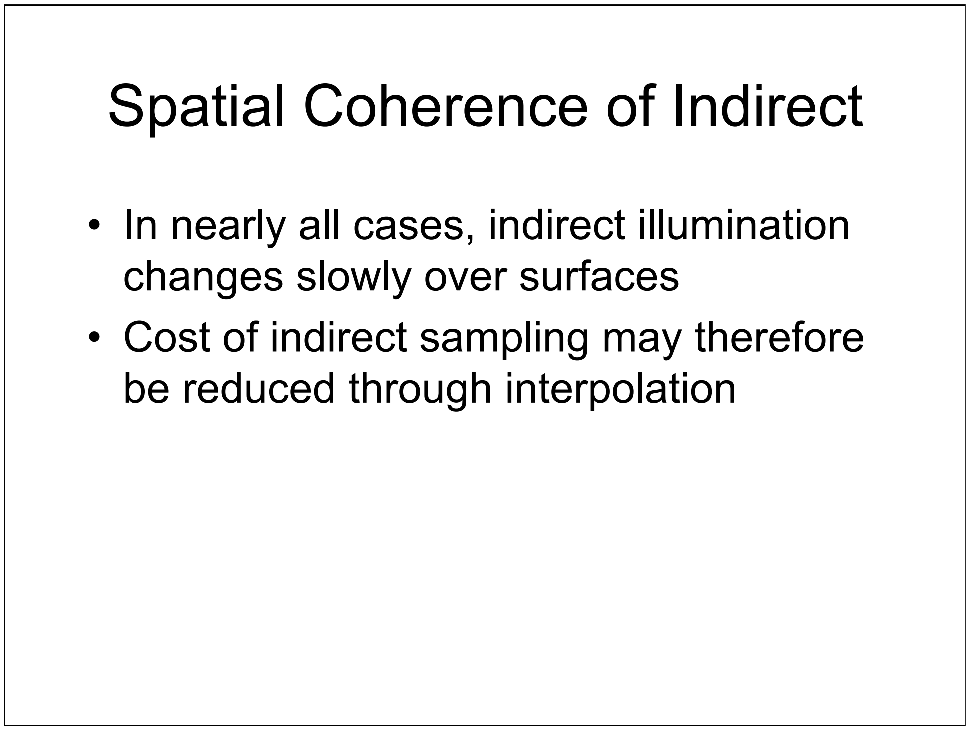
SIGGRAPH2008



SIGGRAPH2008

Irradiance Caching Algorithm

Greg Ward
Dolby Canada



Spatial Coherence of Indirect

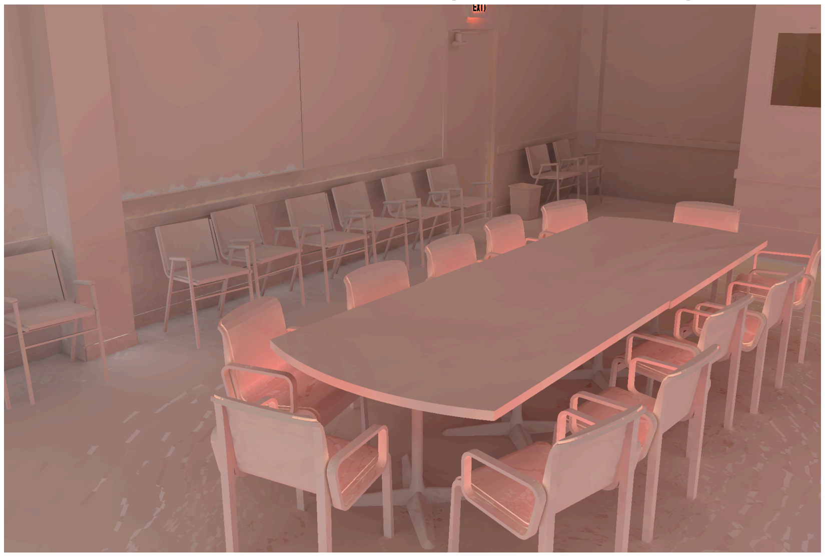
- In nearly all cases, indirect illumination changes slowly over surfaces
- Cost of indirect sampling may therefore be reduced through interpolation

Radiosity techniques work for the same reason -- and have trouble with point light sources because direct illumination does *not* change slowly over surfaces.

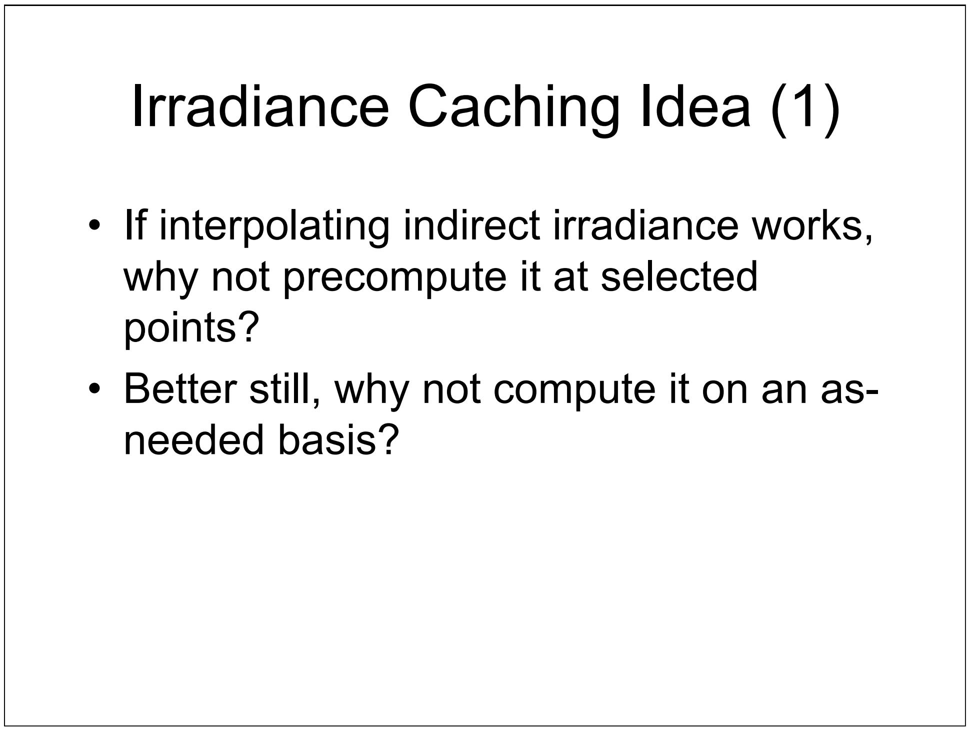
Two-bounce *Radiance* Rendering

Radiance is a physically-based renderer that has been around for over 20 years, and the irradiance cache was in the first public release in 1989.

Indirect Irradiance (two bounces)



Shown here is the indirect (RGB) irradiance, which changes slowly over flat surface areas, and more rapidly over curved regions. Notice the saturation of red near the chairs, though their final color is not shown. Depicted is only the light arriving at surfaces after one or more bounces.



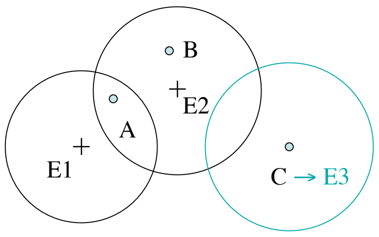
Irradiance Caching Idea (1)

- If interpolating indirect irradiance works, why not precompute it at selected points?
- Better still, why not compute it on an as-needed basis?

I didn't coin the term "irradiance caching" -- that was someone else's good idea. I just called it "lazy evaluation," being too lazy myself to think up a good name.

Irradiance Caching Idea (2)

- Lazy evaluation scheme
- Point A interpolates
- Point B extrapolates
- Point C calculates

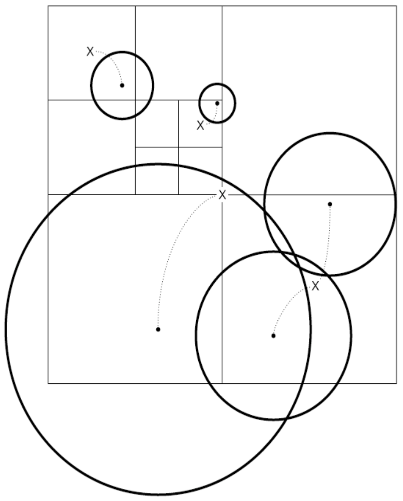


Q: How do we find nearby values?

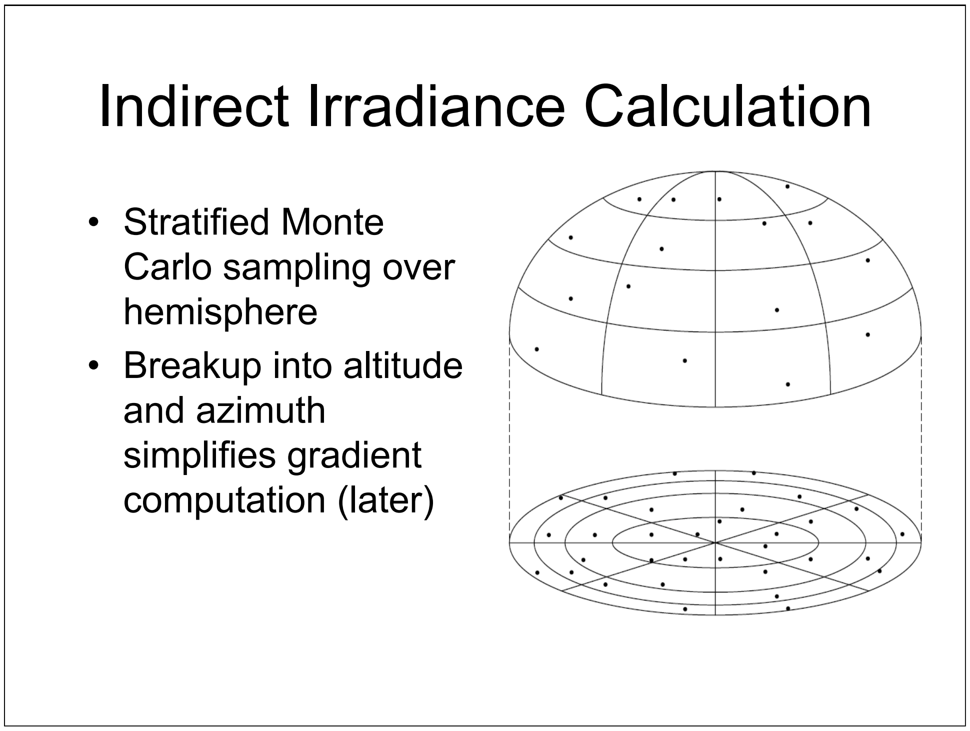
The E's are evaluation points. E1 and E2 shown here are previous evaluations, where E3 is a new evaluation triggered at query position C.

Octree Cache Lookup

- Each node contains list of indirect irrad. values with valid radii less than width
- Lookup proceeds down octree until all values within valid radius are found

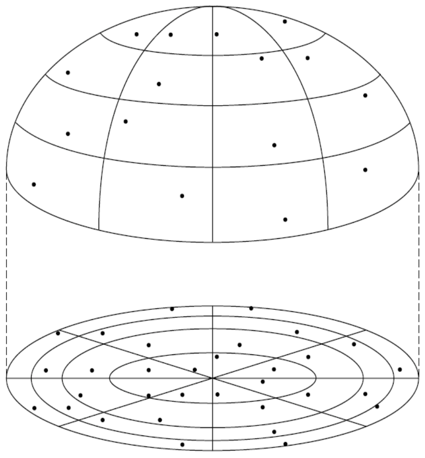


Using an octree that is independent of surfaces avoids placing restrictions on the geometric representation. It even works for volume data, sort of…



Indirect Irradiance Calculation

- Stratified Monte Carlo sampling over hemisphere
- Breakup into altitude and azimuth simplifies gradient computation (later)



Stratified sampling reduces noise in the results at no extra cost, without adding bias. Divisions are placed to maintain constant projected areas on unit circle (Nusselt analog).

Irradiance Computation

$$E_{ind} = \iint L_{ind}(\theta_i, \phi_i) \cos\theta_i \sin\theta_i d\theta_i d\phi_i$$

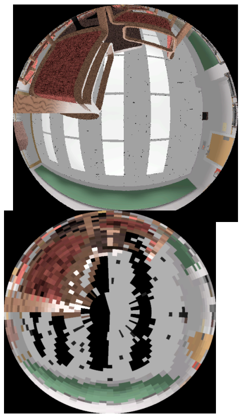
$$E = \left(\frac{\pi}{M \cdot N}\right) \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} L_{j,k}$$

$L_{j,k}$ is the indirect radiance in the direction $(\theta_j, \phi_k) = \left(\operatorname{asin}\sqrt{\frac{j+X_j}{M}}, 2\pi\frac{k+Y_k}{N}\right)$

Integral equation may be converted to equal-weighted sum of selected "measures" using standard Monte Carlo integration techniques (inverting cosine projection). X_j and Y_k are uniformly distributed random variables between 0 and 1. To compute indirect irradiance, we count as zero any samples that intersect a light source.

Indirect Irradiance Example

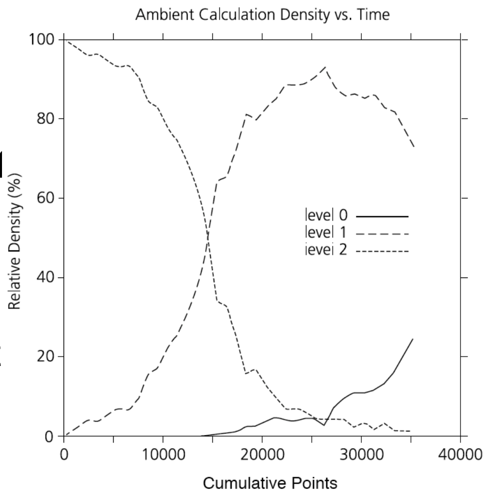
- Send hemisphere ray samples
- Do not count direct sources
- Deeper levels use fewer samples
- Main *Radiance* controls:
  - **-ab** number of bounces
  - **-ad**, **-as** samples per hemisphere
  - **-aa** interpolation accuracy
  - **-ar** spatial resolution



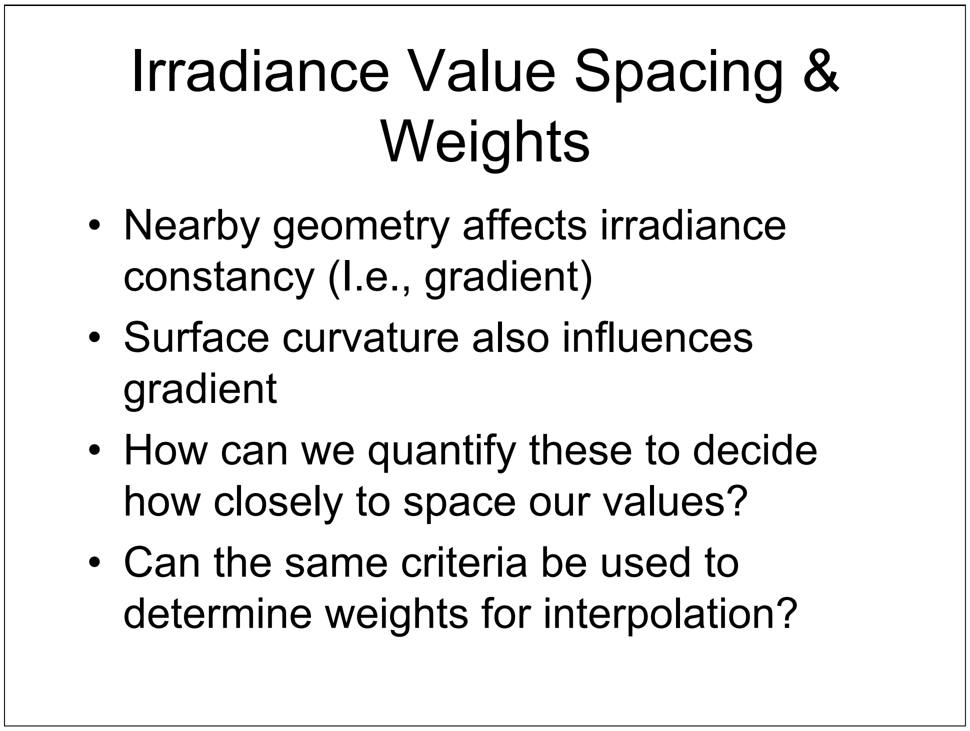
The top image shows a hemispherical equal-contribution projection from the floor near one of the chairs in the conference room model. The bottom image shows the actual shape and number of samples sent for a particular indirect irradiance computation. Note that the sources are black, and the sampling areas follow altitude and azimuth divisions. Actual sample placement within each area is random.

Irradiance Cache Growth

- At deeper levels in the ray tree, growth is limited by reuse of values
- Average of upper levels may be used to estimate constant "ambient" term



In a multi-bounce calculation, the deepest level is filled first, as the initial sample starts a hemisphere calculation, which begets another at the next level, and so on. The plot shows a three-bounce calculation, which starts by filling in the "level 2" cache, then spends time on the "level 1" cache before finally reaching the "level 0" cache -- the final indirect irradiance.



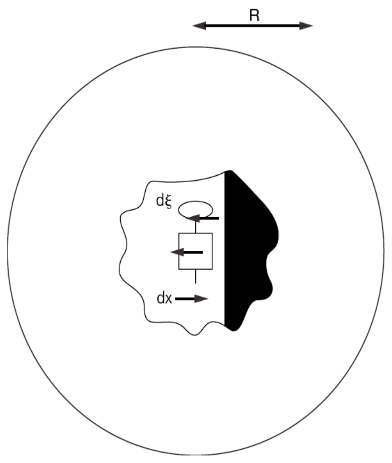
Irradiance Value Spacing & Weights

- Nearby geometry affects irradiance constancy (I.e., gradient)
- Surface curvature also influences gradient
- How can we quantify these to decide how closely to space our values?
- Can the same criteria be used to determine weights for interpolation?

We have an intuition about the behavior of indirect irradiance -- we need to translate this into an algorithm that is robust and reliable.

Split Sphere Approximation

- Target worst case
- Half light, half dark sphere
- Point at center, looking at break
- Q: How does irradiance change?



The “split sphere” is not technically a “worst case” -- it’s just a “pretty bad case.” It is better to qualify this as an assumption. I.e., we assume our actual environment will behave no worse than the split sphere. If it does, our technique won’t necessarily break, but it won’t be as accurate as we hoped, either.

Split Sphere Gradients

$$\varepsilon \le \left|\frac{\partial E}{\partial x}(x - x_0) + \frac{\partial E}{\partial \xi}(\xi - \xi_0)\right|$$

$$\varepsilon \le \frac{4}{\pi}\frac{E}{R}\left|x - x_0\right| + E\left|\xi - \xi_0\right|$$

$$\varepsilon(\vec{P}) \le \frac{4}{\pi}E\frac{\|\vec{P} - \vec{P}_0\|}{R_0} + E\sqrt{2 - 2\vec{N}(\vec{P}) \cdot \vec{N}(\vec{P}_0)}$$

R_0 = average distance to surfaces at $\vec{P}_0$

The first two inequalities bound the split sphere partial derivatives. The final equation uses these bounds to derive a “pretty bad case” error estimate for movement and rotation based on this first-order analysis. The calculation of R0 uses a harmonic mean to neighboring surfaces as measured by our ray samples. A harmonic mean is preferred since the radius appears in the denominator of our equation. Obviously, zero ray lengths are forbidden.

Generalized Error Estimate

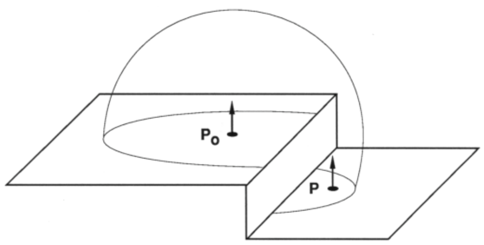
$$E(\vec{P}) = \frac{\sum_{i \in s} w_i(\vec{P}) E_i(\vec{P})}{\sum_{i \in s} w_i(\vec{P})}$$

$$w_i(\vec{P}) = \frac{1}{\frac{\|\vec{P} - \vec{P}_i\|}{R_i} + \sqrt{1 - \vec{N}(\vec{P}) \cdot \vec{N}(\vec{P}_i)}}$$

$$S = \{i : w_i(\vec{P}) > 1/a\}$$

Ignoring the constants from our previous equation, we can derive a weighting function based on the split sphere approximation. Since our weights correlate to 1/error, applying them in a weighted average distributes error uniformly between the interpolated irradiance values. We hope to maintain final accuracy by restricting our set of values to ones whose estimated error is below some user tolerance, “a”.

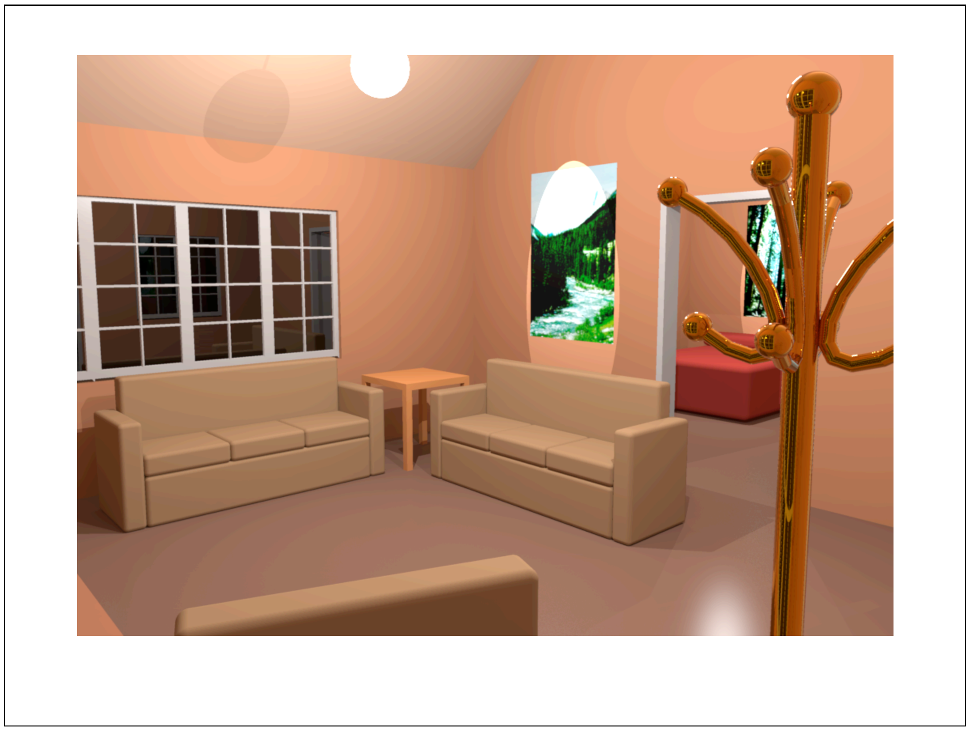
Additional Constraint



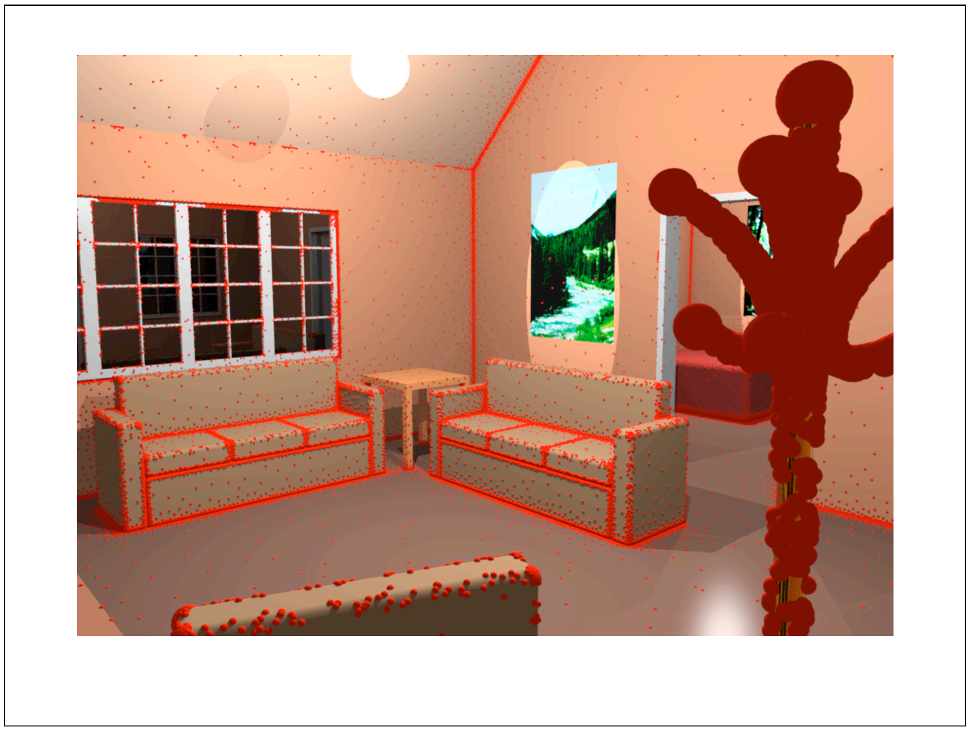
- Point ***P*** is within the valid radius of P_o, but is occluded by nearby geometry
- Assume surface of sphere to decide whether point is "behind" reference using distance vs. normals

$$\left(\vec{P}-\vec{P}_i\right)\cdot\left[\vec{N}(\vec{P})+\vec{N}(\vec{P}_i)\right]\big/2\geq 0$$

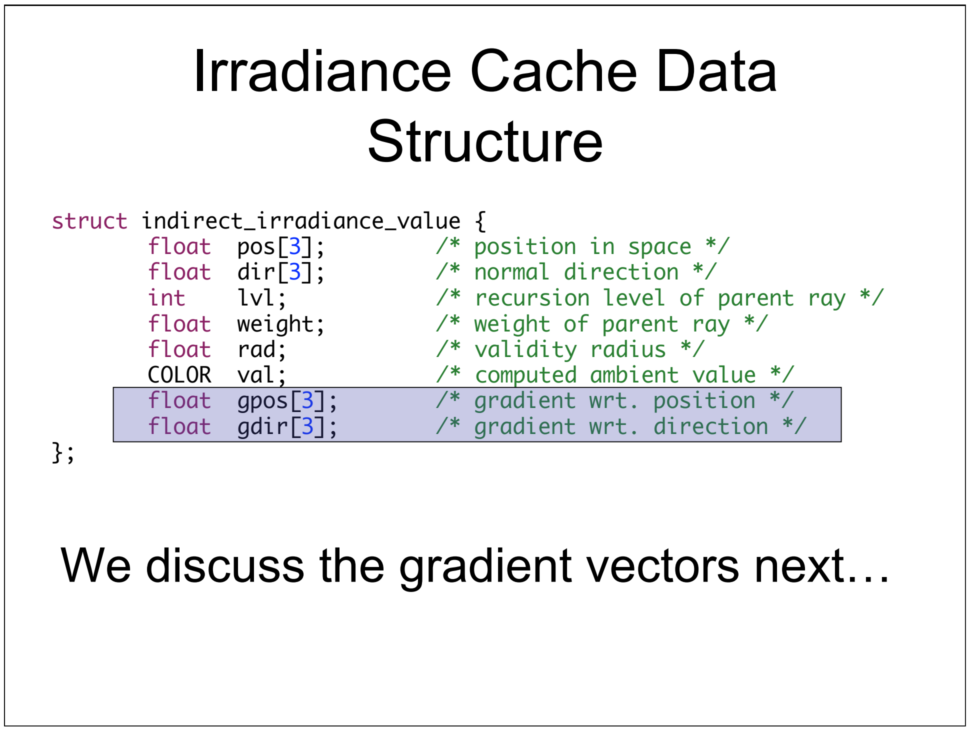
One special case we need to consider occurs when an irradiance value *thinks* it is valid over a larger region than it actually is. This case is depicted in the figure, where we are thinking about using an irradiance value at P0 at the new query position P. We can avoid this problem condition by adding a "behind test" to our criteria, which via the weighting function already incorporates a curvature test and a relative distance test.



Cabin model showing single-bounce calculation.



Indirect irradiance record placement shown as red spheres. Note how record spacing is large on flat surfaces far from neighboring geometry, then bunches up on outside curves and (especially) inside corners.

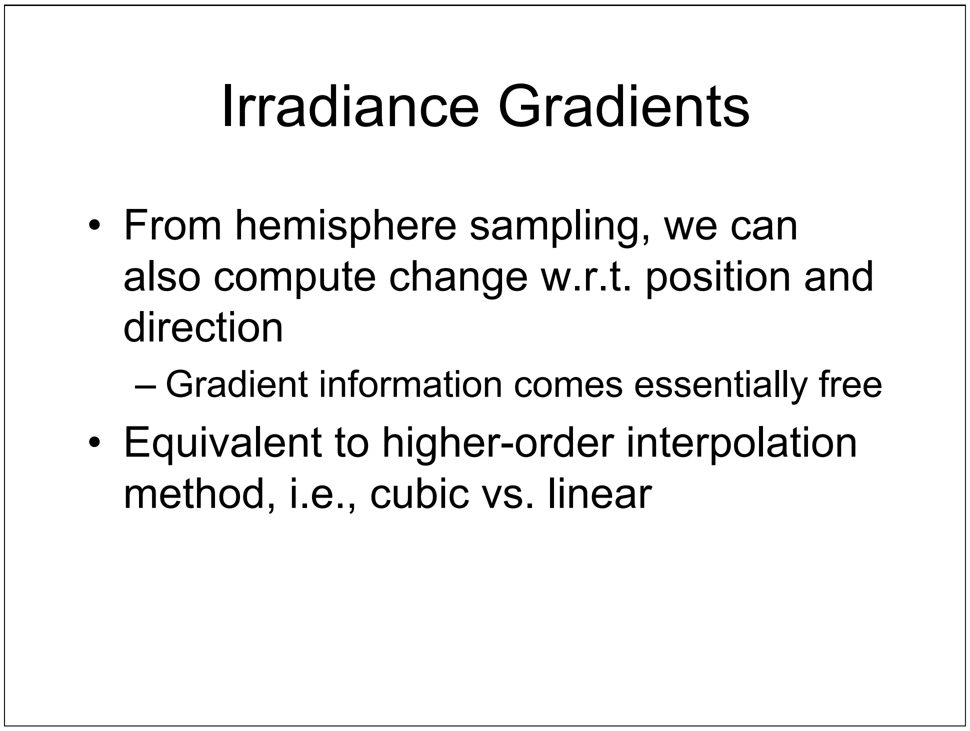


Irradiance Cache Data Structure

```
struct indirect_irradiance_value {
        float  pos[3];         /* position in space */
        float  dir[3];         /* normal direction */
        int    lvl;            /* recursion level of parent ray */
        float  weight;         /* weight of parent ray */
        float  rad;            /* validity radius */
        COLOR  val;            /* computed ambient value */
        float  gpos[3];        /* gradient wrt. position */
        float  gdir[3];        /* gradient wrt. direction */
};
```

We discuss the gradient vectors next…

The ray level and weight are used to determine when to truncate the ray tree. The position and normal vectors are used in the weight/threshold computation, together with the “validity radius.”



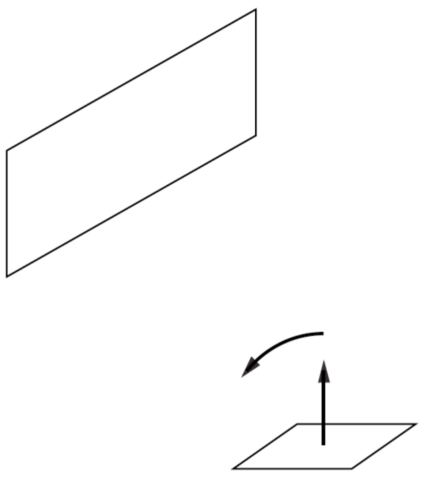
Irradiance Gradients

- From hemisphere sampling, we can also compute change w.r.t. position and direction
  - Gradient information comes essentially free
- Equivalent to higher-order interpolation method, i.e., cubic vs. linear

Initially, Paul Heckbert and I were thinking a gradient calculation could inform better value spacing. A year went by before we realized that it was better to apply them directly during interpolation to reduce discontinuities. This also avoids bias issues.

Rotational Gradient

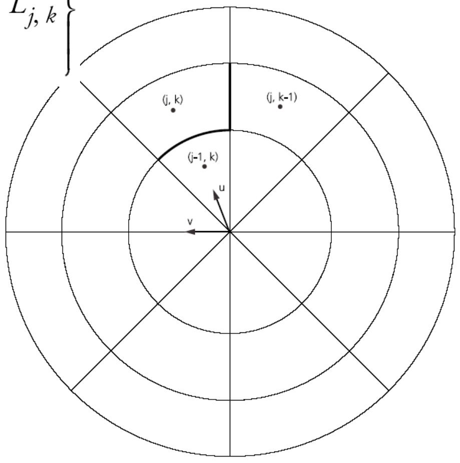
- As view rotates, surface sees more (or less) of bright object
- Estimate rotational change based on hemisphere samples



Caveat: we cannot know what will appear over the horizon.

Rotational Gradient Formula

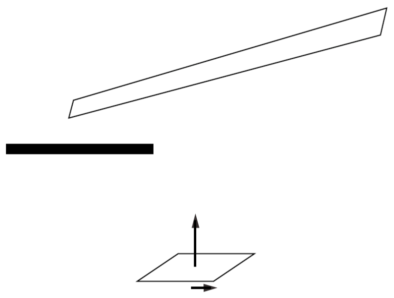
$$\vec{\nabla}_r E = \frac{\pi}{M \cdot N} \sum_{k=0}^{N-1} \left\{ \hat{v}_k \sum_{j=0}^{M-1} -\tan\theta_j \cdot L_{j,k} \right\}$$



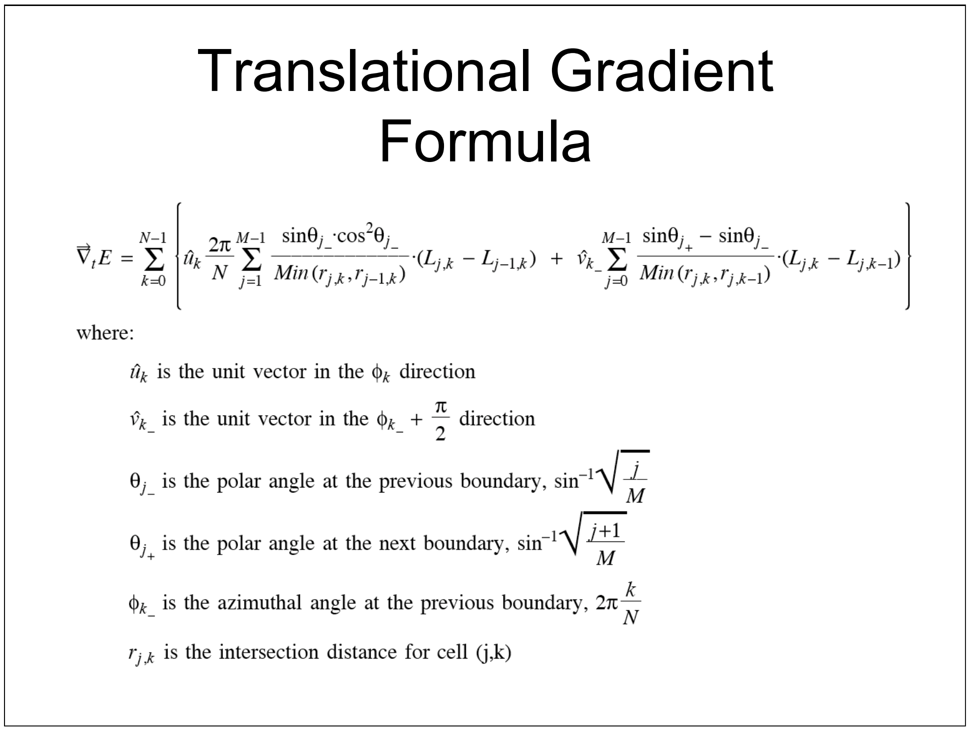
The rotational gradient is zero at the zenith because the derivative of the cosine is zero, hence small rotations don’t affect contribution for samples looking straight up.

Translational Gradient

- Translation of surface element exposes (or hides) bright occluded objects and shifts boundaries
- Estimate changes using hemisphere



The distance to our sampled geometry is important, and we have to assume that disocclusions look like what we already see of the background object. Hence, the boundary moves as the nearer edge moves rather than the further surface.



Translational Gradient Formula

$$\vec{\nabla}_t E = \sum_{k=0}^{N-1} \left\{ \hat{u}_k \frac{2\pi}{N} \sum_{j=1}^{M-1} \frac{\sin\theta_{j\_} \cdot \cos^2\theta_{j\_}}{Min(r_{j,k}, r_{j-1,k})} \cdot (L_{j,k} - L_{j-1,k}) \; + \; \hat{v}_{k\_} \sum_{j=0}^{M-1} \frac{\sin\theta_{j_+} - \sin\theta_{j\_}}{Min(r_{j,k}, r_{j,k-1})} \cdot (L_{j,k} - L_{j,k-1}) \right\}$$

where:

$\hat{u}_k$ is the unit vector in the ϕ_k direction

$\hat{v}_{k\_}$ is the unit vector in the $\phi_{k\_} + \frac{\pi}{2}$ direction

$\theta_{j\_}$ is the polar angle at the previous boundary, $\sin^{-1}\sqrt{\frac{j}{M}}$

θ_{j_+} is the polar angle at the next boundary, $\sin^{-1}\sqrt{\frac{j+1}{M}}$

$\phi_{k\_}$ is the azimuthal angle at the previous boundary, $2\pi\frac{k}{N}$

$r_{j,k}$ is the intersection distance for cell (j,k)

Refer to the vector diagram on the rotational gradient slide for variable definitions. Here, the ‘+’ and ‘-’ suffixes refer to the vectors (not drawn) corresponding to the leading and trailing edges of each cell, respectively. The Min(a,b) function is used to determine the closer of two neighboring surface samples.

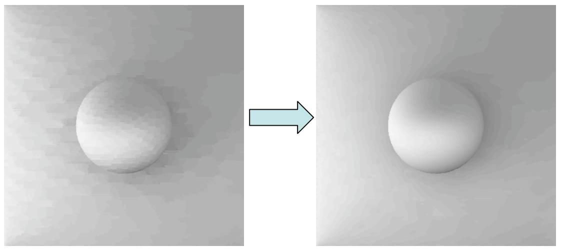
Gradient Interpolation

$$E(\vec{P}) = \frac{\sum_{\mathbf{S}} w_i(\vec{P}) \left[E_i + (\hat{n}_i \times \hat{n}) \cdot \vec{\nabla}_r E_i + (\vec{P} - \vec{P}_i) \cdot \vec{\nabla}_t E_i \right]}{\sum_{\mathbf{S}} w_i(\vec{P})}$$

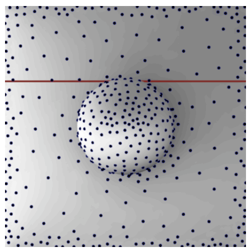
- Weights $w_i(\mathbf{P})$ same as before
- Essentially modifies E_i's used for interpolation
- Gradient also used to cap valid radii

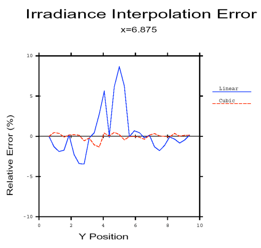
Once we have our rotational and translational gradients corresponding to each irradiance value, we can apply them in a first-order, weighted interpolation as shown.

Irradiance Gradient Results

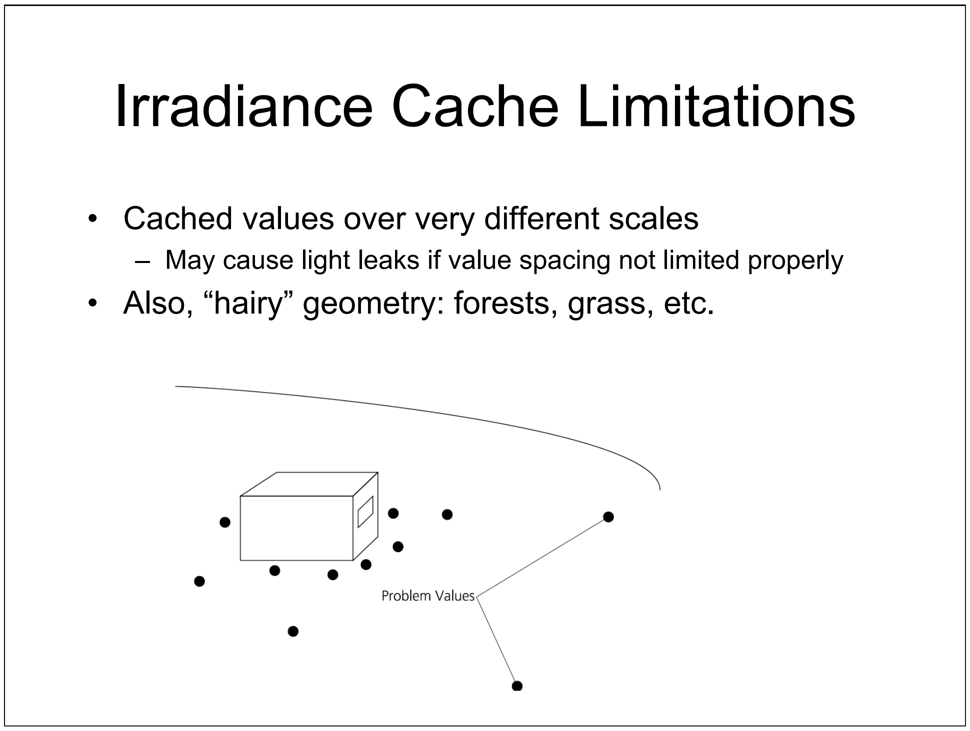


(no overture)



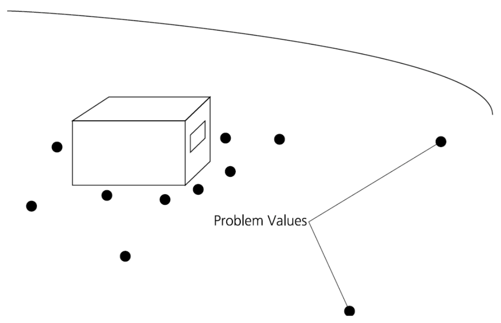


The effect of applying the irradiance gradient technique is highlighted in this extrapolated calculation, where lazy evaluation is shown in its “full glory.” Despite discontinuities introduced by adding values to the cache during scanline rendering, our first-order extrapolation makes these artifacts all but disappear. Value placement is shown together with the comparative error when applying gradients to interpolation (I.e., adding an overture calculation).

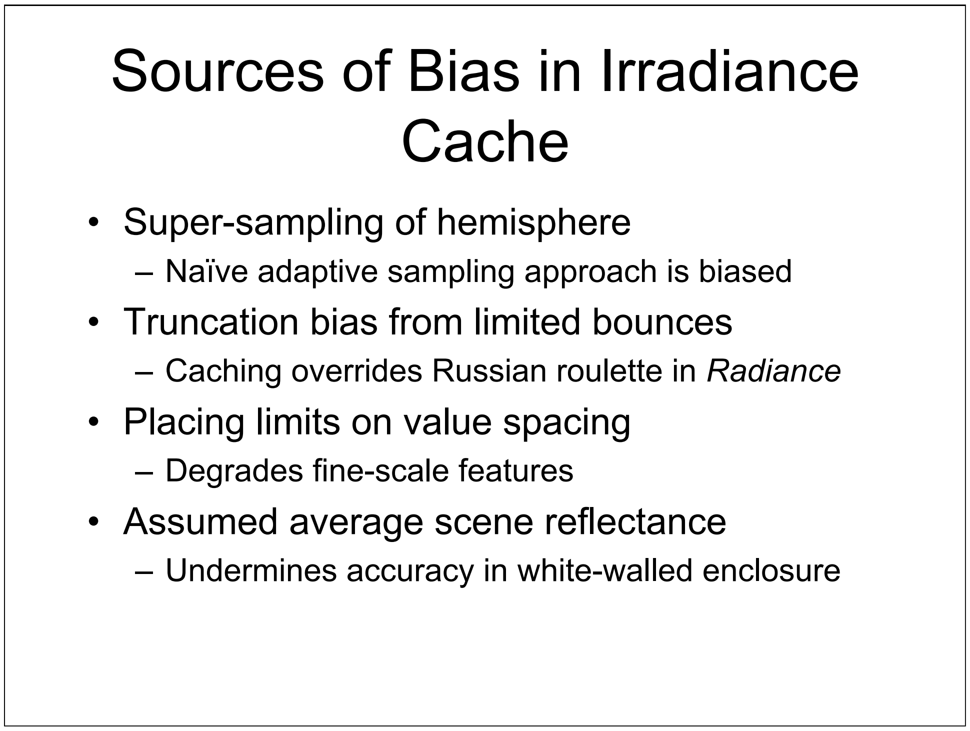


Irradiance Cache Limitations

- Cached values over very different scales
  - May cause light leaks if value spacing not limited properly
- Also, "hairy" geometry: forests, grass, etc.



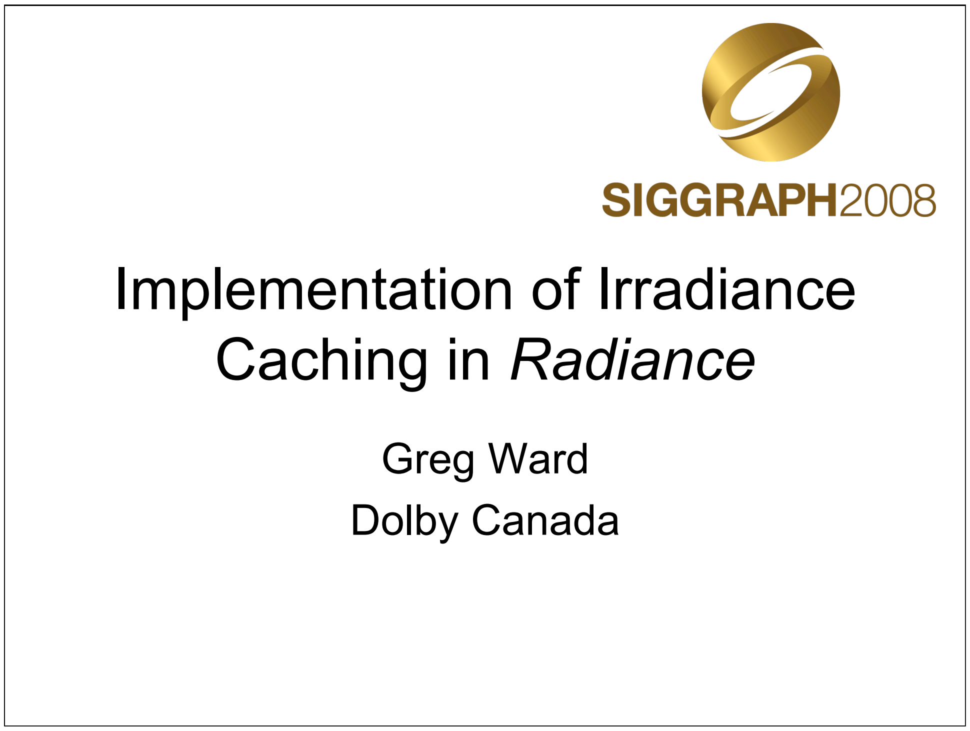
Jaroslav has a solution called "neighbor clamping" for this problem. Hairy geometry is best dealt with by switching to a noisy MCPT method for busy topologies.



Sources of Bias in Irradiance Cache

- Super-sampling of hemisphere
  - Naïve adaptive sampling approach is biased
- Truncation bias from limited bounces
  - Caching overrides Russian roulette in *Radiance*
- Placing limits on value spacing
  - Degrades fine-scale features
- Assumed average scene reflectance
  - Undermines accuracy in white-walled enclosure

To eliminate bias, don’t use adaptive super-sampling, add Russian roulette to final bounce via path tracing, eliminate minimum value spacing (expensive) and use actual surface reflectances (undermines sharing). Decreasing number of hemisphere samples (by reflectance) and increasing sample spacing (by reflectance^-0.5) distributes errors evenly between levels and speeds convergence.



SIGGRAPH2008

Implementation of Irradiance Caching in *Radiance*

Greg Ward
Dolby Canada

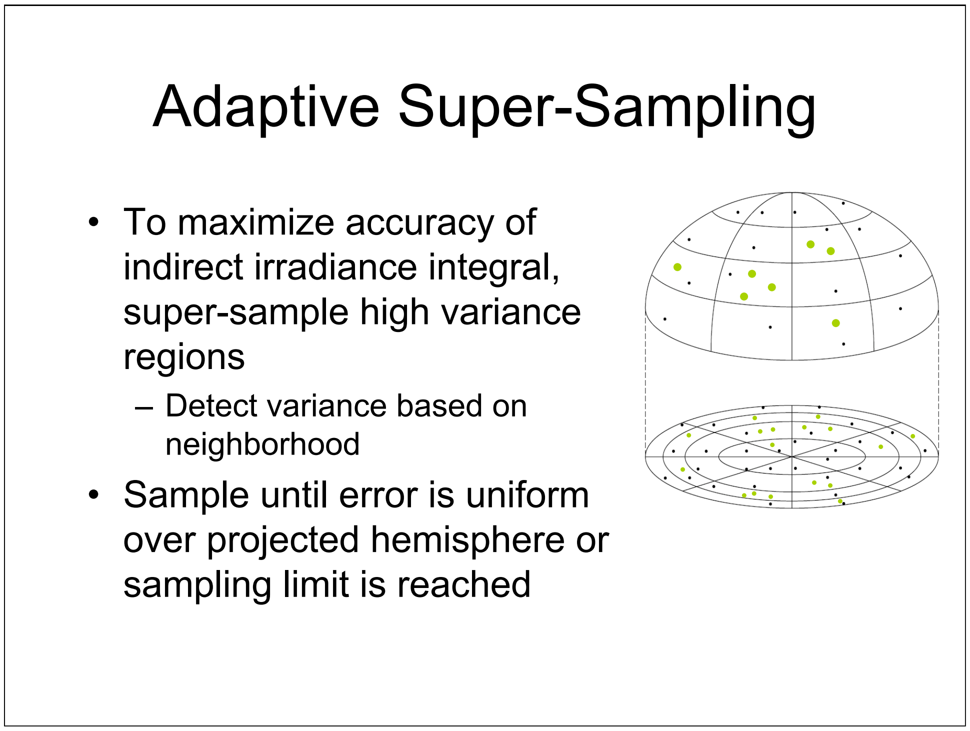
Irradiance Cache Enhancements in *Radiance*

1. Computation of ambient “constant”
2. Adaptive super-sampling on hemisphere
3. Maximum and minimum record spacing
4. Gradient limit on record spacing
5. Bump maps using rotation gradient
6. Options for excluding surfaces/materials
7. Record sharing for multiprocessors
8. Record sorting for cache coherency

Computation of Ambient “Constant”

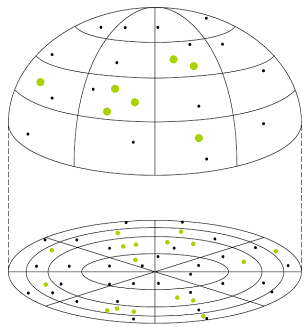
- So-called “ambient term” approximates the remainder of an infinite series
- An average of top-level indirect irradiances is a good approximation
- A moving average may be used as the irradiance cache is filled over time
- Over-estimating ambient term is worse than under-estimating

The -aw option in Radiance controls this function. As indirect values are collected, they overtake the initial, user-specified ambient term. It’s one of those features that seemed like a good idea at the time, but in practice it doesn’t get much use.



Adaptive Super-Sampling

- To maximize accuracy of indirect irradiance integral, super-sample high variance regions
  - Detect variance based on neighborhood
- Sample until error is uniform over projected hemisphere or sampling limit is reached

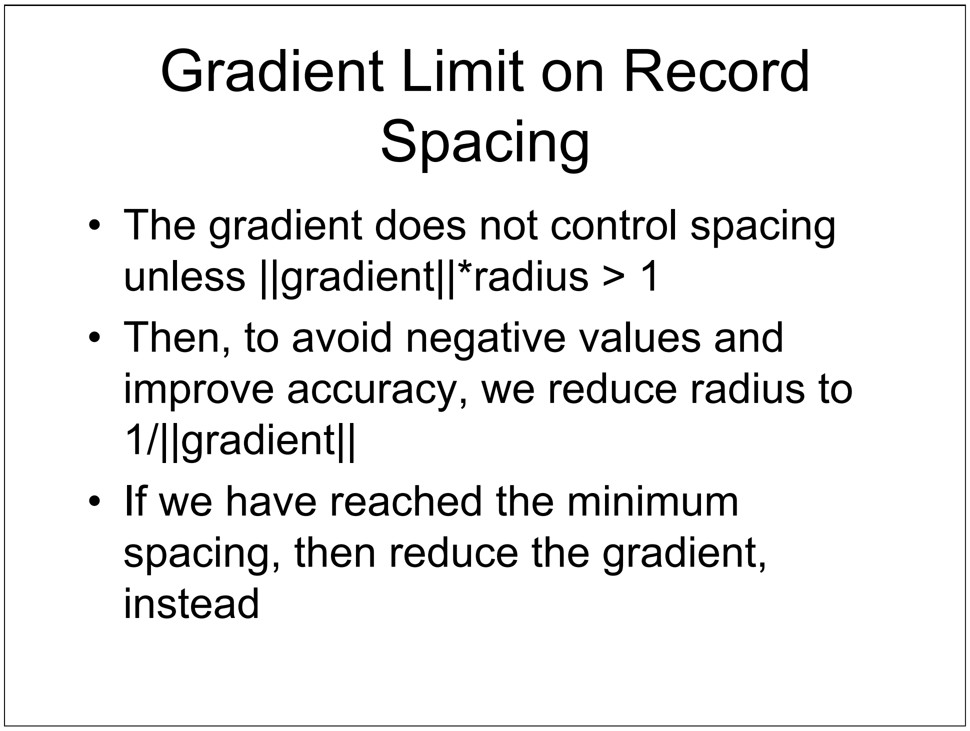


Controlled by the -as option in Radiance, this often-used optimization is especially effective in bright, daylighted interiors.

Maximum and Minimum Record Spacing

- Without a minimum record spacing, inside corners get resolved to a pixel level
- Applying a minimum spacing, accuracy gradually rolls off at a certain scene scale
- A maximum value spacing of 64 times the minimum spacing seems about right

If available, ray pixel size may also be used to adjust record spacing, though this would tend to undermine view independence.



Gradient Limit on Record Spacing

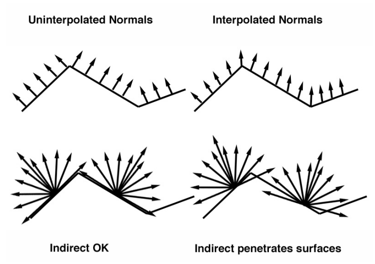
- The gradient does not control spacing unless ||gradient||*radius > 1
- Then, to avoid negative values and improve accuracy, we reduce radius to 1/||gradient||
- If we have reached the minimum spacing, then reduce the gradient, instead

It is important *not* to use the gradient to determine record spacing in general, as we cannot know what the gradient is before sampling, and we don’t know how often to sample if the gradient dictates spacing. It is better to use a conservative metric that doesn’t depend on actual scene radiances, which may or may not behave as expected.

Bump Maps Using Rotation Gradient

- Bumpy surfaces reduce record sharing
- Ignoring bump map, we can apply rotation gradient for irradiance just calculated
- This promotes optimal reuse and spacing

It also avoids the problem of sample leaks

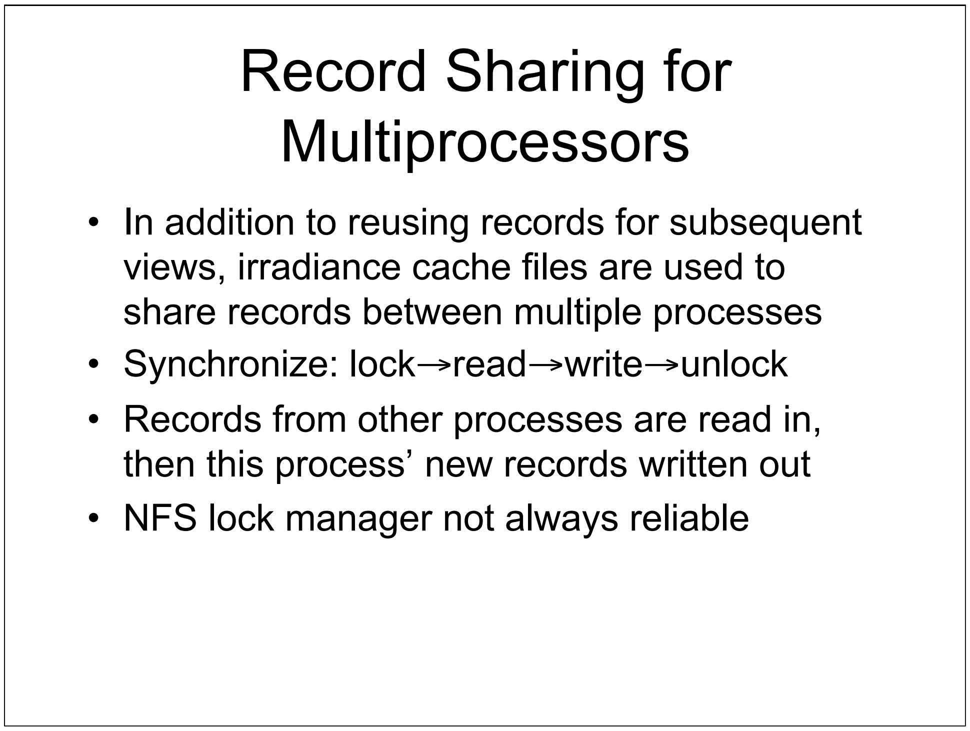


No-cost addition to irradiance gradient calculation -- just apply the perturbed surface normals on the final interpolation.

Options for Excluding Materials

- User-selected materials (and the surfaces they modify) may be excluded from indirect
- This saves hours of pointless interreflection calculations in fields of grass, etc.
- If only a few materials are to be included, an include list may be specified, instead
- It would be better to have a second type of interreflection calculation available

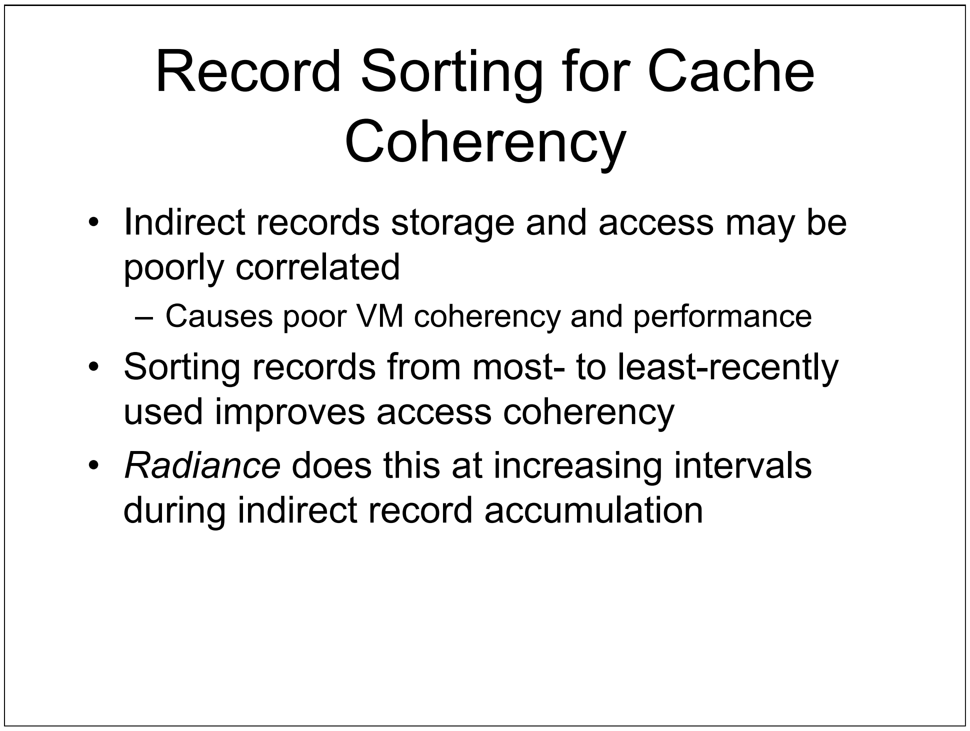
The -ae option excludes a single material (and its assigns), whereas -aE excludes all materials listed in a file. The -ai and -aI options may be used to specify included materials, instead.



Record Sharing for Multiprocessors

- In addition to reusing records for subsequent views, irradiance cache files are used to share records between multiple processes
- Synchronize: lock→read→write→unlock
- Records from other processes are read in, then this process' new records written out
- NFS lock manager not always reliable

This technique works with a fixed buffer size of about 13 records up to 10 processors or so, then a larger buffer works better. At some point, I would like to implement a client-server or broadcast model to reduce overhead and avoid problems with NFS.



Record Sorting for Cache Coherency

- Indirect records storage and access may be poorly correlated
  - Causes poor VM coherency and performance
- Sorting records from most- to least-recently used improves access coherency
- *Radiance* does this at increasing intervals during indirect record accumulation

There are additional caveats in cases where indirect values are being shared. Since a field is stored in each record for when it was last used, and changing this would undermine value sharing, we turn this optimization off when more than a third of the records are being shared across processes. The biggest savings occur in processes that exceed physical memory, which causes VM thrashing.

Conclusions

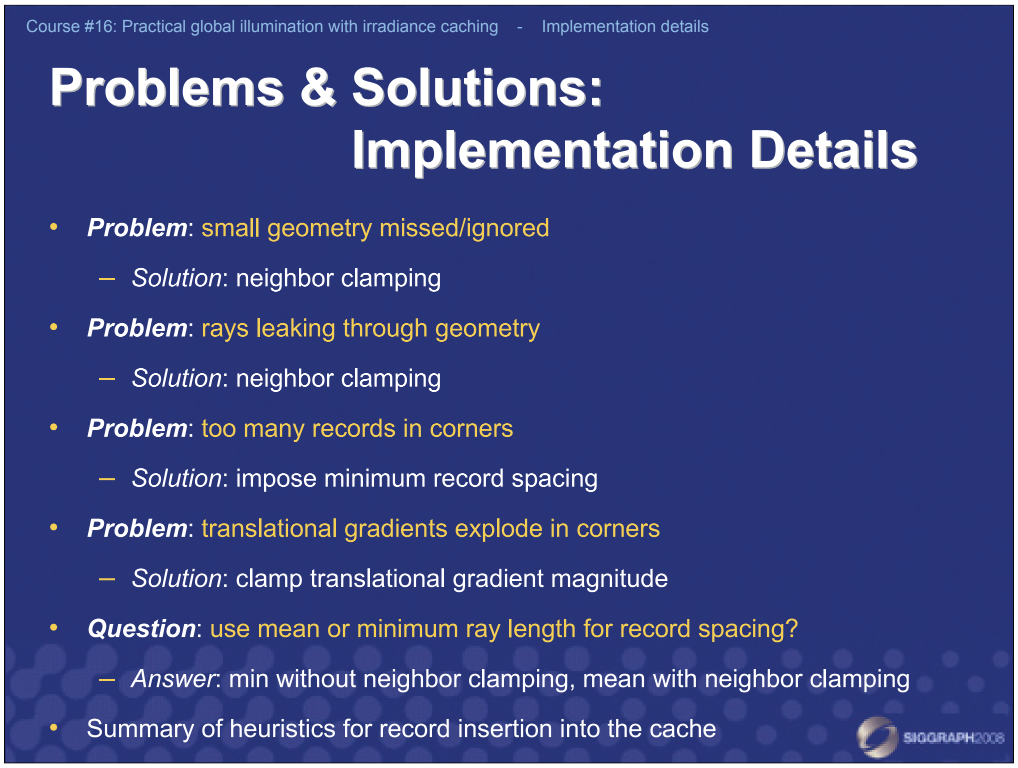
- Irradiance cache was implemented in *Radiance* around 1986
- First SIGGRAPH submission was rejected, and paper was completely rewritten (twice)
- Refinements have been few and subtle
- C code is about 1500 lines of 30,000 in *Radiance* rendering engine

Problems & Solutions: Implementation details

SIGGRAPH2008

Jaroslav Křivánek

Czech Technical University in Prague



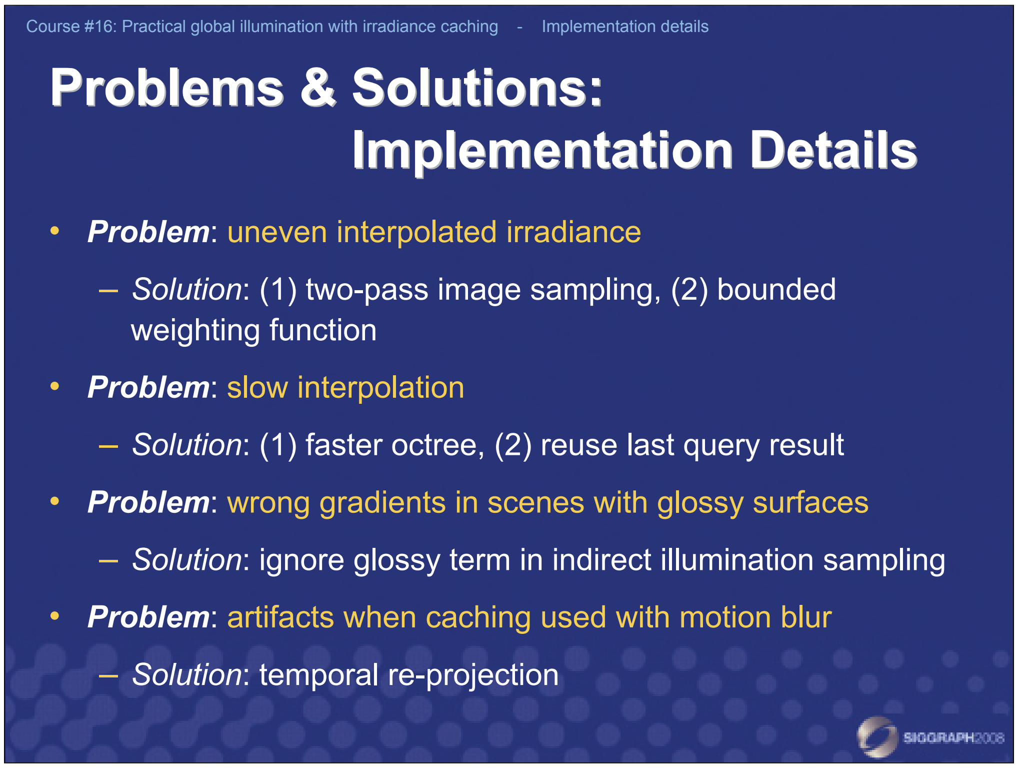
Course #16: Practical global illumination with irradiance caching - Implementation details

Problems & Solutions: Implementation Details

- ***Problem***: small geometry missed/ignored
  - *Solution*: neighbor clamping
- ***Problem***: rays leaking through geometry
  - *Solution*: neighbor clamping
- ***Problem***: too many records in corners
  - *Solution*: impose minimum record spacing
- ***Problem***: translational gradients explode in corners
  - *Solution*: clamp translational gradient magnitude
- ***Question***: use mean or minimum ray length for record spacing?
  - *Answer*: min without neighbor clamping, mean with neighbor clamping
- Summary of heuristics for record insertion into the cache

SIGGRAPH2008

This part of the course summarizes the common problems encountered when implementing irradiance caching. For each of the problems we describe one or more possible solutions to alleviate it. It should be stressed that this part describes the solutions used in my own implementation but slightly different solutions may be used in other implementations. Various tricks used at PDI/Dreamworks are described by Eric Tabellion.

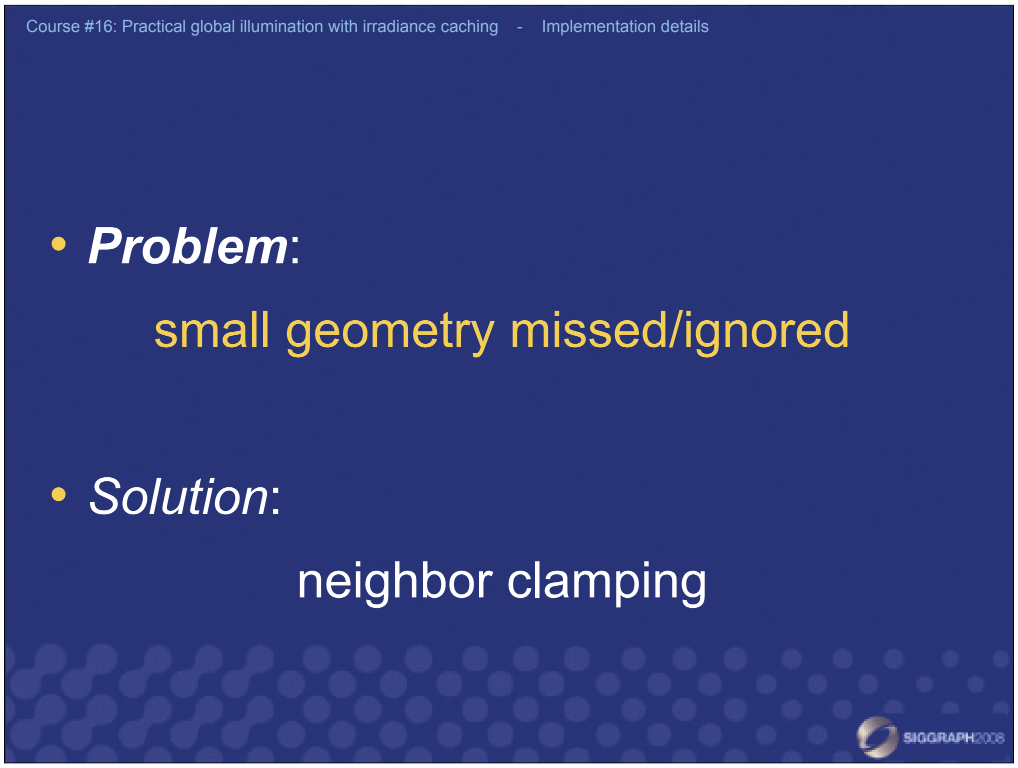


Course #16: Practical global illumination with irradiance caching - Implementation details

Problems & Solutions: Implementation Details

- ***Problem***: uneven interpolated irradiance
  - *Solution*: (1) two-pass image sampling, (2) bounded weighting function
- ***Problem***: slow interpolation
  - *Solution*: (1) faster octree, (2) reuse last query result
- ***Problem***: wrong gradients in scenes with glossy surfaces
  - *Solution*: ignore glossy term in indirect illumination sampling
- ***Problem***: artifacts when caching used with motion blur
  - *Solution*: temporal re-projection

SIGGRAPH2008



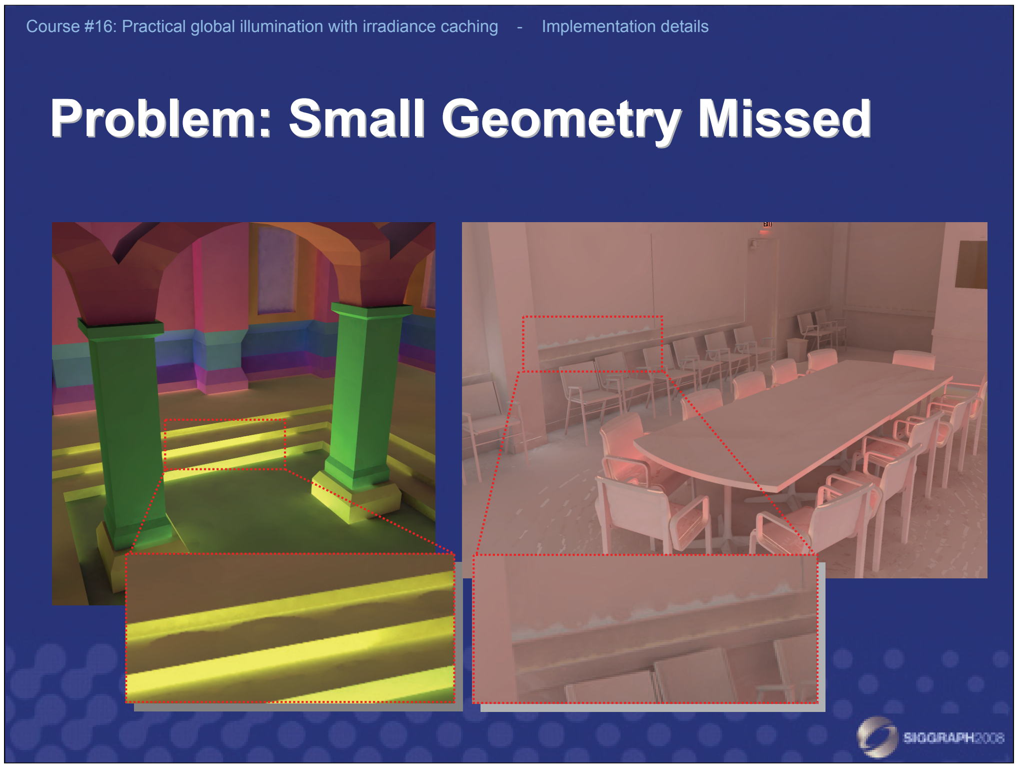
Course #16: Practical global illumination with irradiance caching - Implementation details

- ***Problem***:

  small geometry missed/ignored

- *Solution*:

  neighbor clamping

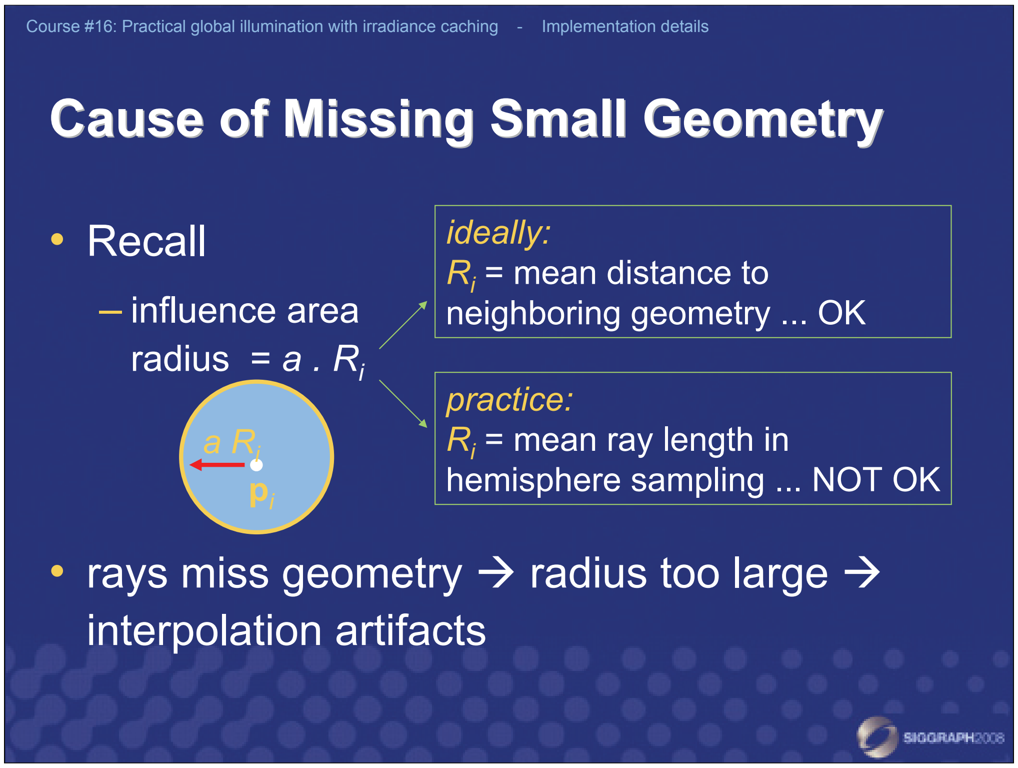


Course #16: Practical global illumination with irradiance caching - Implementation details

Problem: Small Geometry Missed

SIGGRAPH2008

A common problem in irradiance caching is that rays in the stochastic hemisphere sampling miss geometry features in the scene. This can produce clearly visible image artifacts, most often due to the exaggerated validity radius of some records.



Course #16: Practical global illumination with irradiance caching - Implementation details

Cause of Missing Small Geometry

- Recall
  - influence area radius = $a \cdot R_i$

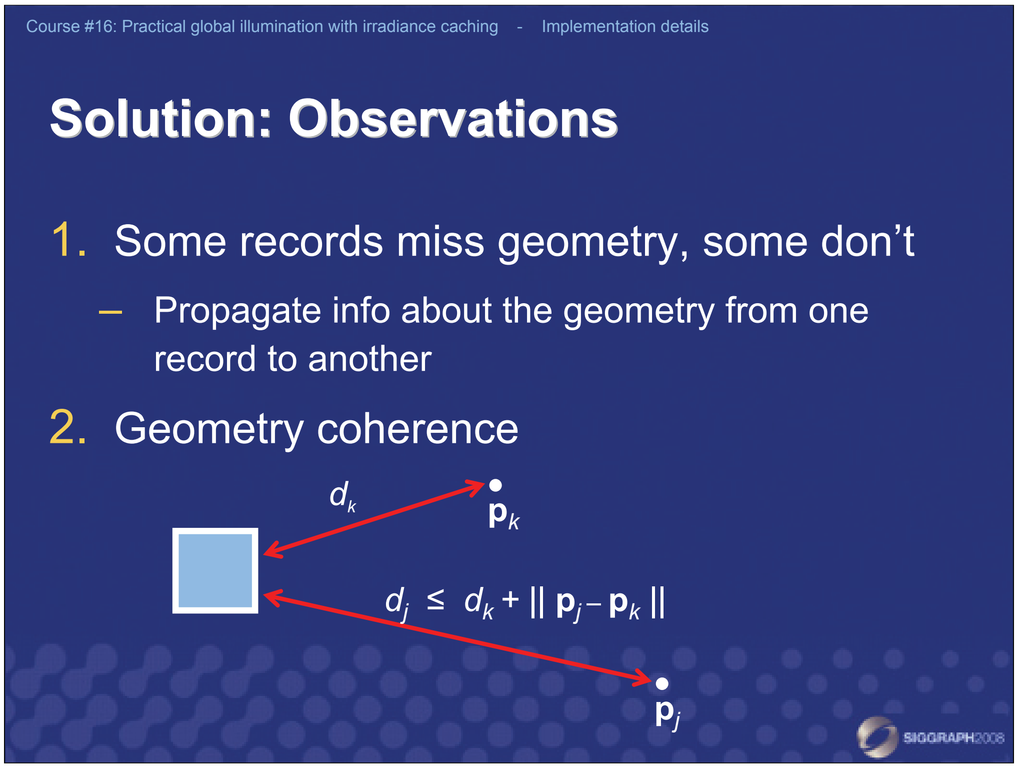
ideally:
R_i = mean distance to neighboring geometry ... OK

practice:
R_i = mean ray length in hemisphere sampling ... NOT OK

- rays miss geometry → radius too large → interpolation artifacts

$a\ R_i$
$\mathbf{p}_i$
SIGGRAPH2008

Ideally, record spacing would be determined by the mean distance to the neighboring geometry. In practice, this distance is determined as the mean of the ray lengths in hemisphere sampling. If rays miss some geometry, the resulting mean distance is overestimated and we can see discontinuities in the resulting images due to interpolation.



Course #16: Practical global illumination with irradiance caching - Implementation details

Solution: Observations

1. Some records miss geometry, some don't
   – Propagate info about the geometry from one record to another
2. Geometry coherence

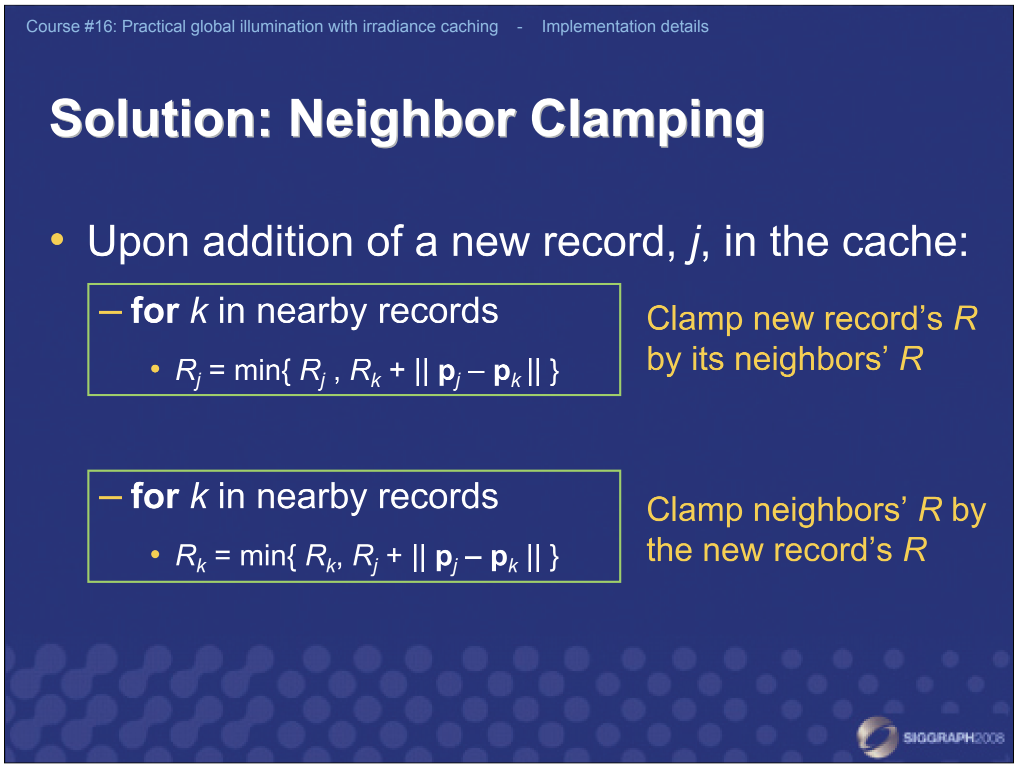
d_k
$\mathbf{p}_k$
$d_j \leq d_k + \| \mathbf{p}_j - \mathbf{p}_k \|$
$\mathbf{p}_j$
SIGGRAPH2008

A reliable way of detecting the over-estimated mean distance due to missing geometry in hemisphere sampling is based on two observations.

1.Because only few records usually suffer from the overestimated mean distance, we can use the distance estimate at other records to rectify the overestimated distance.

2.Distances obey the triangle inequality. If one record, $\mathbf{p}_k$, is at the distance d_k from some geometry feature, then another record, $\mathbf{p}_j$, cannot be farther from this geometry feature than $d_k + \| \mathbf{p}_j - \mathbf{p}_k \|$.

If we replace d_k and d_j by the mean distance R_k, R_j, respectively, we can detect suspicious cases by comparing R_k, R_j to the distance between the two records to verify if the triangle inequality holds. Strictly speaking, this should only work if R_k, R_j, was the distance to the *nearest* geometry feature, but in practice this works fine even for *mean* distance. We call this heuristic 'neighbor clamping'.



Solution: Neighbor Clamping

- Upon addition of a new record, *j*, in the cache:

```
– for k in nearby records
    • $R_j = \min\{ R_j , R_k + || \mathbf{p}_j – \mathbf{p}_k || \}$
```

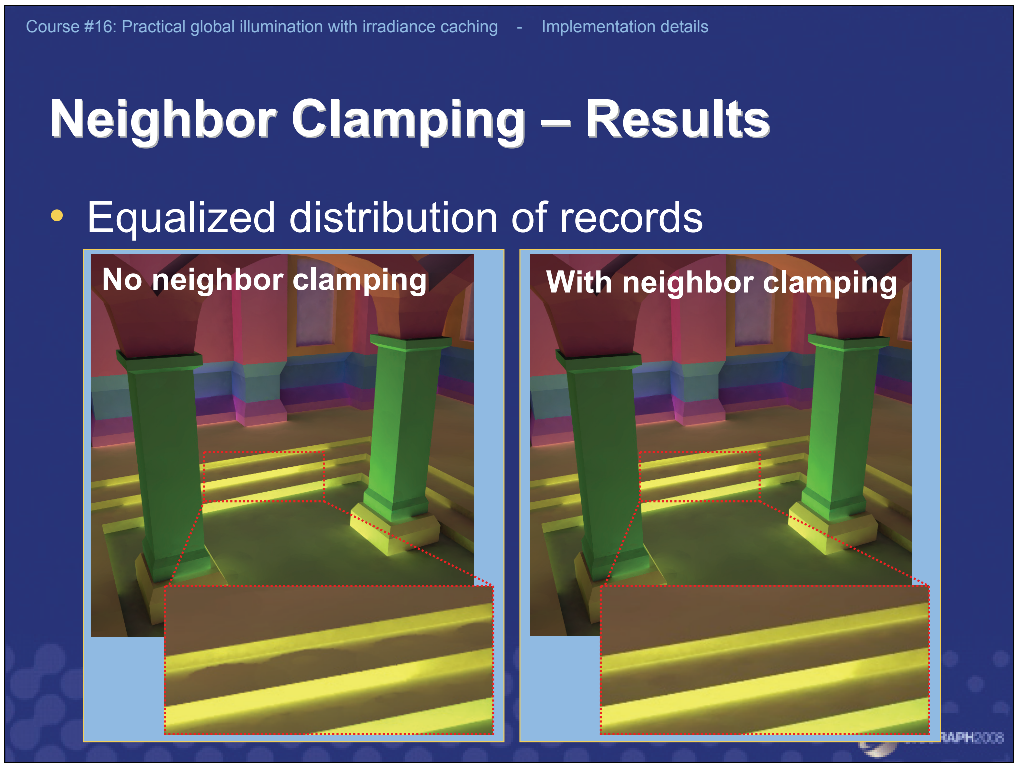
Clamp new record's *R* by its neighbors' *R*

```
– for k in nearby records
    • $R_k = \min\{ R_k, R_j + || \mathbf{p}_j – \mathbf{p}_k || \}$
```

Clamp neighbors' *R* by the new record's *R*

SIGGRAPH2008

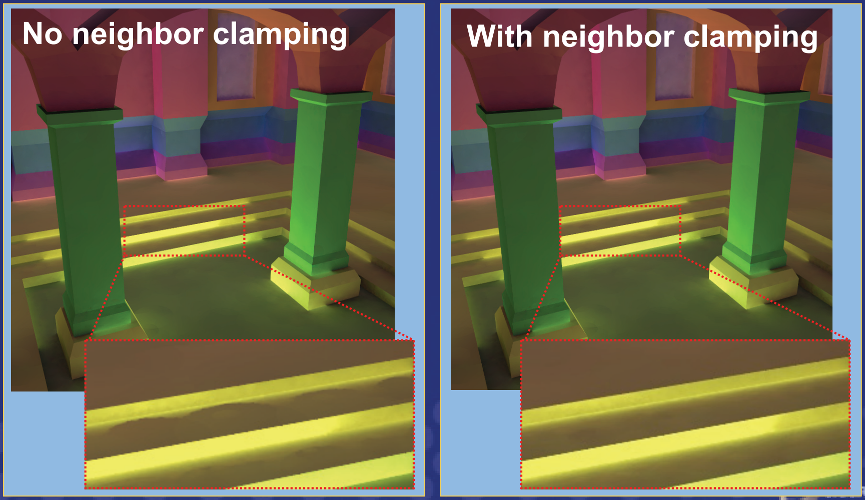
Here is how we proceed in practice. When a new record, *j*, is added to the cache, we first locate all existing records whose area of influence overlap with the area of influence of the record being added. (That is to say, all records *k*, such that $|| \mathbf{p}_j – \mathbf{p}_k || < R_j + R_k$). Then for all those records, we clamp the R_j value of the new record: $R_j = \min\{ R_j , R_k + || \mathbf{p}_j – \mathbf{p}_k || \}$. This enforces the triangle inequality. After that, we use the clamped R_j value of the new record to clamp the R_k values of the nearby records. This second step enforces the *transitivity* of triangle inequality.



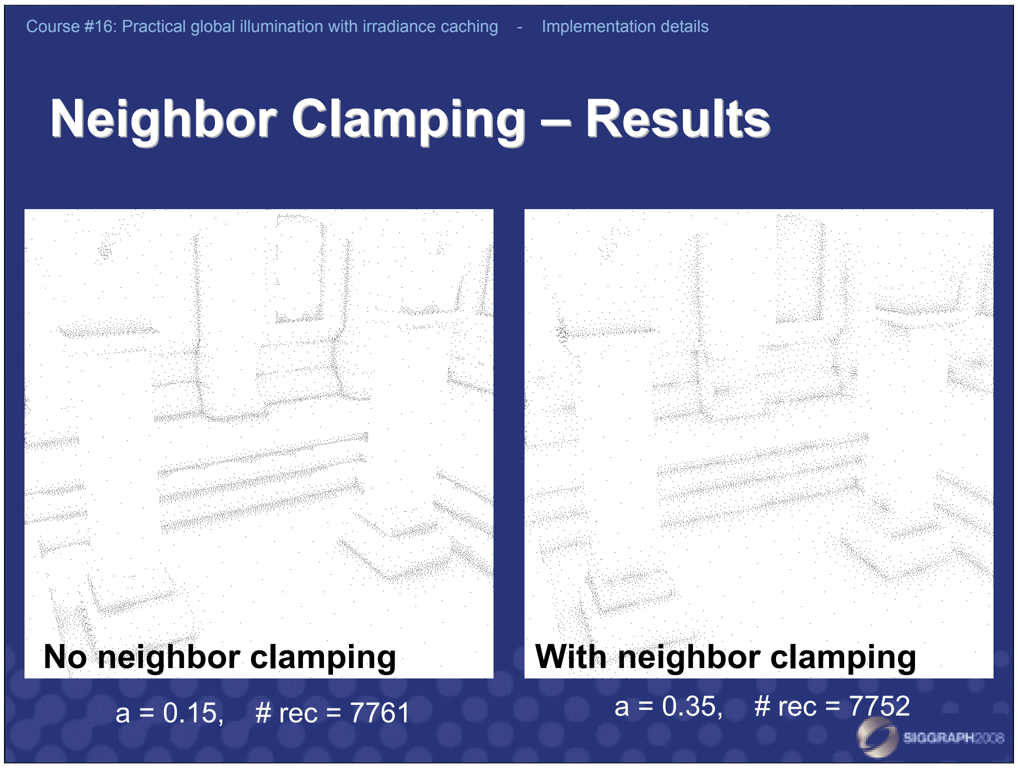
Course #16: Practical global illumination with irradiance caching - Implementation details

Neighbor Clamping – Results

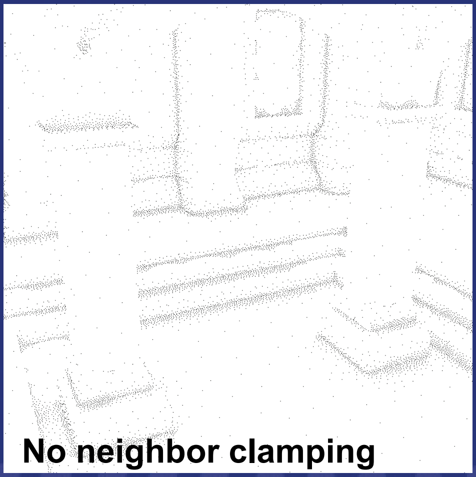
- Equalized distribution of records



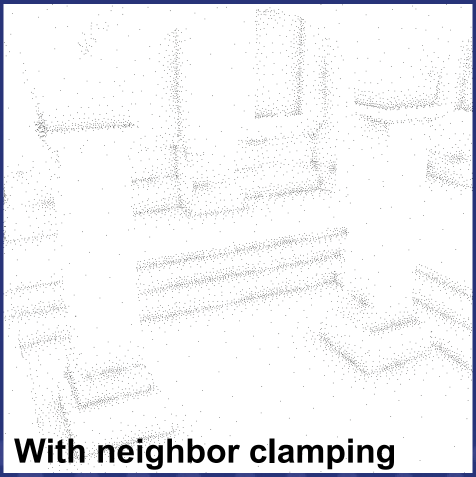
And voilà, the artifacts due to the over-estimated mean distance are gone.



Neighbor Clamping – Results

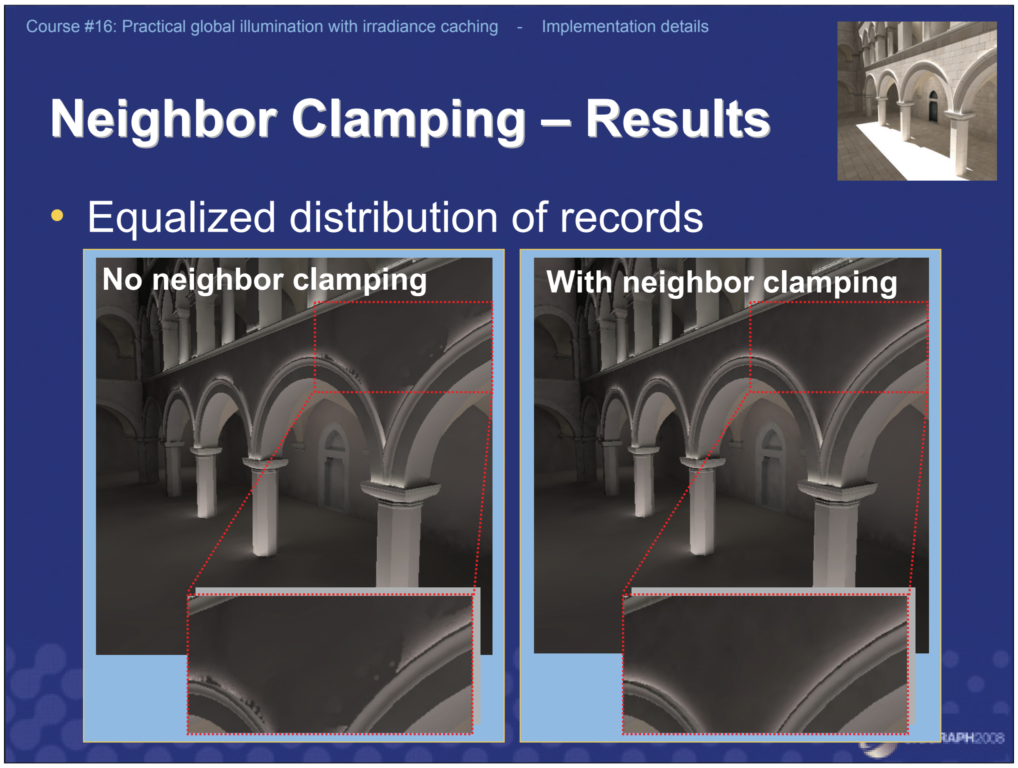


a = 0.15, # rec = 7761



a = 0.35, # rec = 7752

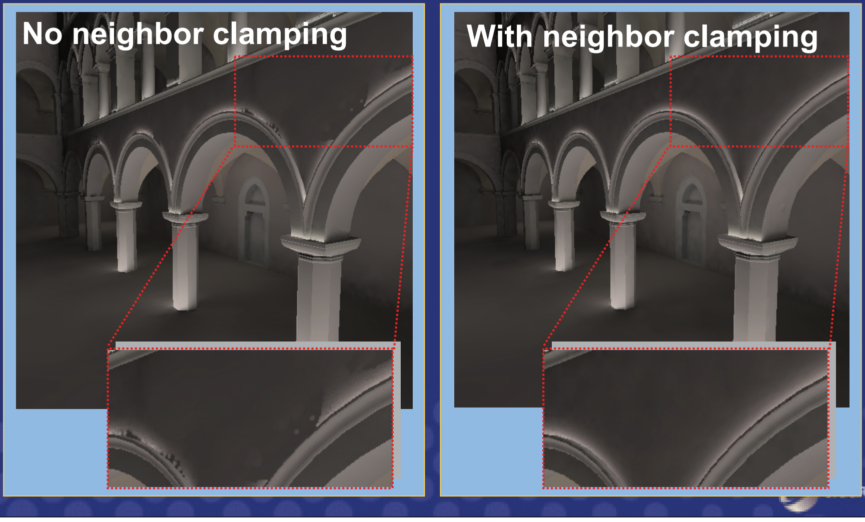
With neighbor clamping, the spacing between the records is equalized. It falls off gradually as we move away from the geometry features.



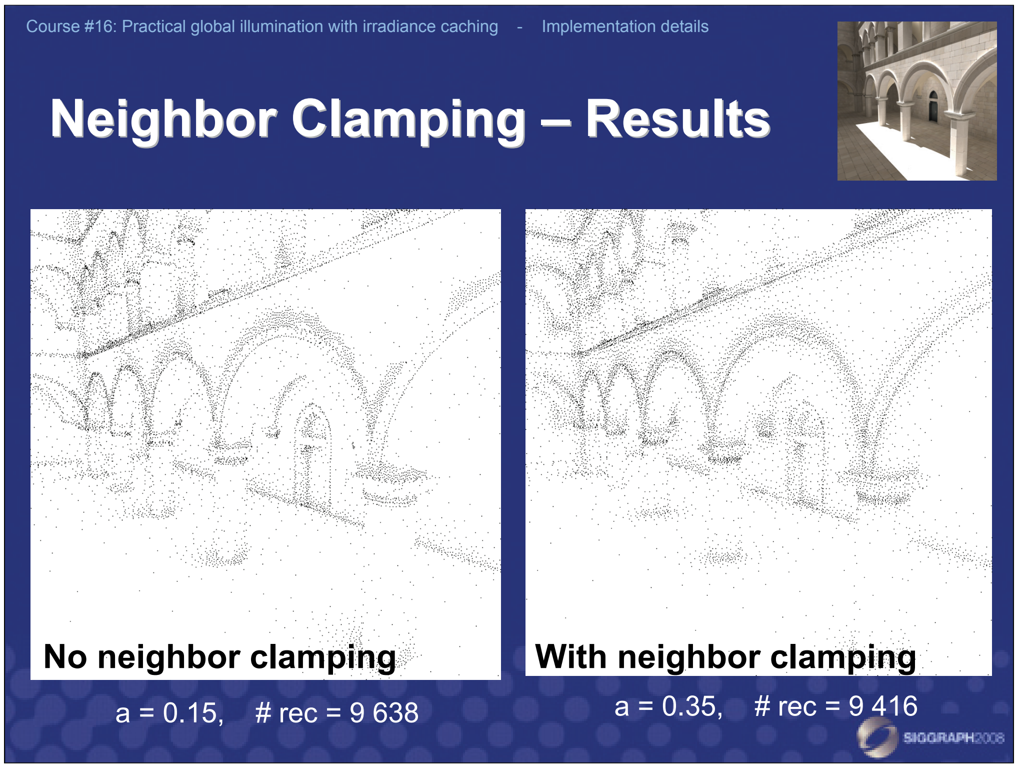
Course #16: Practical global illumination with irradiance caching - Implementation details

Neighbor Clamping – Results

- Equalized distribution of records

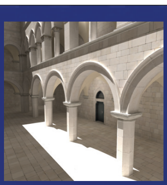


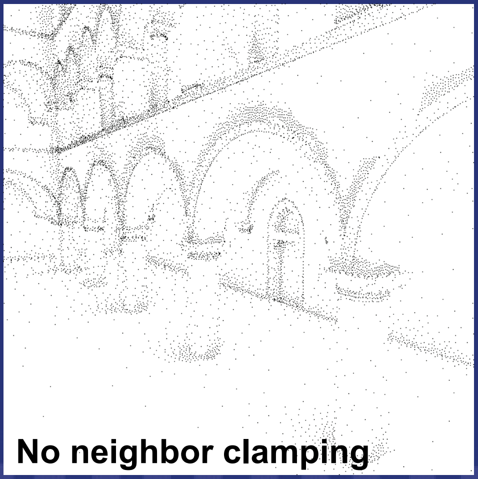
In the Sponza atrium scene, the benefit of neighbor clamping is even more apparent.



Course #16: Practical global illumination with irradiance caching - Implementation details

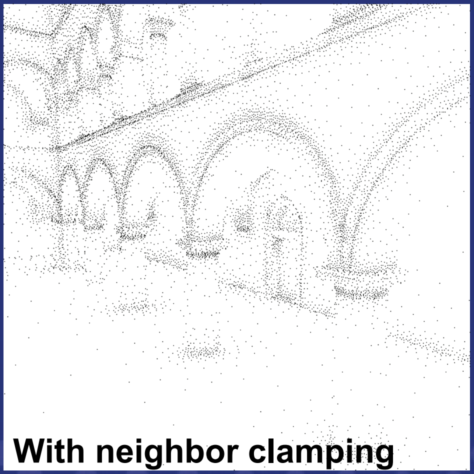
Neighbor Clamping – Results





No neighbor clamping

a = 0.15, # rec = 9 638



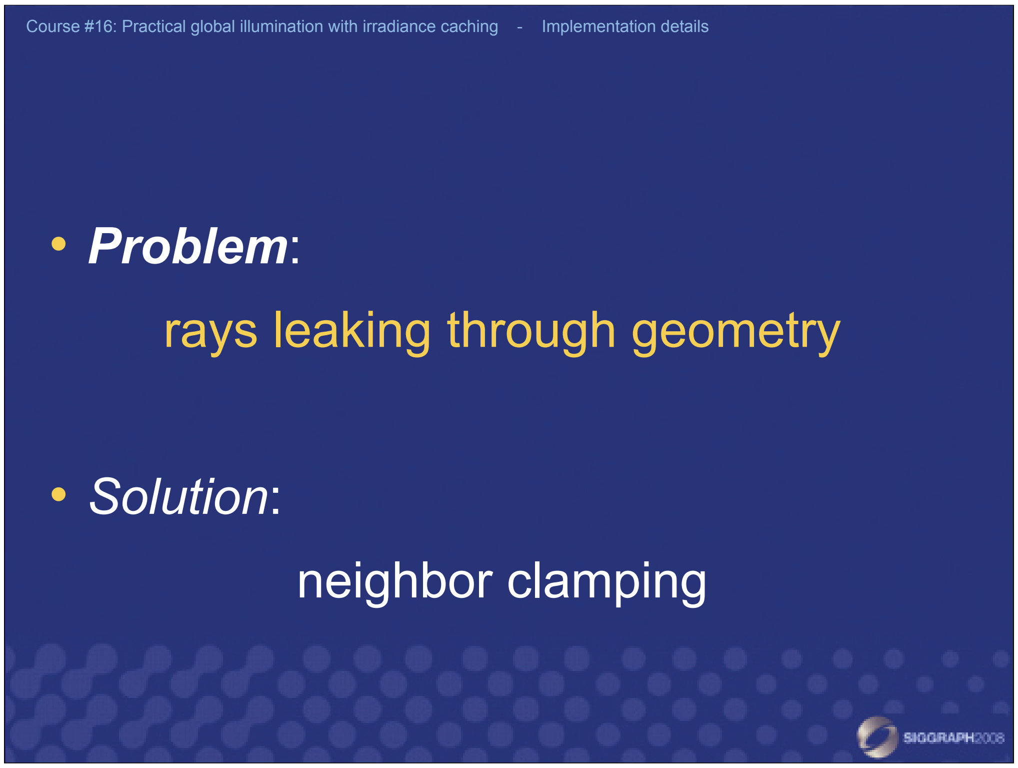
With neighbor clamping

a = 0.35, # rec = 9 416

SIGGRAPH2008

Record spacing is equalized and indirect illumination is properly sampled around the cornices above the arches.

To conclude, irradiance caching produces image artifacts when the mean distance to geometry is overestimated. Neighbor clamping reliably detects and corrects the overestimated mean distance, thereby suppressing these artifacts.



Course #16: Practical global illumination with irradiance caching - Implementation details

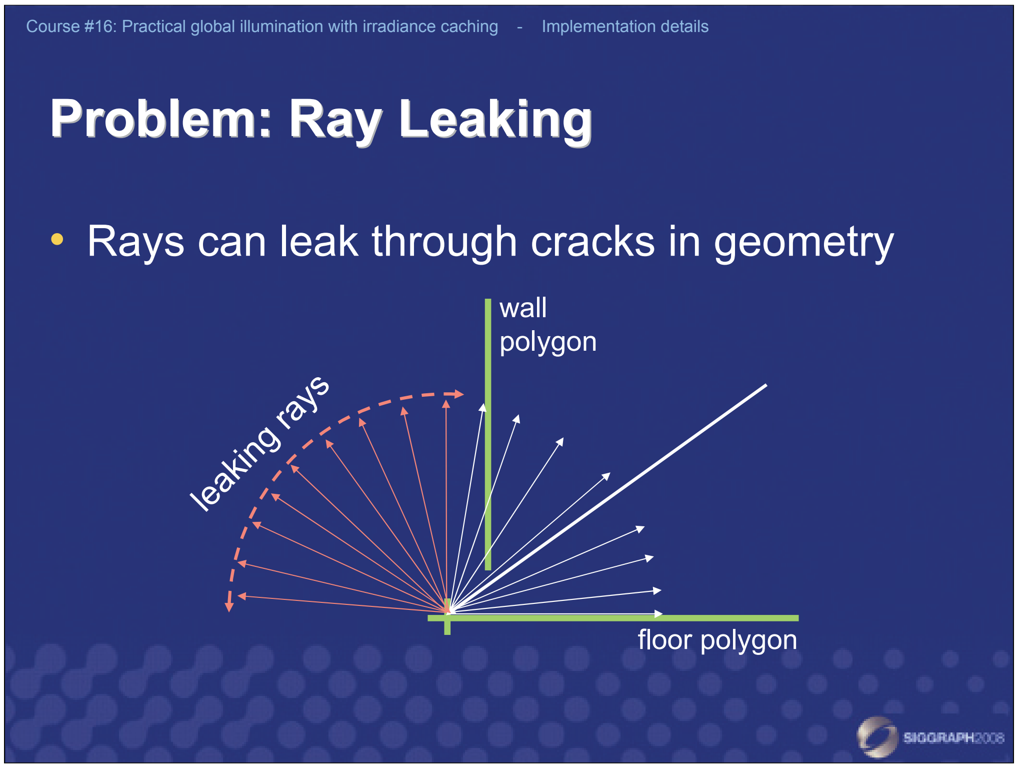
- ***Problem***:

  rays leaking through geometry

- *Solution*:

  neighbor clamping

SIGGRAPH2008



Course #16: Practical global illumination with irradiance caching - Implementation details

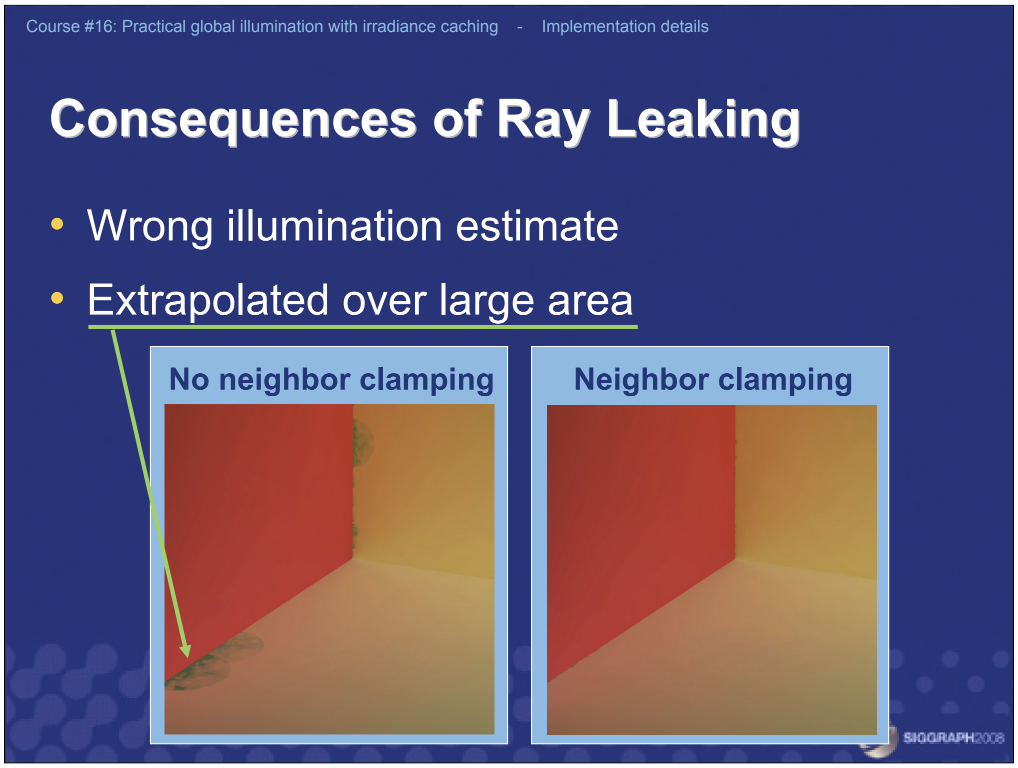
Problem: Ray Leaking

- Rays can leak through cracks in geometry

wall polygon
leaking rays
floor polygon
SIGGRAPH2008

Another serious problem of irradiance caching occurs when handling scenes with small cracks between polygons. Such scenes are quite common in practice – either because the scene is not well modeled or because of a limited numerical precision in the exported scene.

If a primary ray happens to hit such a crack, then most of the secondary rays used in hemisphere sampling “leak” through the crack. As a result, the irradiance estimate is completely wrong and, more seriously, the mean distance is greatly overestimated. (If there were no ray leaking, all those leaking rays would actually be very short.)



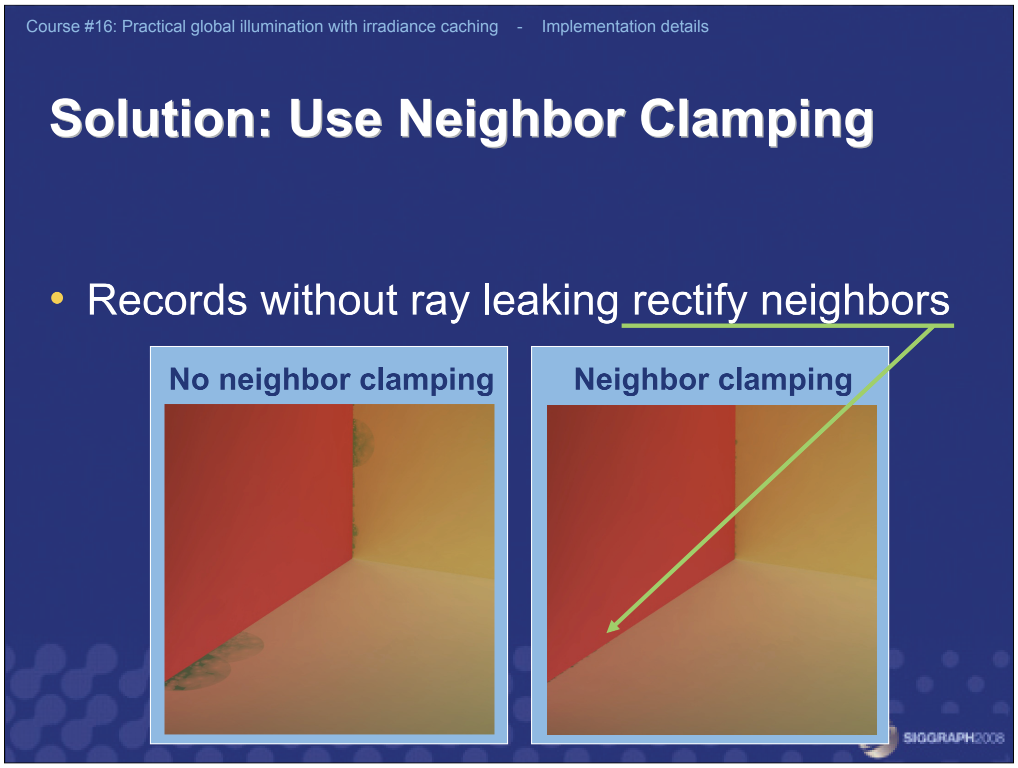
Course #16: Practical global illumination with irradiance caching - Implementation details

Consequences of Ray Leaking

- Wrong illumination estimate
- Extrapolated over large area

No neighbor clamping
Neighbor clamping
SIGGRAPH2008

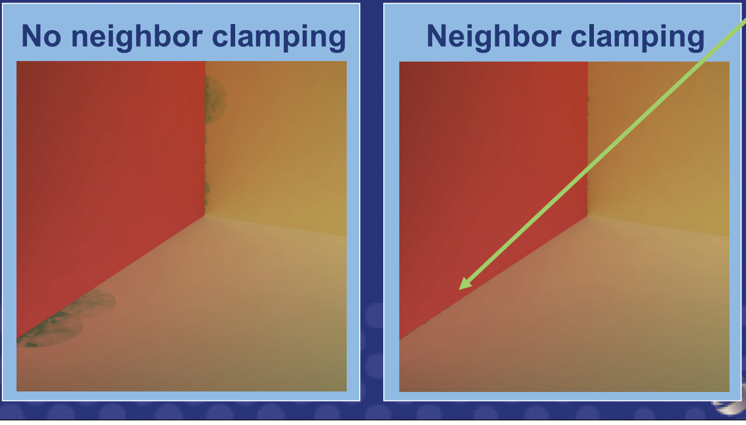
The incorrect irradiance estimate is then extrapolated over a large area.



Course #16: Practical global illumination with irradiance caching - Implementation details

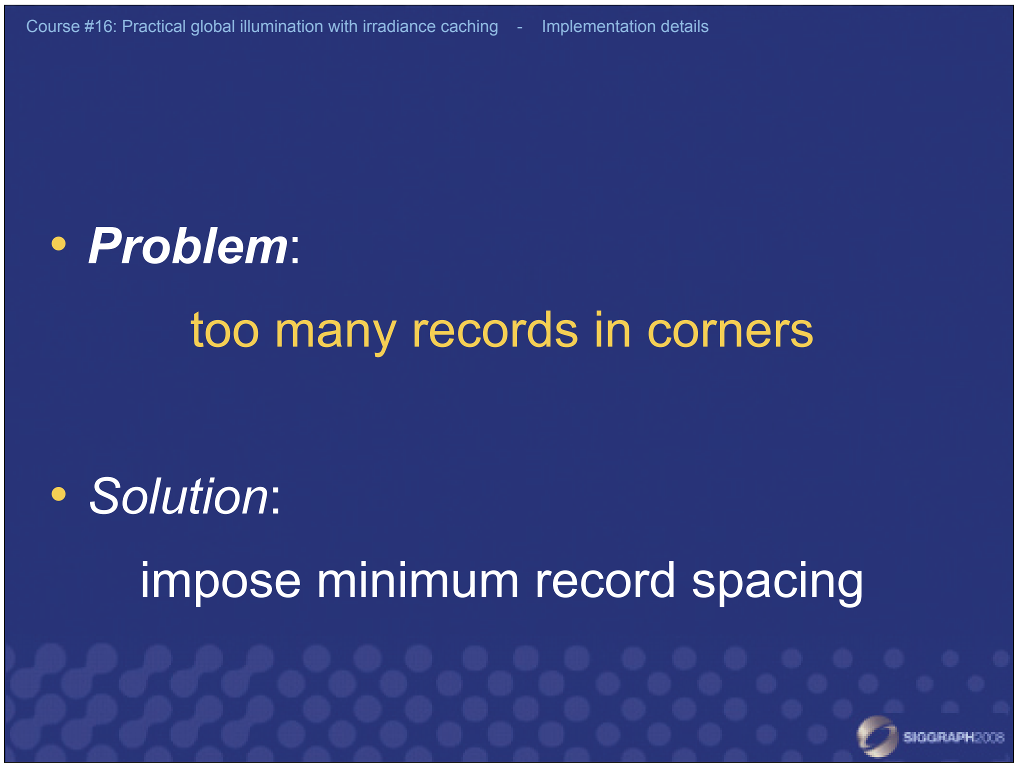
Solution: Use Neighbor Clamping

- Records without ray leaking rectify neighbors



SIGGRAPH2008

A partial remedy is quite simple – just turning on neighbor clamping. Neighbor clamping detects and rectifies overestimated mean distance, so the wrong irradiance estimate is not extrapolated over such a large area. However, neighbor clamping cannot do anything about the wrong irradiance estimate itself.



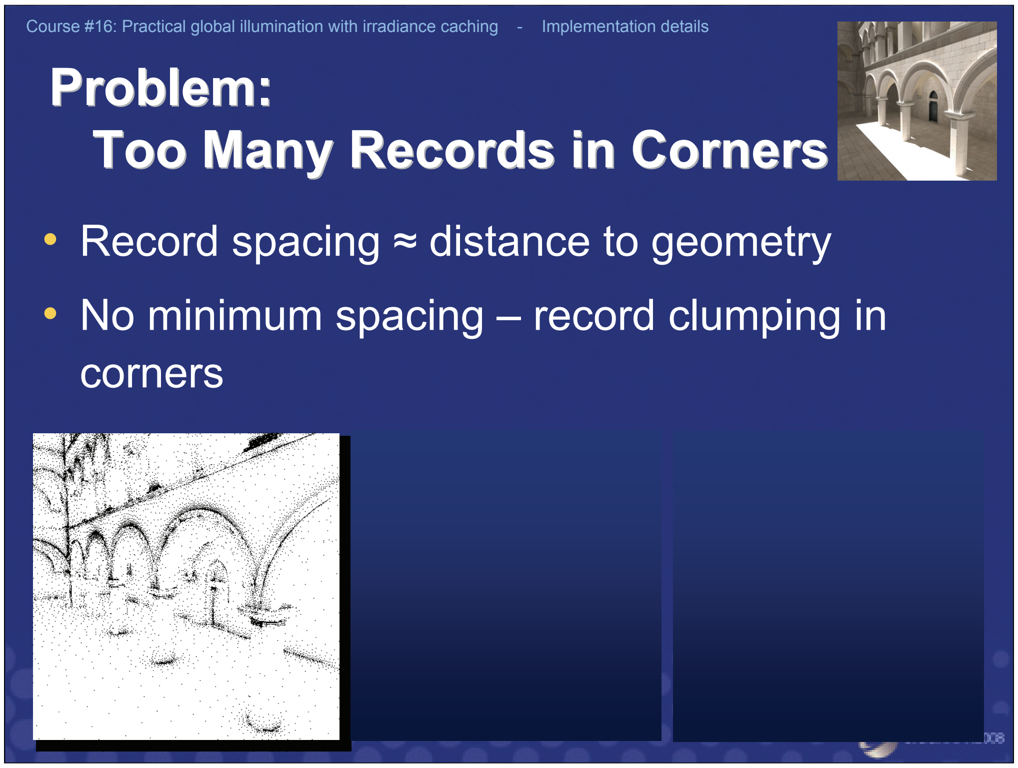
Course #16: Practical global illumination with irradiance caching - Implementation details

- ***Problem***:

  too many records in corners

- *Solution*:

  impose minimum record spacing

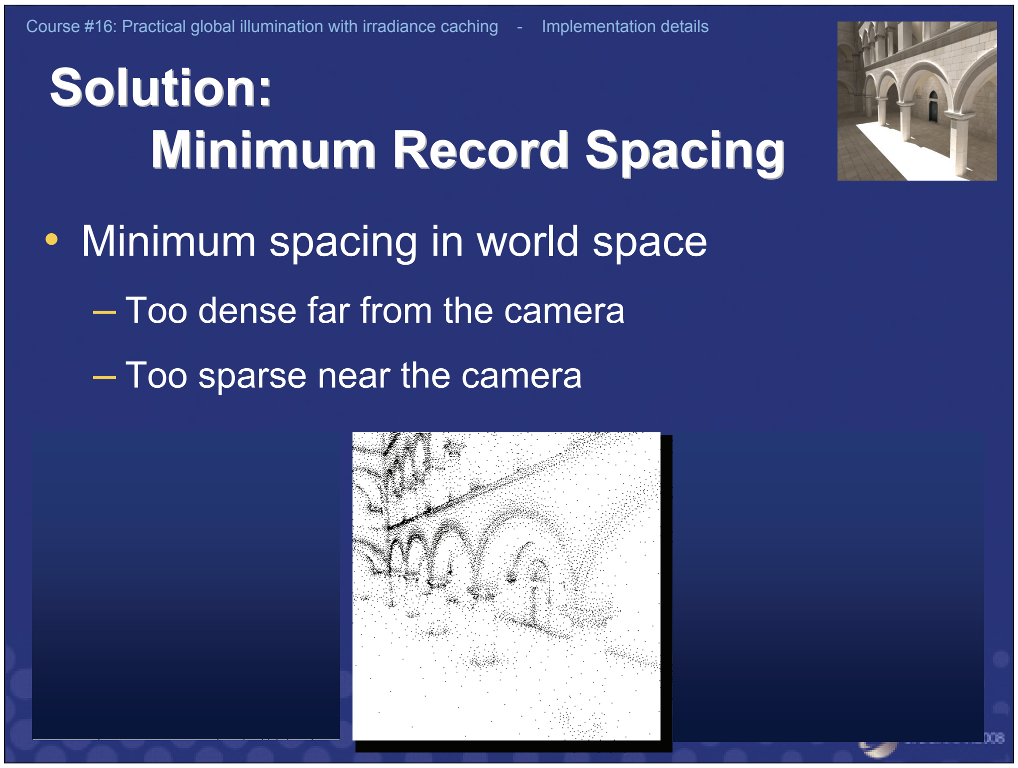


Course #16: Practical global illumination with irradiance caching - Implementation details

Problem: Too Many Records in Corners

- Record spacing ≈ distance to geometry
- No minimum spacing – record clumping in corners

Remember that the spacing of irradiance records is given by the mean distance to the neighboring geometry (and also by the object curvature, which we disregard in this discussion). If you do not impose any minimum limit on the spacing, irradiance caching will spend most of the time generating too many records around edges and corners. To avoid this problem, it is a good idea to impose a minimum distance between the records. This can be done by setting some minimum threshold R_{min} on the R_i value of a record.

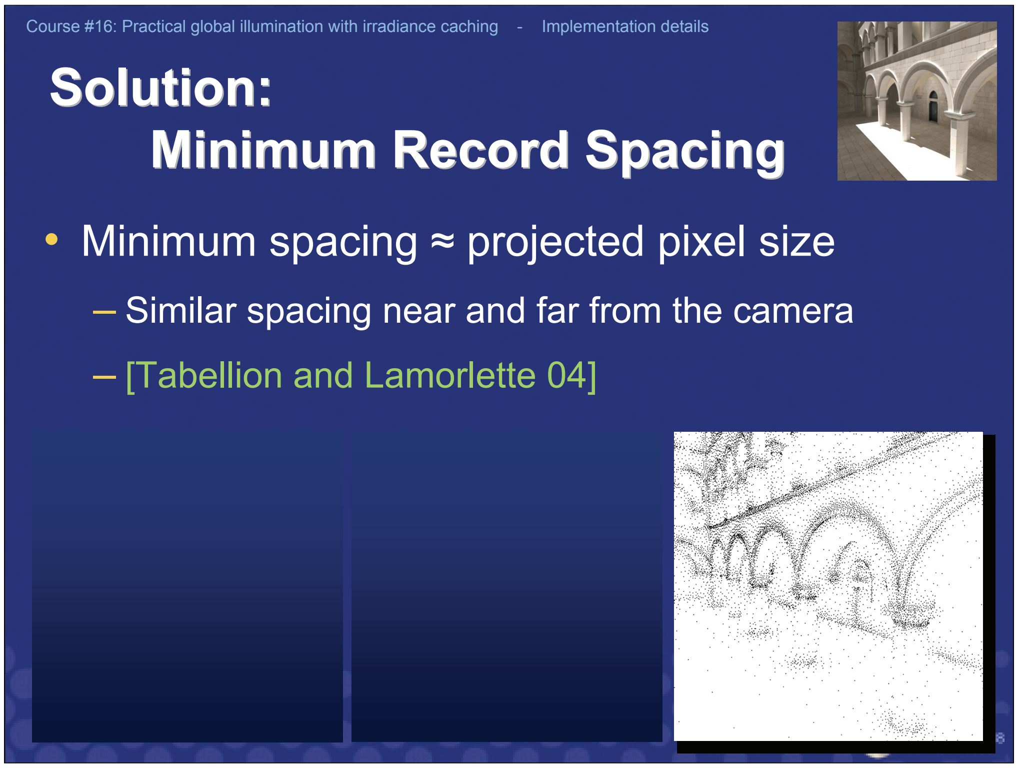


Course #16: Practical global illumination with irradiance caching - Implementation details

Solution: Minimum Record Spacing

- Minimum spacing in world space
  - Too dense far from the camera
  - Too sparse near the camera

One possibility is to limit the spacing in world space, by fixing the threshold, R_{min}, to the same value all over the scene. In *Radiance*, R_{min} is specified as a fraction of the scene size. This way of limiting the record spacing tends to generate too few radiance cache records near the camera and too many records far away.

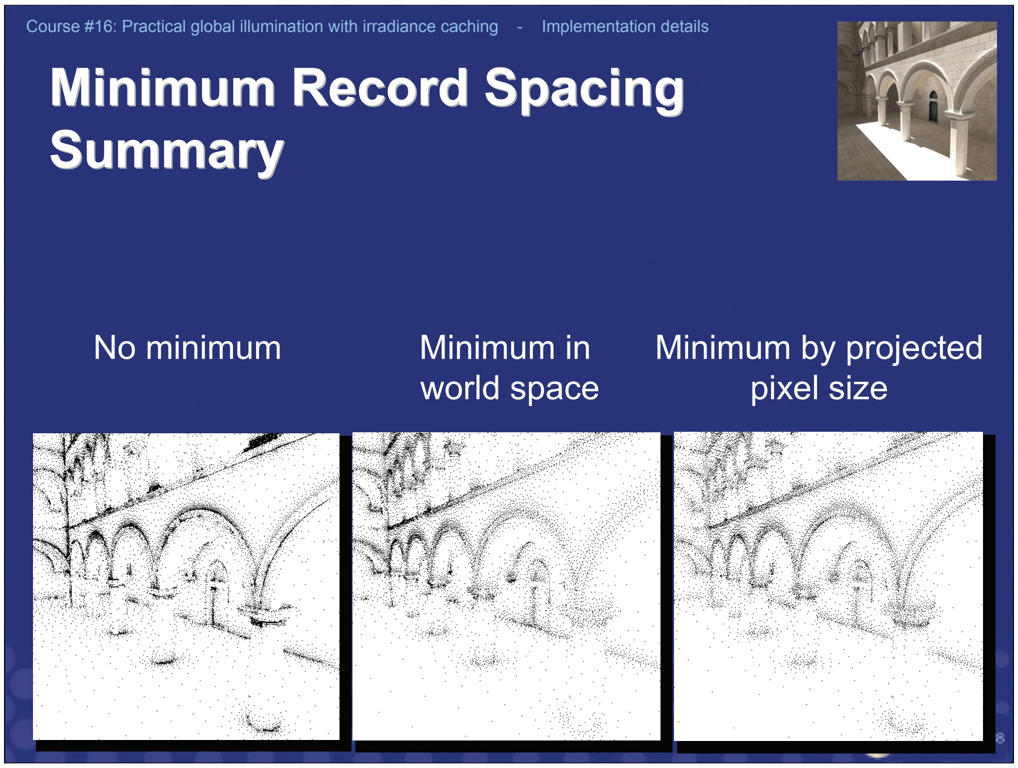


Course #16: Practical global illumination with irradiance caching - Implementation details

Solution: Minimum Record Spacing

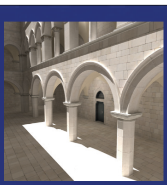
- Minimum spacing ≈ projected pixel size
  - Similar spacing near and far from the camera
  - [Tabellion and Lamorlette 04]

A better idea, proposed by Tabellion and Lamorlette [2004], is to use a multiple of the projected pixel size for the threshold R_{min}. Good values for R_{min} range between 1.5x and 3x the projected pixel size.

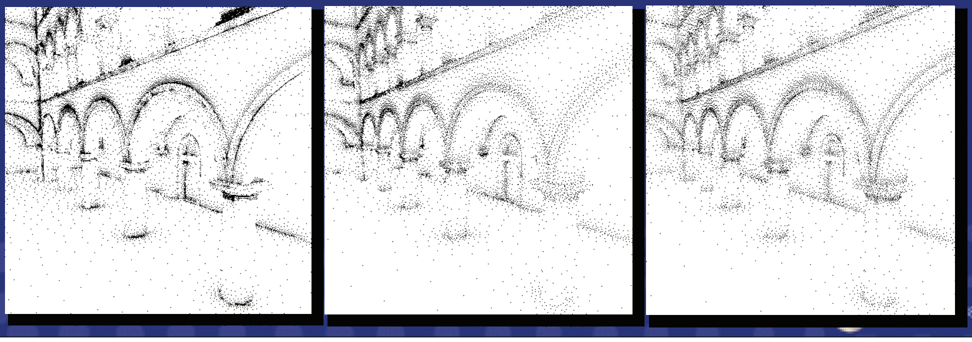


Course #16: Practical global illumination with irradiance caching - Implementation details

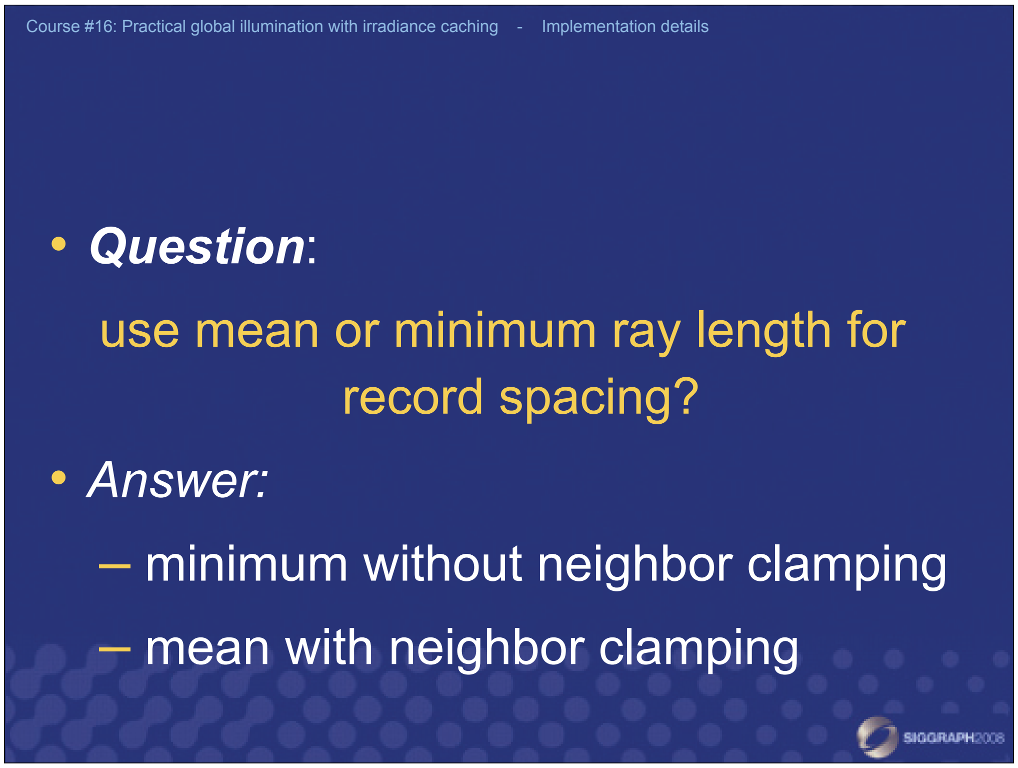
Minimum Record Spacing Summary



No minimum | Minimum in world space | Minimum by projected pixel size



Especially for exterior scenes, it is also important to limit the maximum value of R_i. In *Radiance*, this maximum is 64 times the minimum. Tabellion and Lamorlette [2004] use the maximum of 10x the projected pixel size.



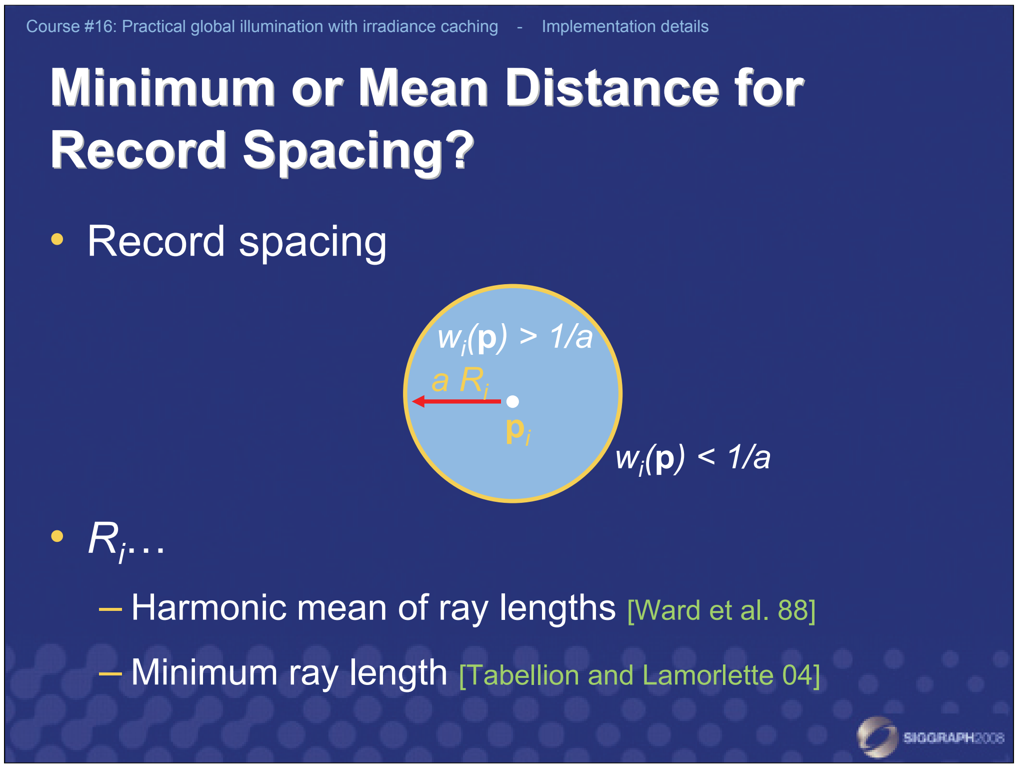
Course #16: Practical global illumination with irradiance caching - Implementation details

- ***Question***:

  use mean or minimum ray length for record spacing?

- *Answer:*
  - minimum without neighbor clamping
  - mean with neighbor clamping

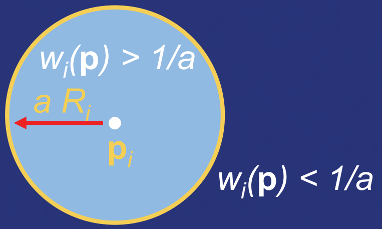
SIGGRAPH2008



Course #16: Practical global illumination with irradiance caching - Implementation details

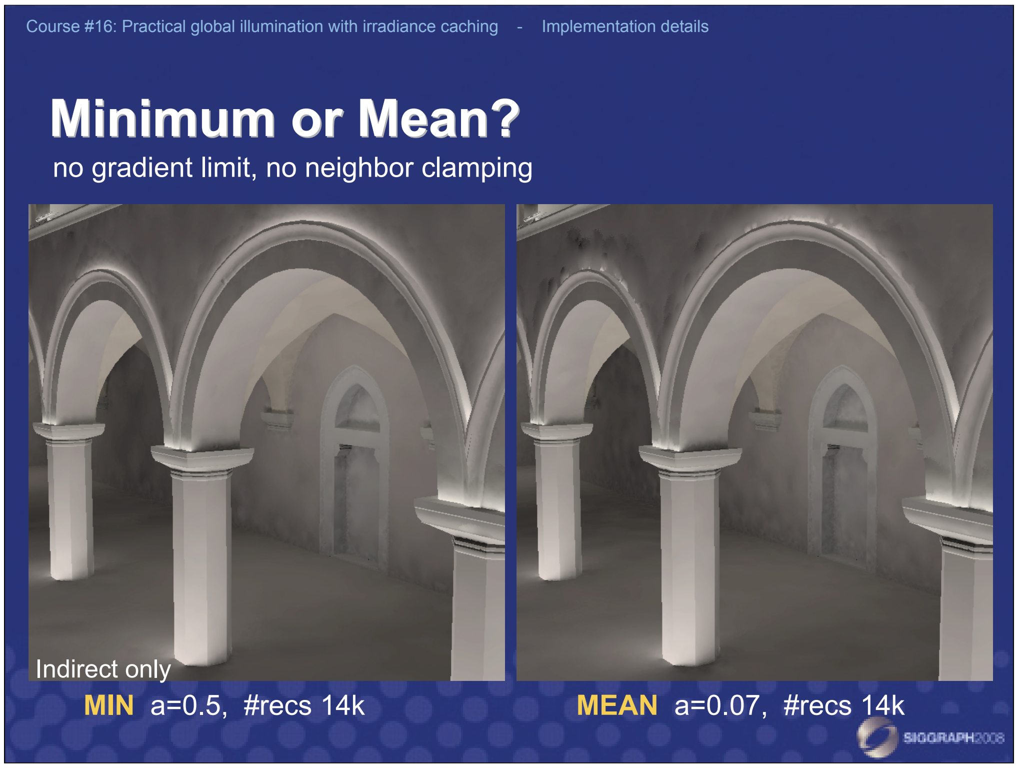
Minimum or Mean Distance for Record Spacing?

- Record spacing



- R_i…
  - Harmonic mean of ray lengths [Ward et al. 88]
  - Minimum ray length [Tabellion and Lamorlette 04]

In the irradiance caching implementation in *Radiance* [Ward et al. 1988], the record spacing is based upon the *mean* distance to neighboring geometry. This approach tends to miss some small geometry features. At EGSR in 2006 [Křivánek et al. 2006], we proposed neighbor clamping to resolve these problems. Tabellion and Lamorlette [2004] use the minimum distance instead of the mean distance to resolve these problems. Let us see how irradiance caching behaves when using the minimum and mean distance.



Course #16: Practical global illumination with irradiance caching - Implementation details

Minimum or Mean?

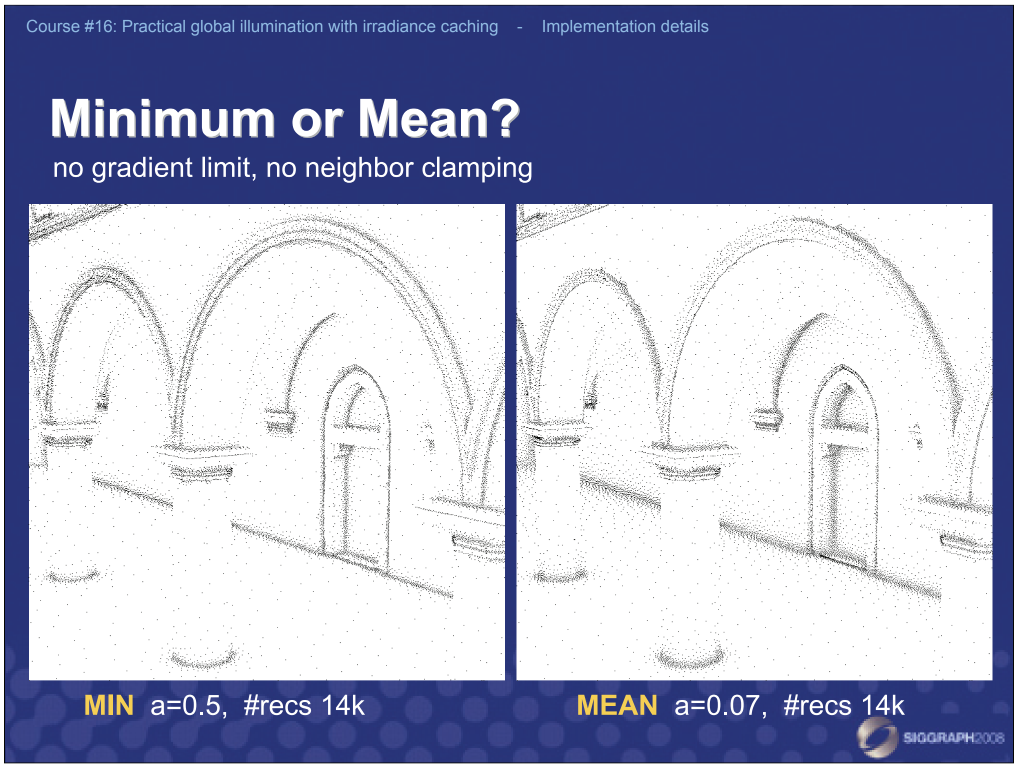
no gradient limit, no neighbor clamping

MIN a=0.5, #recs 14k

MEAN a=0.07, #recs 14k

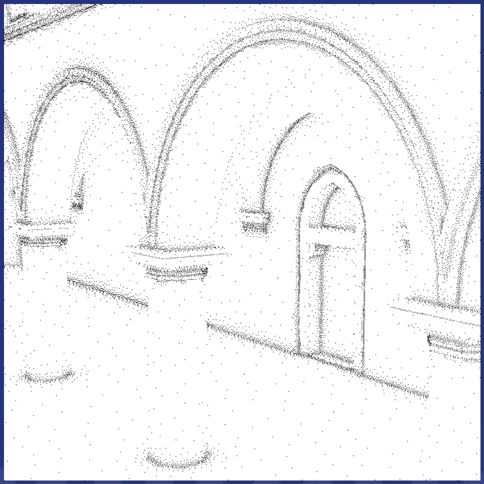
First, when there is *no* gradient limit on record spacing and neighbor clamping is *not* used, then the minimum distance indeed produces much better images than the mean distance.

(See Greg Ward's slides on Radiance implementation for more information about the gradient limit on record spacing.)

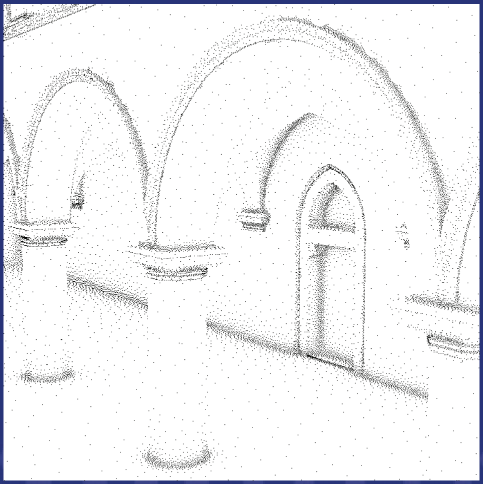


Minimum or Mean?

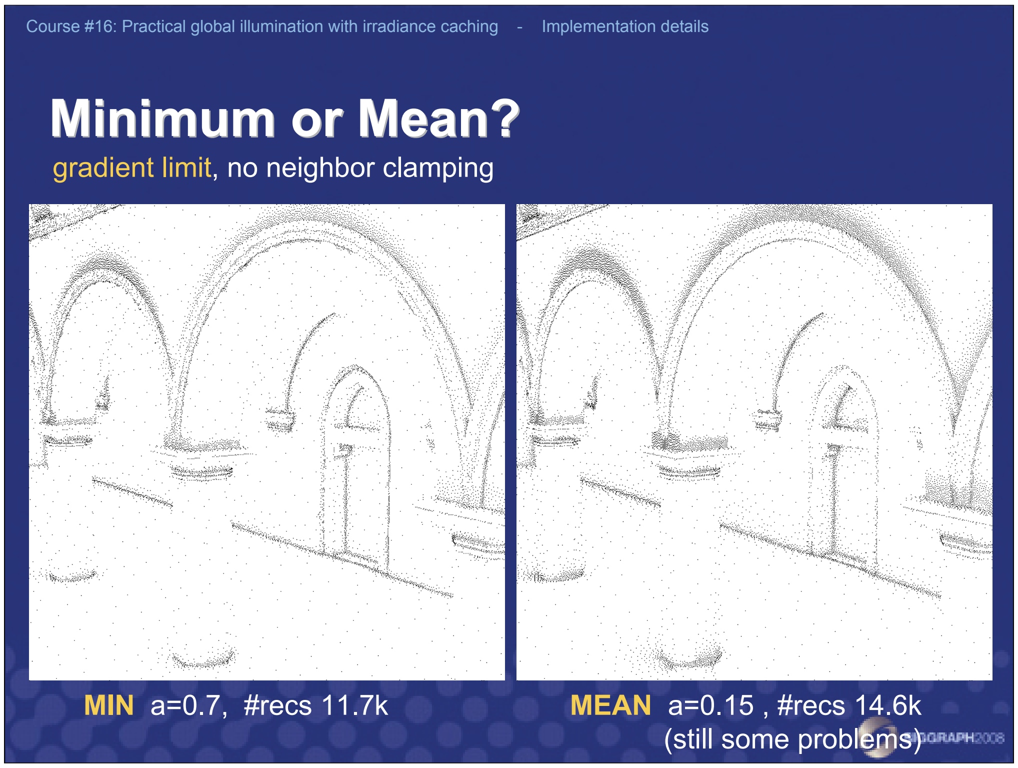
no gradient limit, no neighbor clamping



MIN a=0.5, #recs 14k



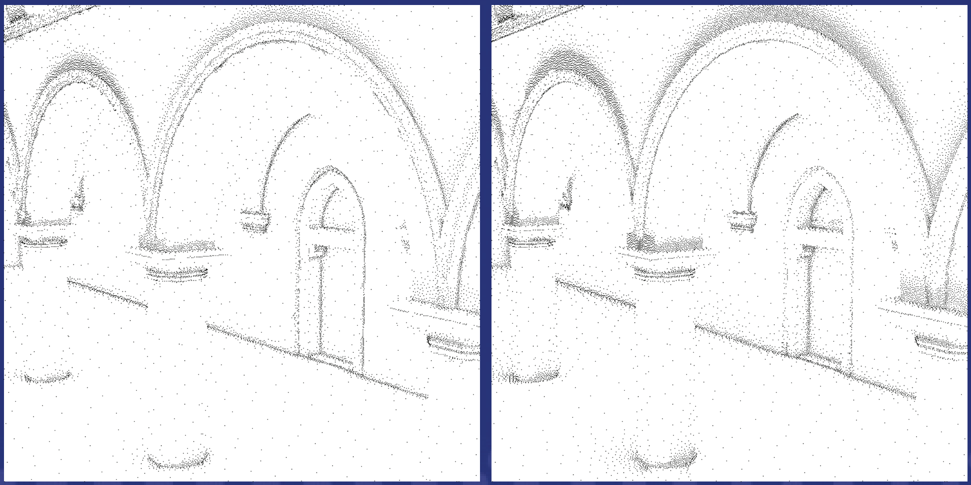
MEAN a=0.07, #recs 14k



Course #16: Practical global illumination with irradiance caching - Implementation details

Minimum or Mean?

gradient limit, no neighbor clamping

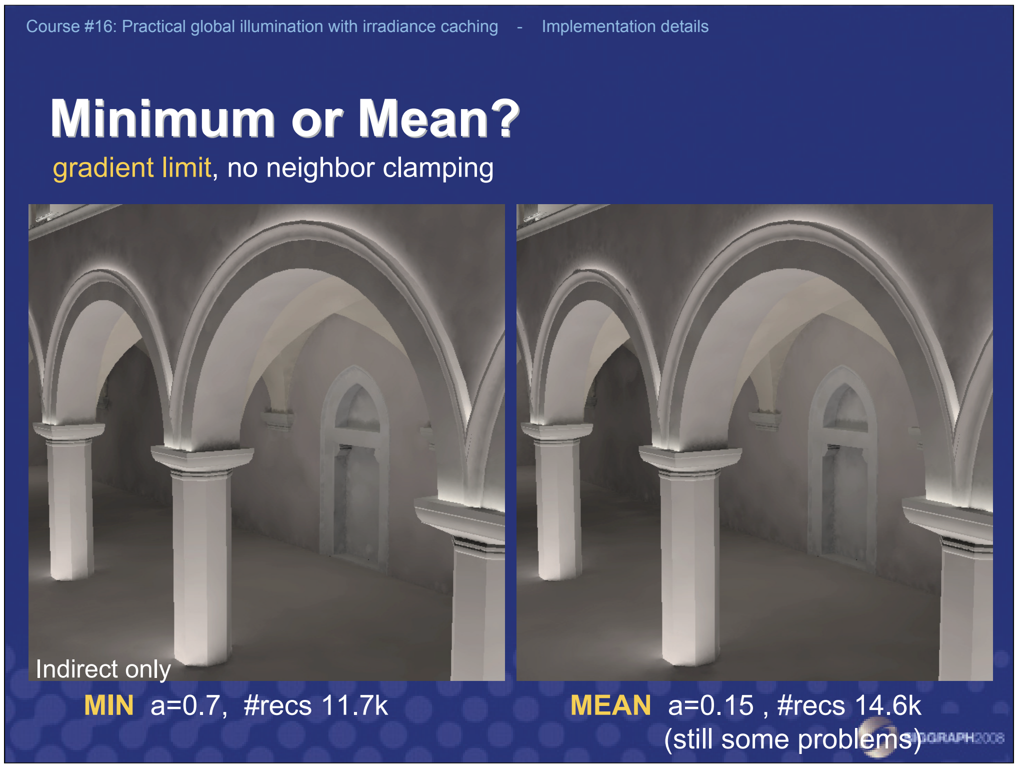


MIN a=0.7, #recs 11.7k

MEAN a=0.15 , #recs 14.6k
(still some problems)

If we limit record spacing by the translational gradient, we get much more records in the high-gradient areas around the cornices.

This is also a nice example of the gradient limit on record spacing.



Course #16: Practical global illumination with irradiance caching - Implementation details

Minimum or Mean?

gradient limit, no neighbor clamping

Indirect only

MIN a=0.7, #recs 11.7k

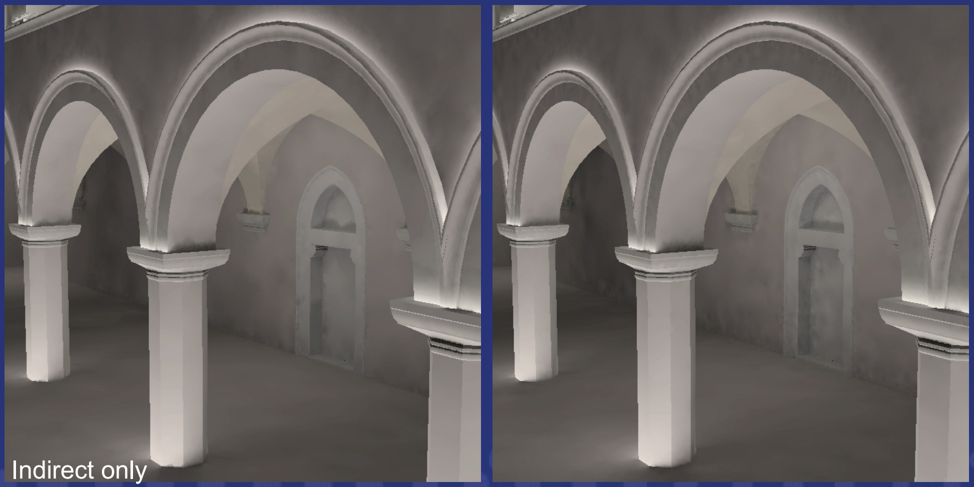
MEAN a=0.15 , #recs 14.6k (still some problems)

SIGGRAPH2008

But still, even with the gradient limited spacing, using the mean distance leaves some artifacts around the cornices.

Minimum or Mean?

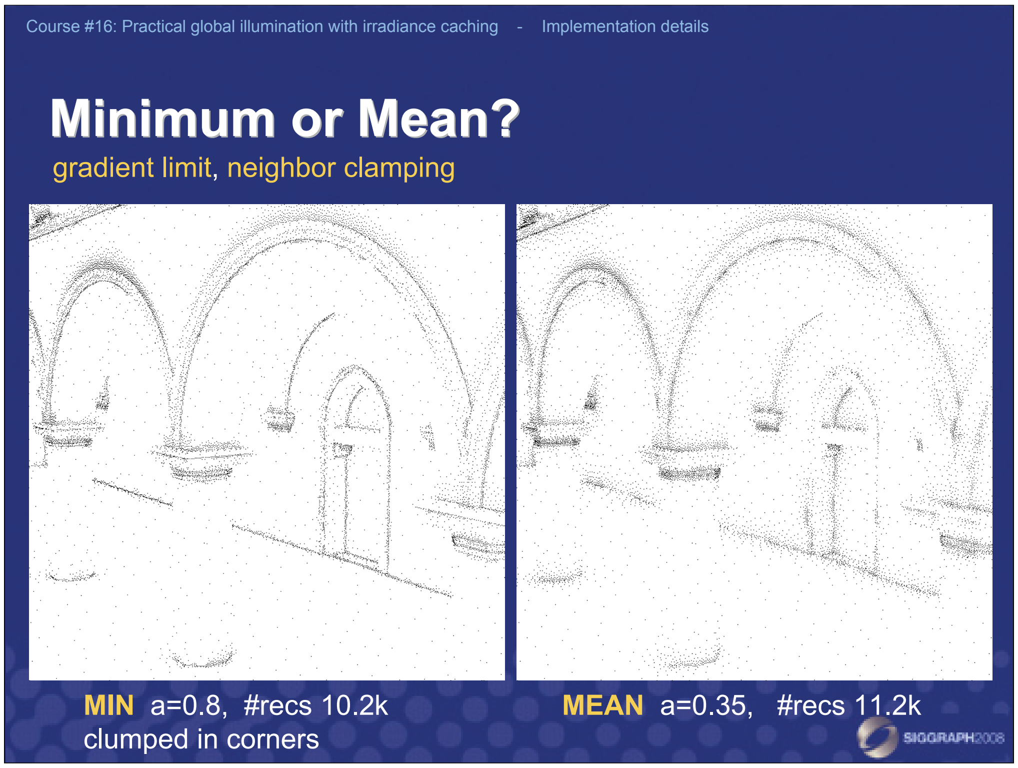
gradient limit, neighbor clamping



MIN a=0.8, #recs 10.2k

MEAN a=0.35, #recs 11.2k

However, when we turn on neighbor clamping, the artifacts are gone.



Course #16: Practical global illumination with irradiance caching - Implementation details

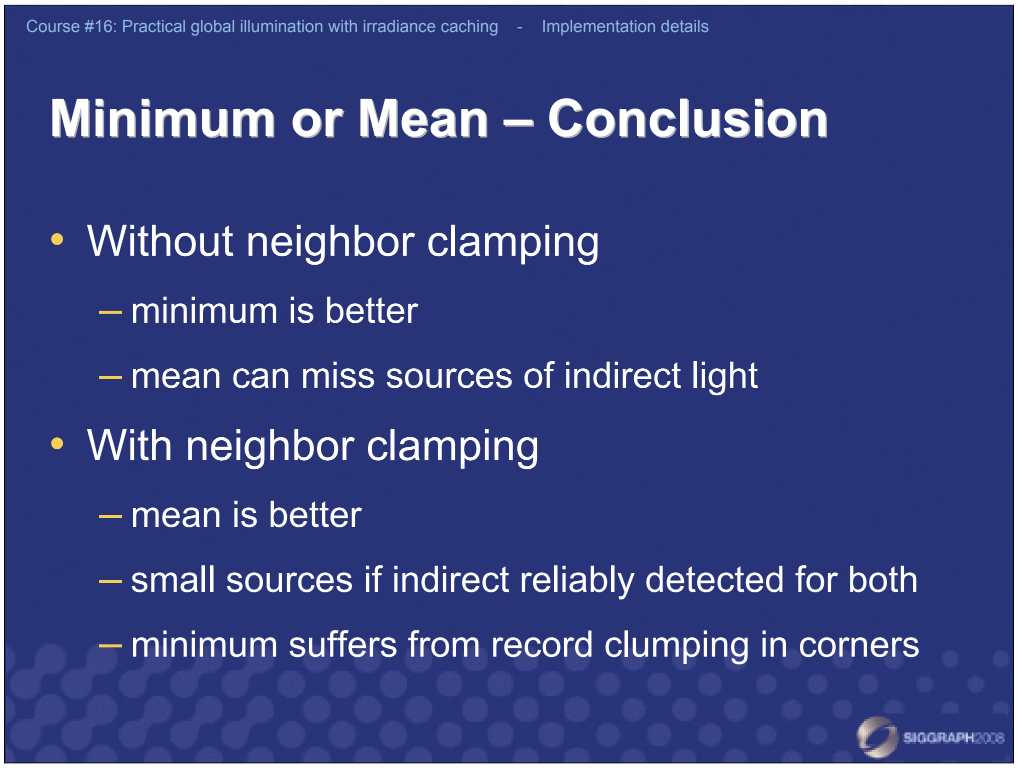
Minimum or Mean?

gradient limit, neighbor clamping

MIN a=0.8, #recs 10.2k
clumped in corners

MEAN a=0.35, #recs 11.2k

Looking at the record distribution, we see that with the mean distance, record spacing falls of gradually as we move away from the geometry, whereas with the minimum distance, records tend to concentrate in the corners.



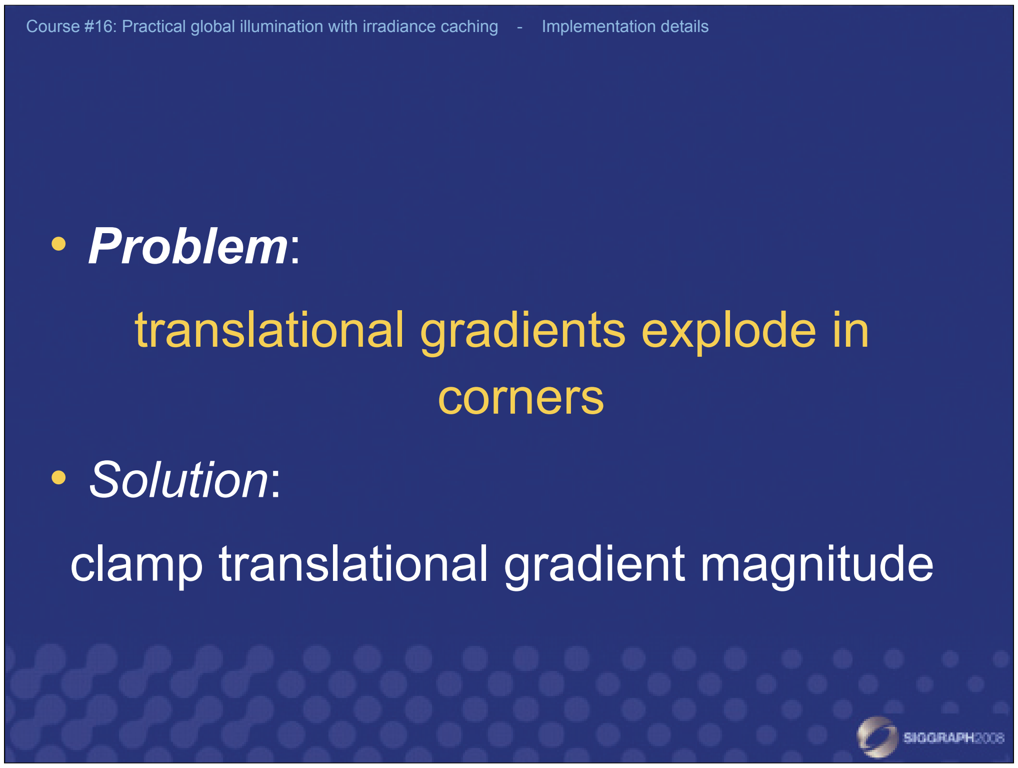
Course #16: Practical global illumination with irradiance caching - Implementation details

Minimum or Mean – Conclusion

- Without neighbor clamping
  - minimum is better
  - mean can miss sources of indirect light
- With neighbor clamping
  - mean is better
  - small sources if indirect reliably detected for both
  - minimum suffers from record clumping in corners

SIGGRAPH2008

In our experience, the gradual falloff of the spacing is desirable. In our practice, when gradient limit on record spacing and neighbor clamping is used, then the mean distance usually produces better images with the same number of records than the minimum distance.



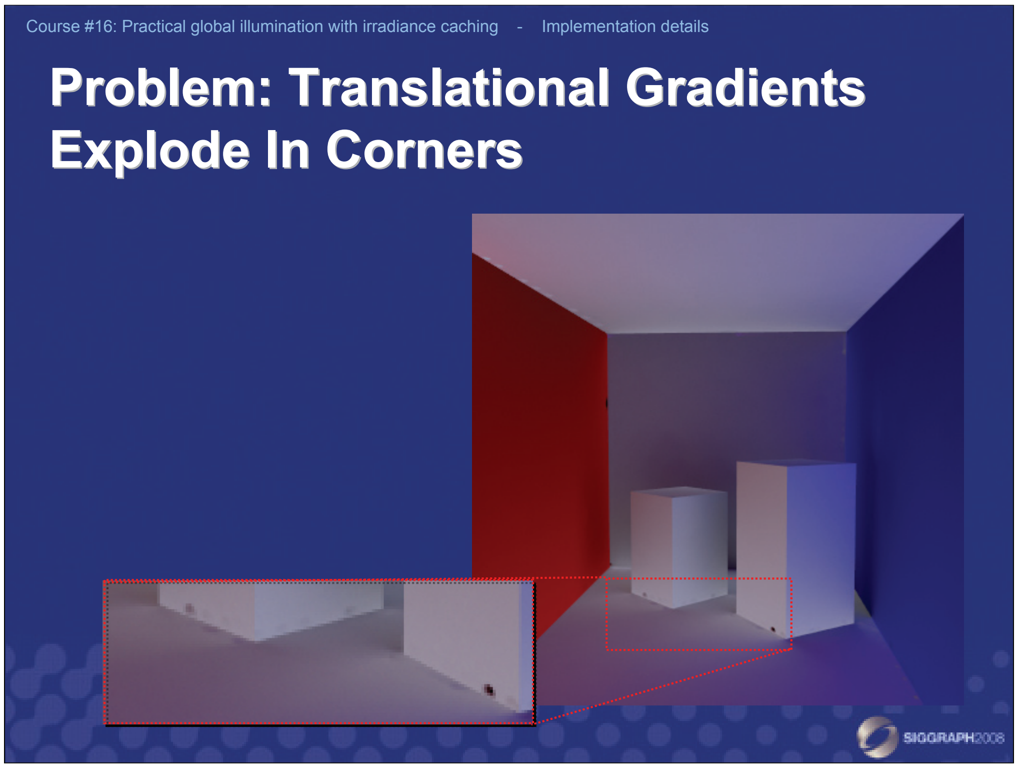
Course #16: Practical global illumination with irradiance caching - Implementation details

- ***Problem***:

translational gradients explode in corners

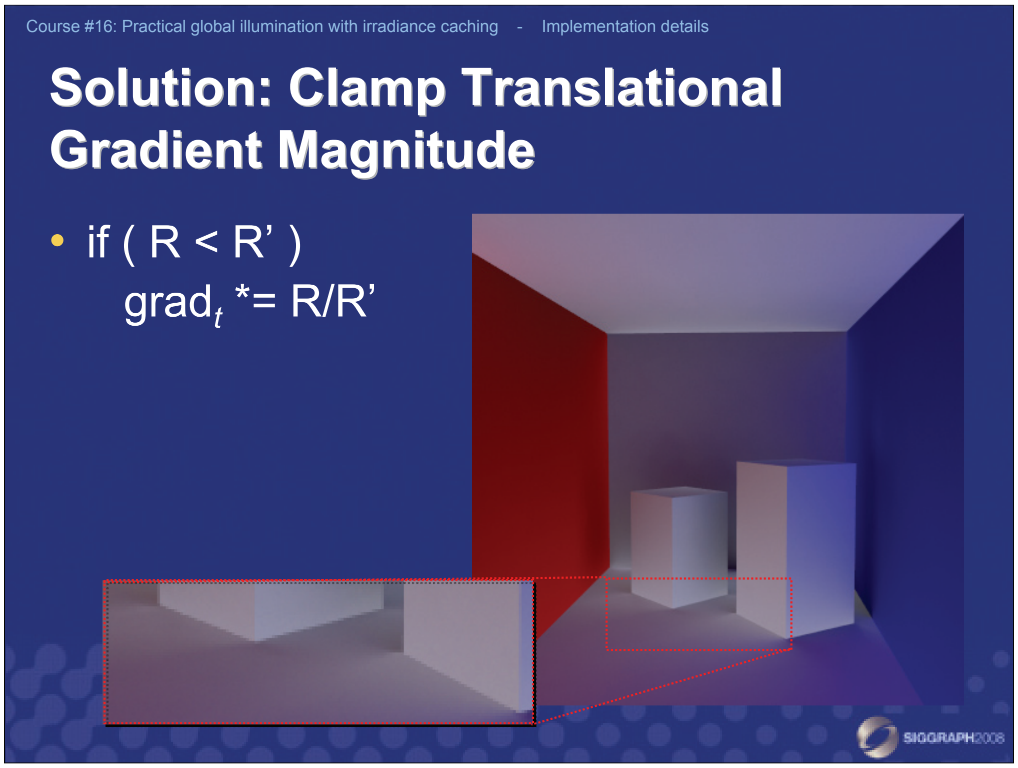
- *Solution*:

clamp translational gradient magnitude


Course #16: Practical global illumination with irradiance caching - Implementation details

Problem: Translational Gradients Explode In Corners

A common problem in irradiance caching implementation are the dark and bright patches in corner areas due to the exaggerated gradient magnitude. This happens because of the division by the ray length in gradient computation, which happens to be is very tiny in the corners.



Course #16: Practical global illumination with irradiance caching - Implementation details

Solution: Clamp Translational Gradient Magnitude

- if (R < R')
  grad_t *= R/R'

Similarly to other IC implementations, I limit the gradient magnitude in the corner areas, although the heuristic I use slightly differs from the Greg Ward's *Radiance* implementation or Eric Tabellions's PDI/Dreamworks implementation. Mine is quite straightforward:

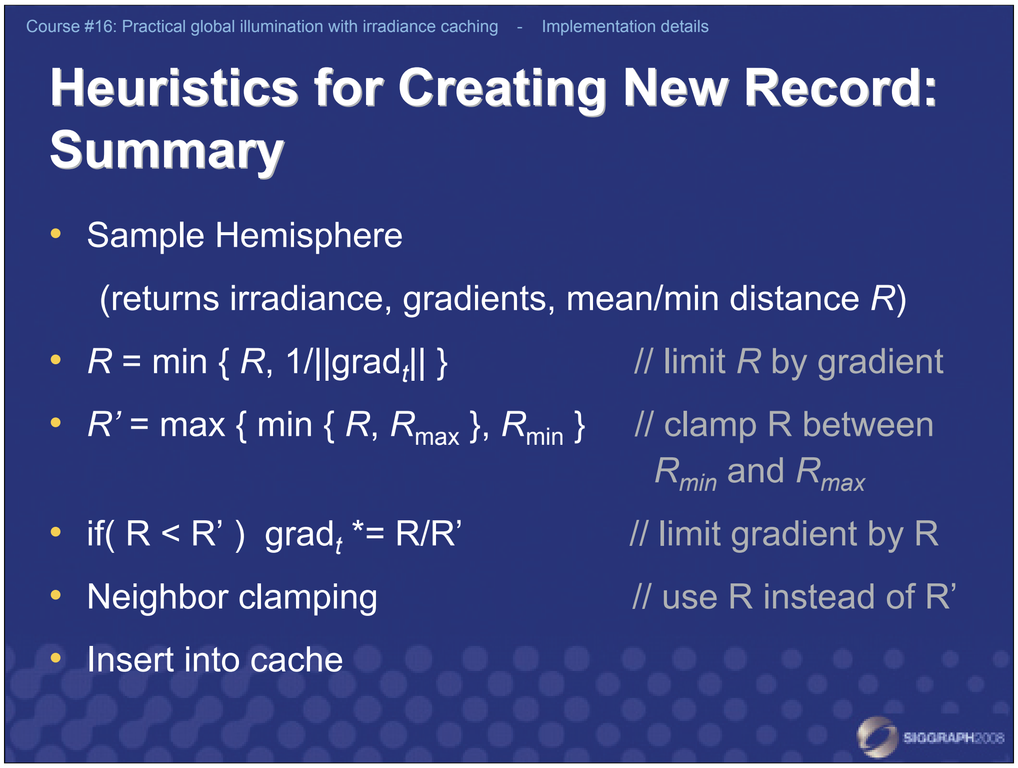
```
if ( R < R' )
    grad_t *= R/R'
```

Here:

- *R* is the minimum/mean ray length computed by the hemisphere sampling, possibly reduced by the gradient magnitude.

- *R*' is the value of *R* after clamping between the minimum and maximum spacing thresholds, $R' = \max \{ \min \{ R, R_{max} \}, R_{min} \}$

- grad_t is the translational gradient (one 3D vector per R,G,B)


Course #16: Practical global illumination with irradiance caching - Implementation details

Heuristics for Creating New Record: Summary

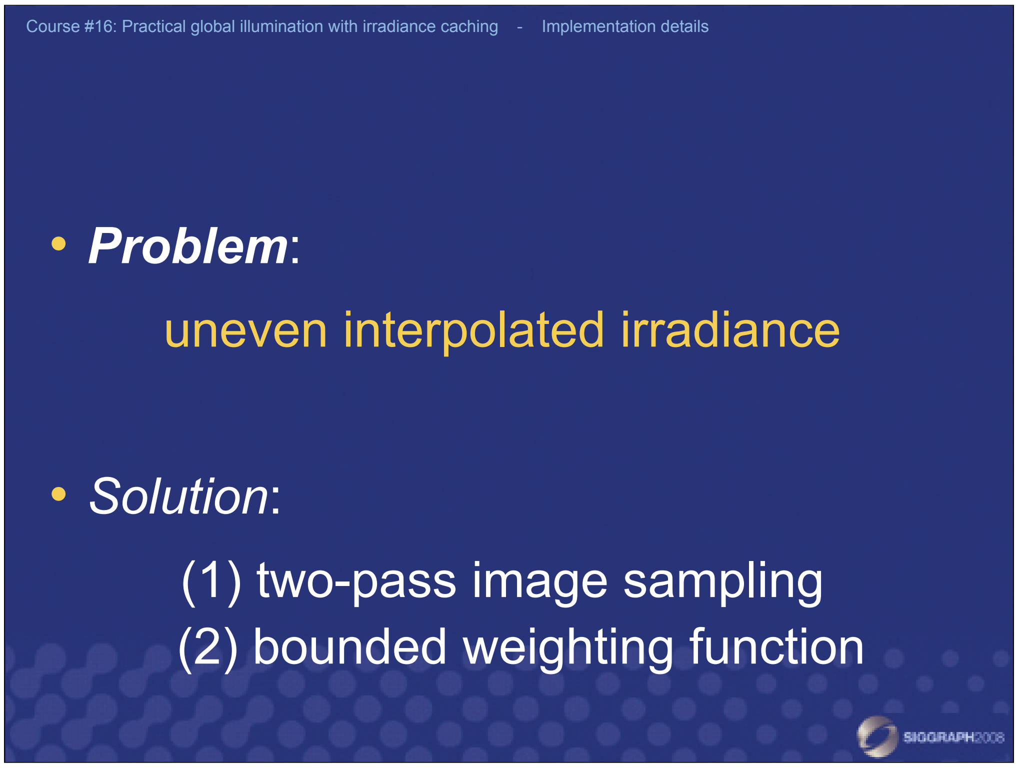
- Sample Hemisphere

  (returns irradiance, gradients, mean/min distance R)
- $R = \min \{ R, 1/||\mathrm{grad}_t|| \}$ // limit R by gradient
- $R' = \max \{ \min \{ R, R_{\max} \}, R_{\min} \}$ // clamp R between R_{min} and R_{max}
- if(R < R') grad_t *= R/R' // limit gradient by R
- Neighbor clamping // use R instead of R'
- Insert into cache

This slide summarizes the actions taken to add a new irradiance record into the cache. First, we sample the hemisphere by casting a number of secondary rays. This gives the irradiance estimate, the translational and the rotational gradients, and the estimate of the mean (or minimum) distance to the neighboring geometry, R. We then limit the value of R by the translational gradient as follows: $R = \min \{ R, 1/||\mathrm{grad}_t|| \}$. After that we clamp R by the minimum and maximum threshold (determined from the projected pixel size). This produces the clamped value $R' = \max \{ \min \{ R, R_{\max} \}, R_{\min} \}$. If R was increased by this clamping (*i.e.* if R was originally less than $R_{\min}$), we decrease the gradient magnitude accordingly, in order to avoid negative values in extrapolation.

The next step is neighbor clamping. For its correct functionality, it is essential to use the original, unclamped value of R (not R') in the neighbor clamping procedure and redo the clamping every time R is reduced. It also means that each record must store the value of both R and R' and also $R_{\max}$, $R_{\min}$ used for the clamping.

Finally, we insert the new record into the cache.



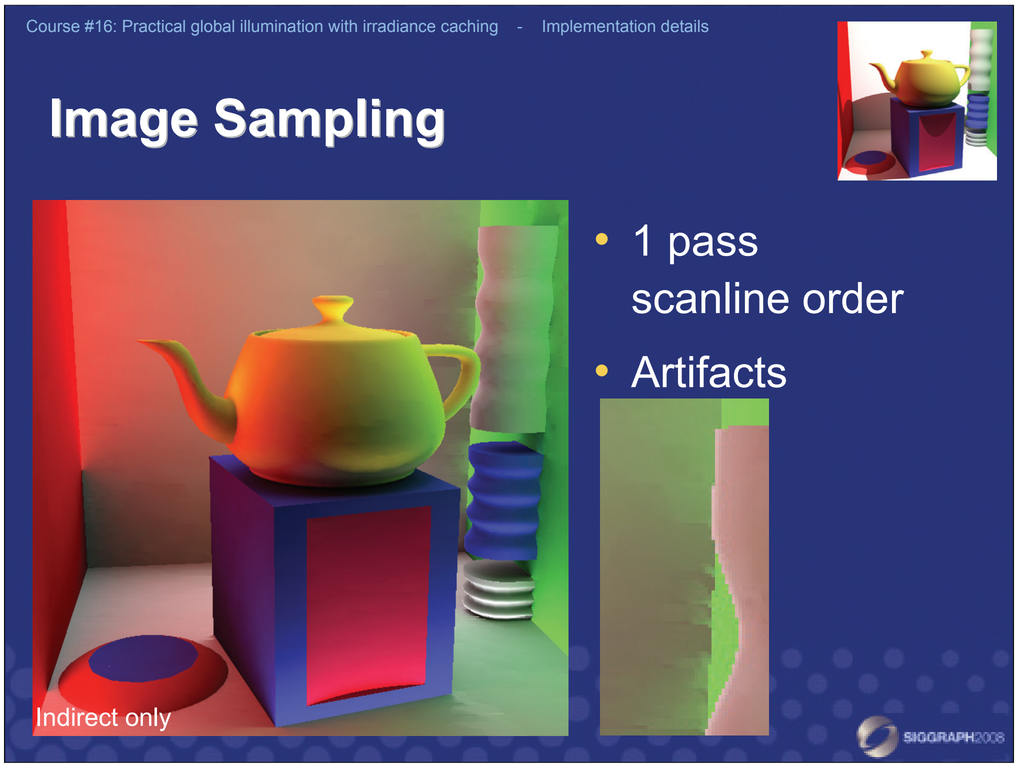
Course #16: Practical global illumination with irradiance caching - Implementation details

- ***Problem***:

  uneven interpolated irradiance

- *Solution*:

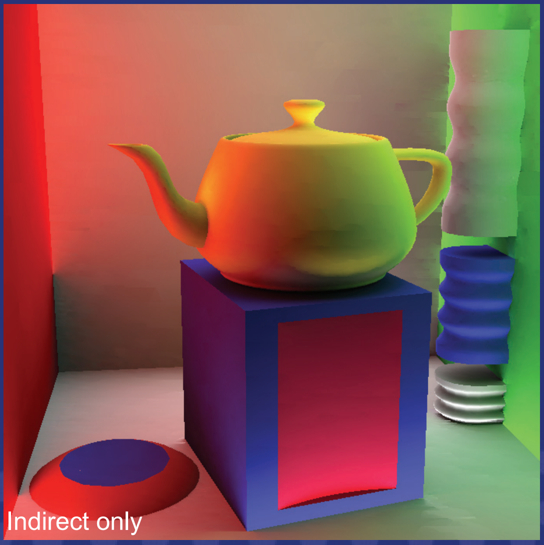
  (1) two-pass image sampling
  (2) bounded weighting function

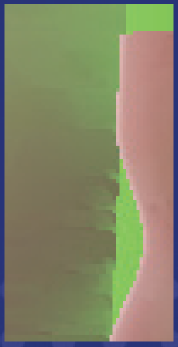


Course #16: Practical global illumination with irradiance caching - Implementation details

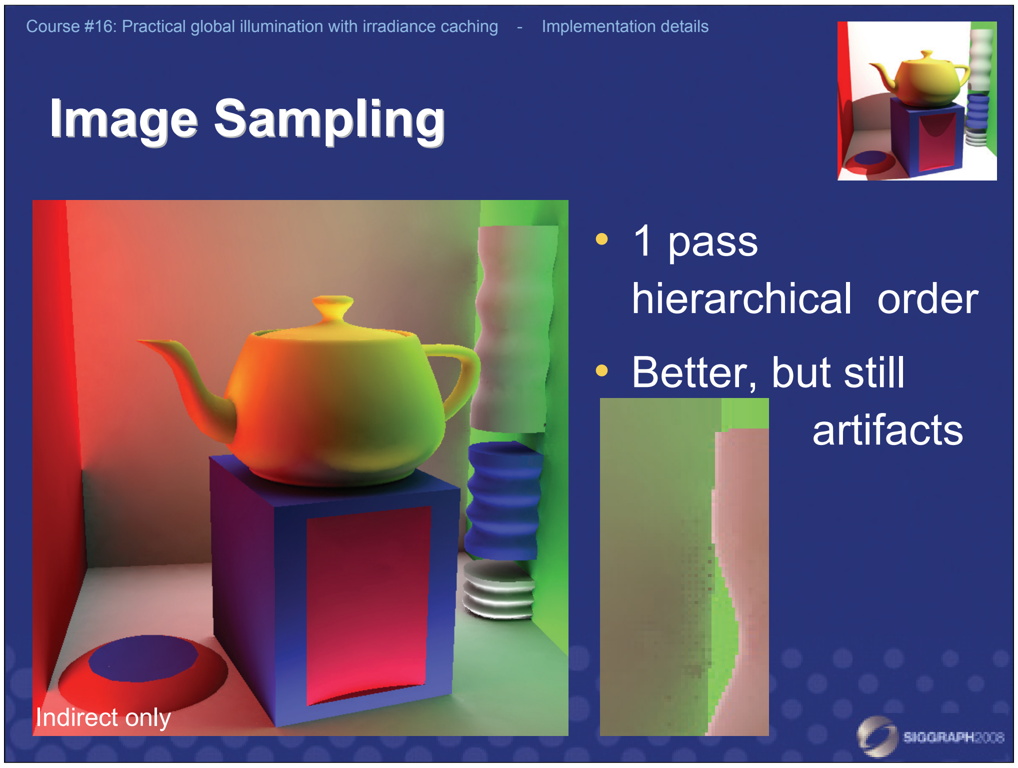
Image Sampling

- 1 pass
  scanline order
- Artifacts





Lazy evaluation of irradiance values is a great feature of irradiance caching that makes the algorithm very flexible. However, if not used carefully, it has a negative impact on the image quality. This slide shows the kind of artifacts you may expect when generating image pixels in the scanline order, adding new irradiance values lazily as needed.

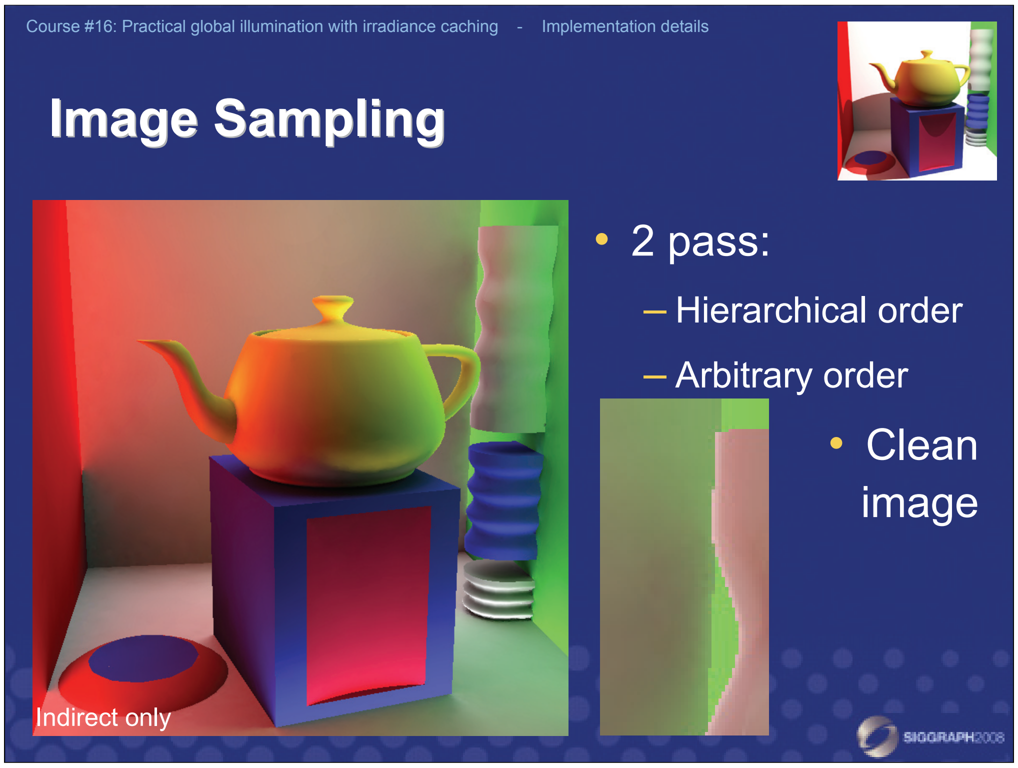


Course #16: Practical global illumination with irradiance caching - Implementation details

Image Sampling

- 1 pass hierarchical order
- Better, but still artifacts

Using hierarchical image traversal instead of the scanline order improves the image quality (and, actually, decreases the number of records needed to cover the whole image), but image artifacts still remain.



Course #16: Practical global illumination with irradiance caching - Implementation details

Image Sampling

- 2 pass:
  - Hierarchical order
  - Arbitrary order
- Clean image

In our experience, the best solution is a two pass traversal of the image. In the first pass, the irradiance cache is filled so that all pixels are covered, but no image is generated. In the second pass, arbitrary pixel traversal can be used to generate the image.

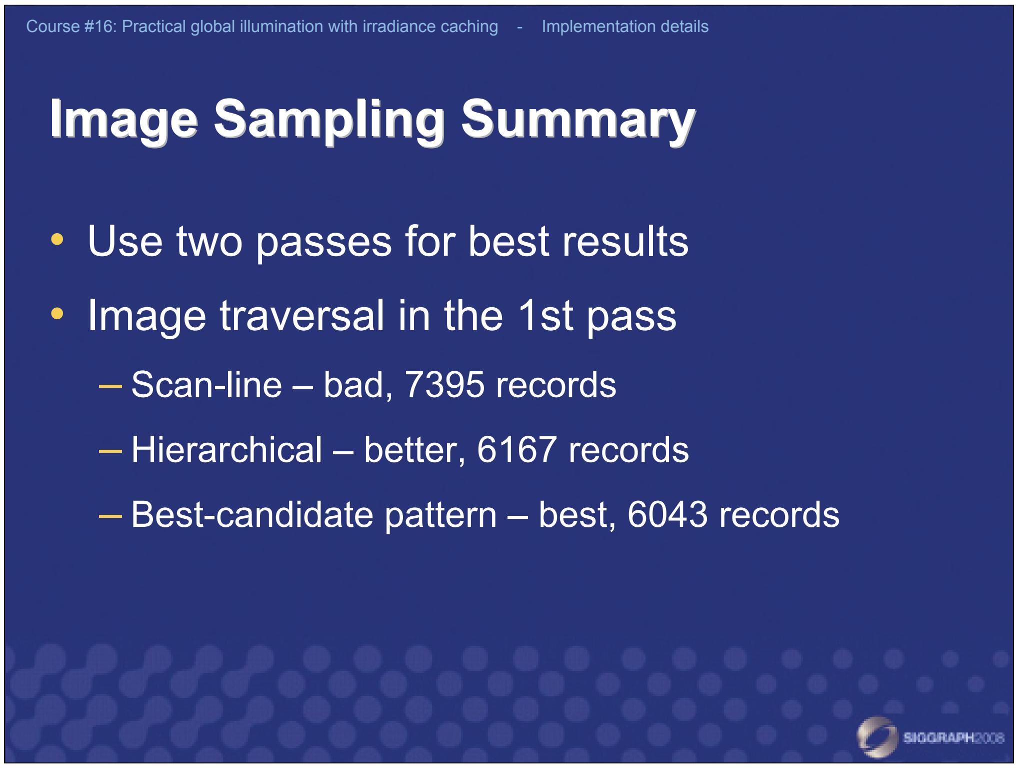
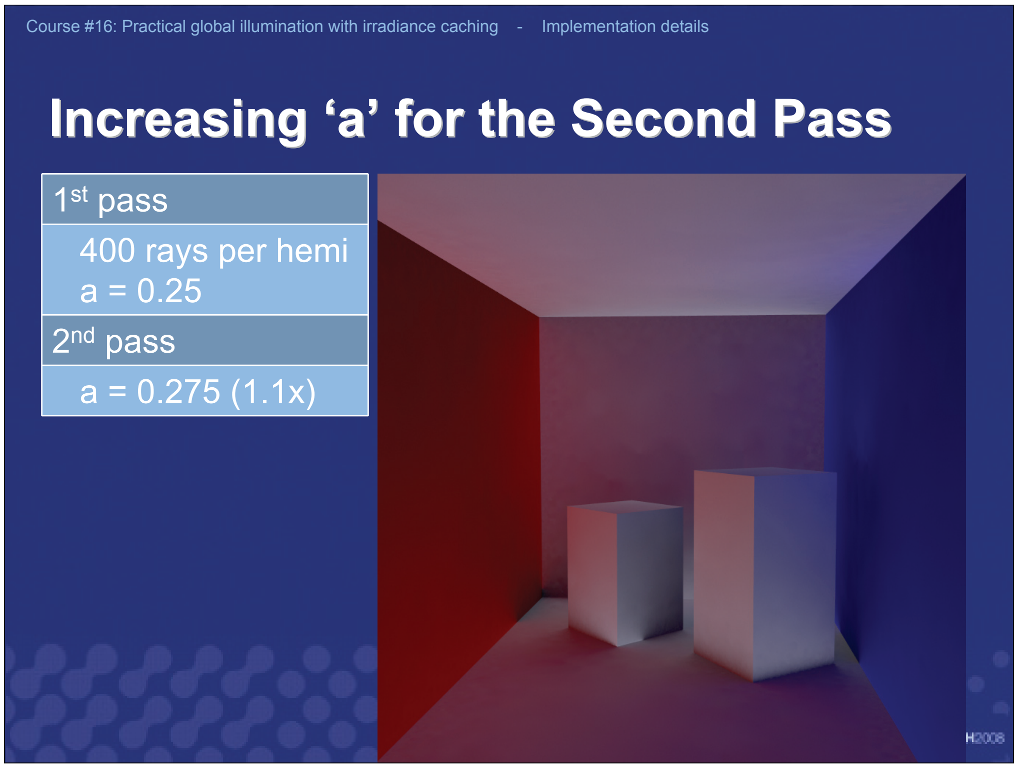

Course #16: Practical global illumination with irradiance caching - Implementation details

Image Sampling Summary

- Use two passes for best results
- Image traversal in the 1st pass
  - Scan-line – bad, 7395 records
  - Hierarchical – better, 6167 records
  - Best-candidate pattern – best, 6043 records

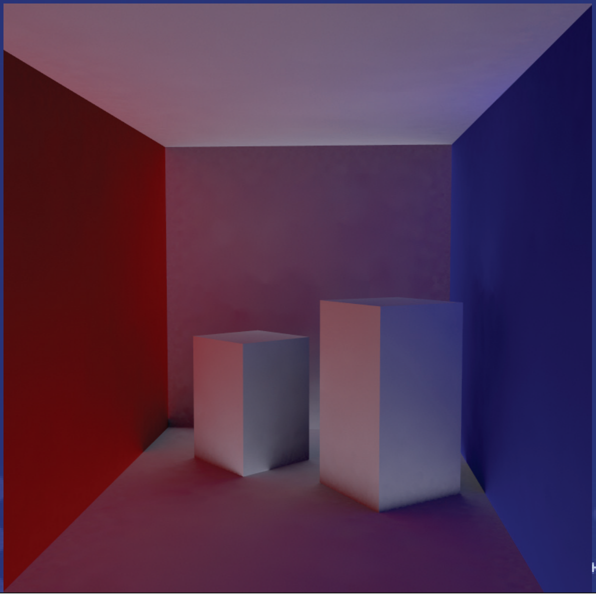
The order of pixel traversal in the first pass has an impact both on the image quality and also on the distribution of records in the scene and, consequently, on the number of records required to cover the whole image. A good solution is to use a best-candidate pattern pixel traversal. It generates the best image quality with the fewest records.



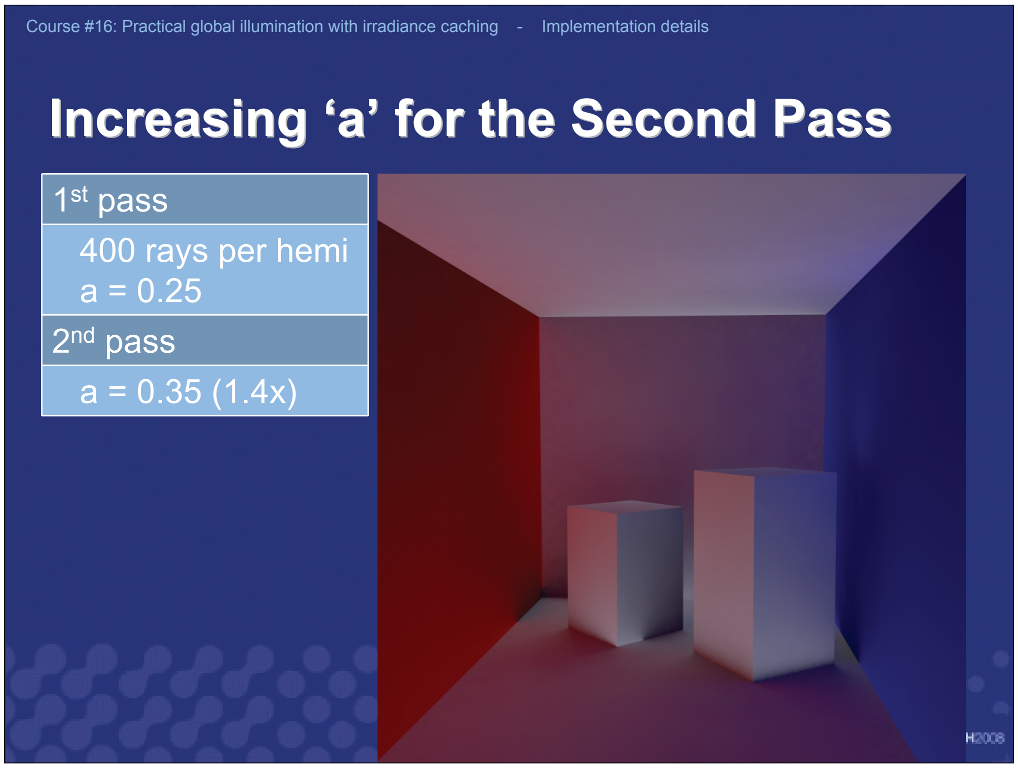
Course #16: Practical global illumination with irradiance caching - Implementation details

Increasing ‘a’ for the Second Pass

1st pass
400 rays per hemi a = 0.25
2nd pass
a = 0.275 (1.1x)



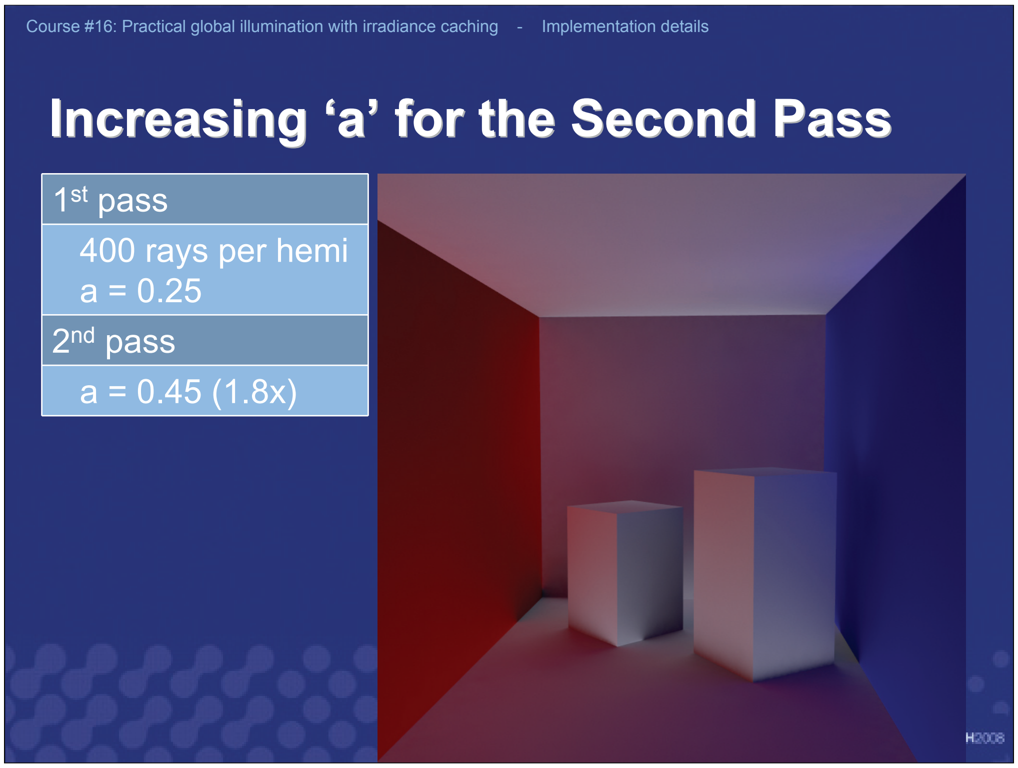
Another way to achieve smooth indirect illumination is to increase the area over which a record can be reused by increasing the ‘a’ value in the second pass.



Course #16: Practical global illumination with irradiance caching - Implementation details

Increasing ‘a’ for the Second Pass

- 1st pass
  - 400 rays per hemi
  - a = 0.25
- 2nd pass
  - a = 0.35 (1.4x)

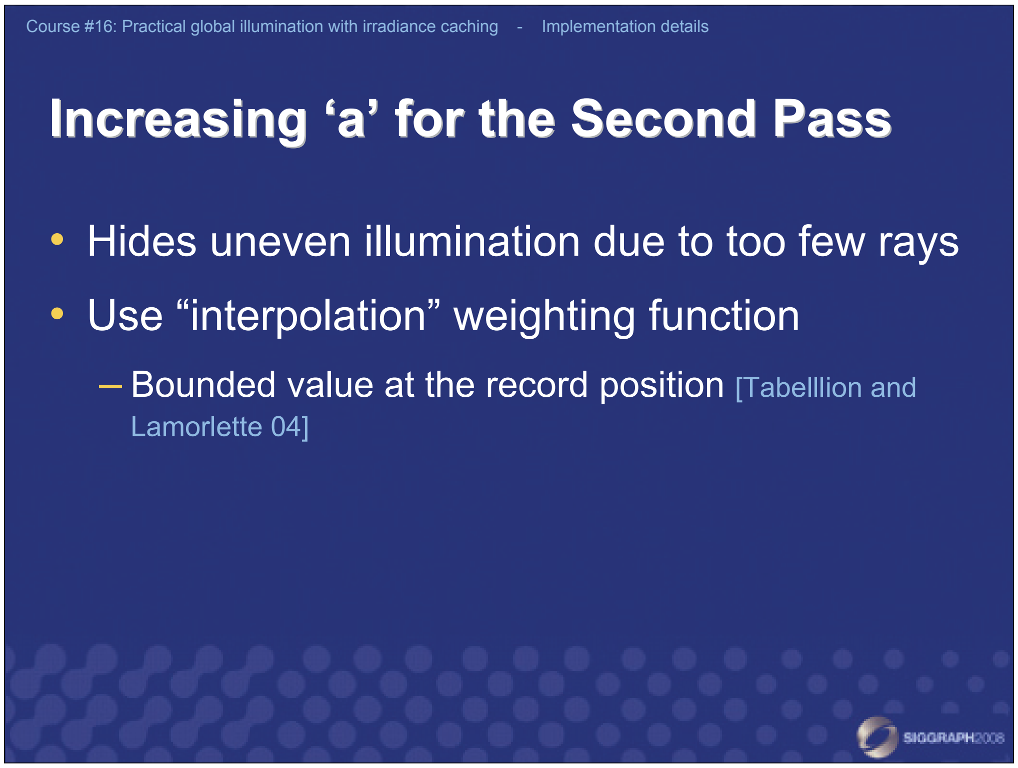


Course #16: Practical global illumination with irradiance caching - Implementation details

Increasing ‘a’ for the Second Pass

1st pass
400 rays per hemi a = 0.25
2nd pass
a = 0.45 (1.8x)

Using a very large value of `a’ in the second pass makes it possible to obtain relatively smooth indirect illumination even for a small number of rays in hemisphere sampling (here 400 per hemisphere), at the cost of blurring some of its details. However, blurring is rarely a problem since indirect illumination is mostly smooth by its nature anyhow.

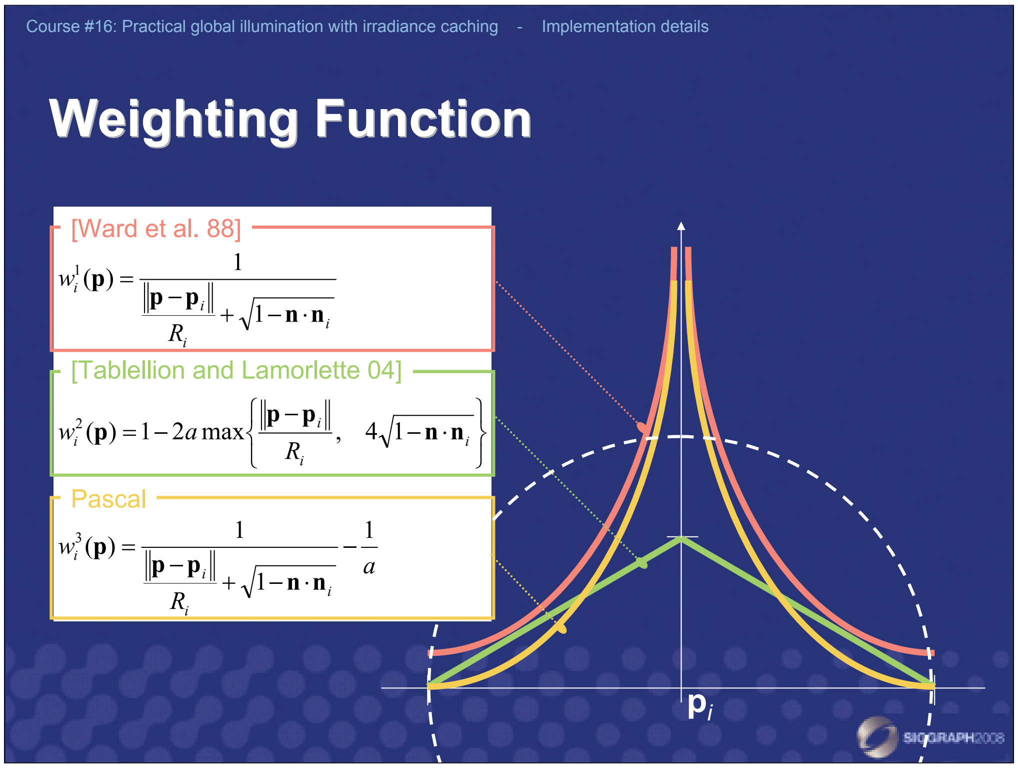


Course #16: Practical global illumination with irradiance caching - Implementation details

Increasing ‘a’ for the Second Pass

- Hides uneven illumination due to too few rays
- Use “interpolation” weighting function
  - Bounded value at the record position [Tabelllion and Lamorlette 04]

To achieve the desired smoothing effect when increasing the `a’ value, we have to pay attention to using a suitable weighting function for interpolation.



Course #16: Practical global illumination with irradiance caching - Implementation details

Weighting Function

[Ward et al. 88]

$$w_i^1(\mathbf{p}) = \frac{1}{\frac{\|\mathbf{p}-\mathbf{p}_i\|}{R_i} + \sqrt{1-\mathbf{n}\cdot\mathbf{n}_i}}$$

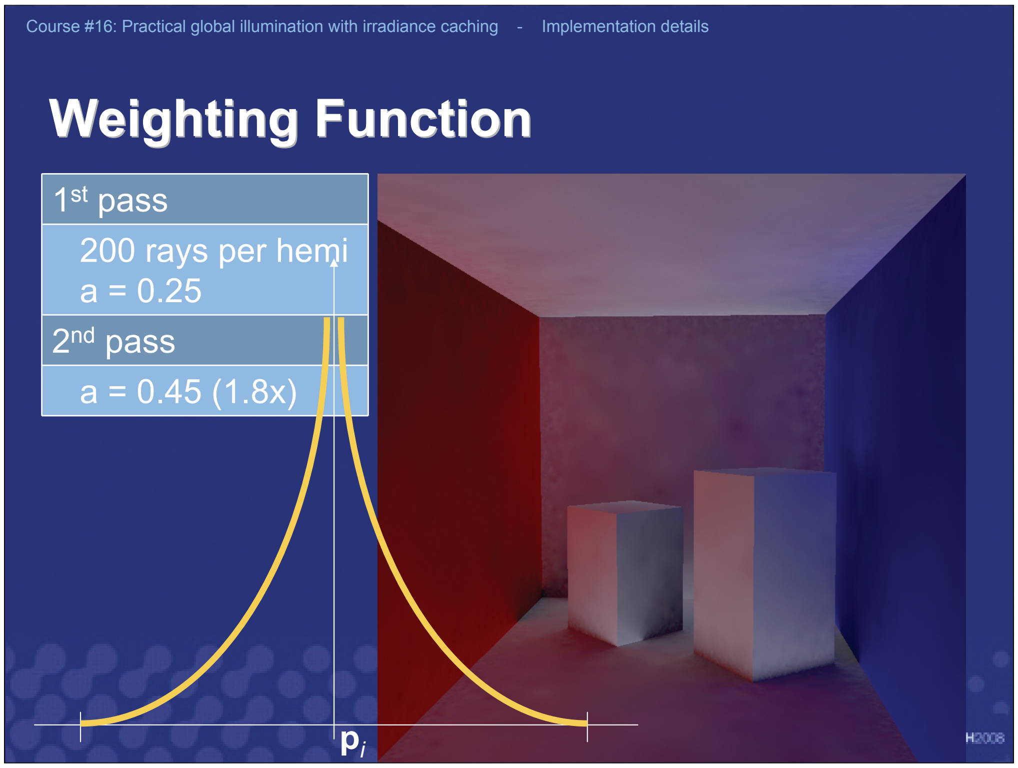
[Tablellion and Lamorlette 04]

$$w_i^2(\mathbf{p}) = 1 - 2a\max\left\{\frac{\|\mathbf{p}-\mathbf{p}_i\|}{R_i},\quad 4\sqrt{1-\mathbf{n}\cdot\mathbf{n}_i}\right\}$$

Pascal

$$w_i^3(\mathbf{p}) = \frac{1}{\frac{\|\mathbf{p}-\mathbf{p}_i\|}{R_i} + \sqrt{1-\mathbf{n}\cdot\mathbf{n}_i}} - \frac{1}{a}$$

The original weighting function proposed by Ward et al. [1988] has two undesirable properties. First, it goes to infinity when the distance between the point of interpolation **p** and the location of a record, **p**$_i$, goes to zero. Second, there is a discontinuity at the border of the influence area of a record. The discontinuity tends to produce some visible seams in the images. One approach to suppressing the seams, used in *Radiance*, is to randomize the acceptance of a record for interpolation. In our experience, a better way is to make the weight function zero at the border of the influence area – two possibilities are shown on the slide.



Course #16: Practical global illumination with irradiance caching - Implementation details

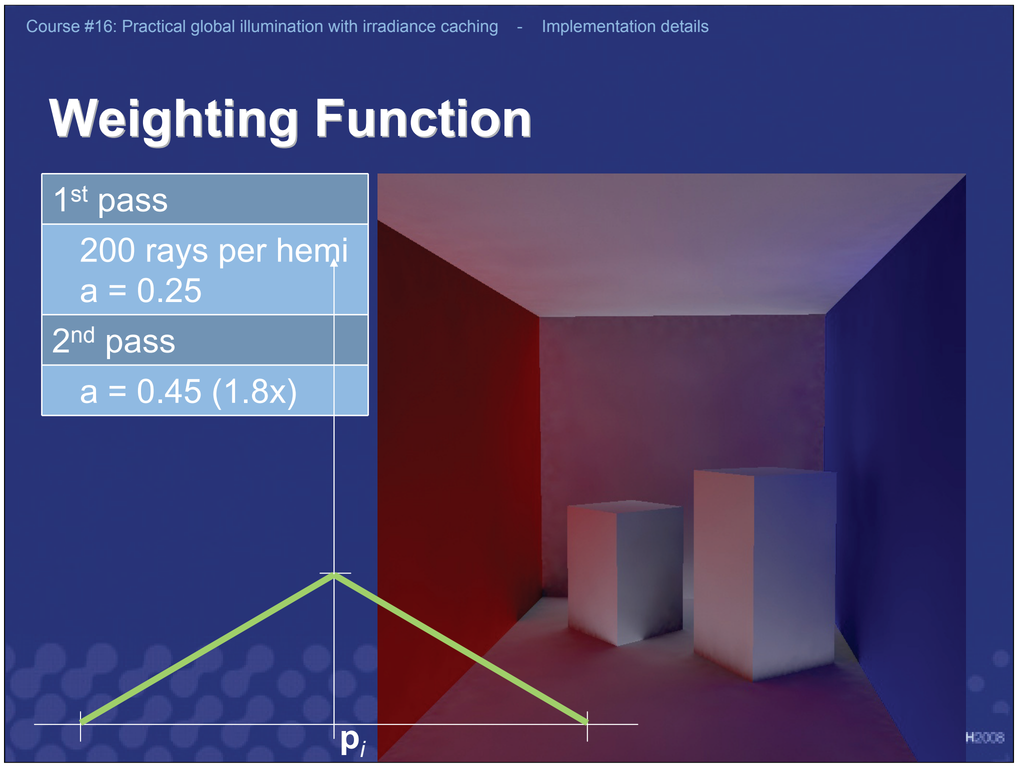
Weighting Function

1st pass
- 200 rays per hemi
- a = 0.25

2nd pass
- a = 0.45 (1.8x)

$\mathbf{p}_i$
H2008

Even when the weight is zero at the border, it may still be virtually impossible to get smooth indirect illumination. The reason is the high value of the weight at the record location.

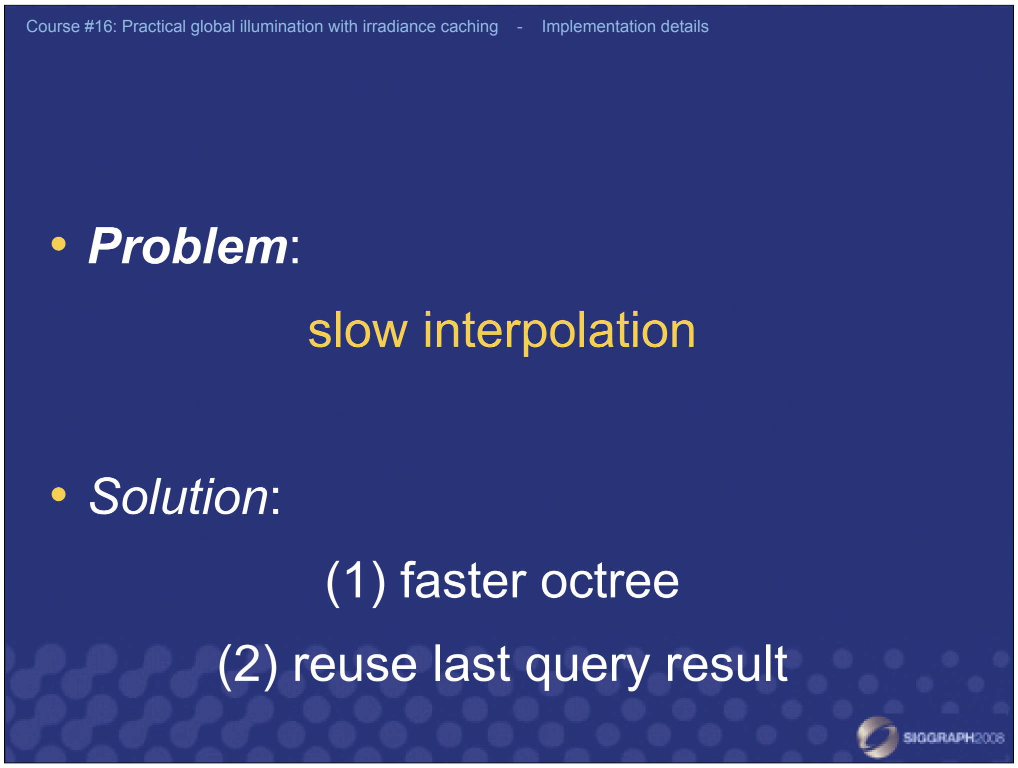


Course #16: Practical global illumination with irradiance caching - Implementation details

Weighting Function

1st pass
200 rays per hemi a = 0.25
2nd pass
a = 0.45 (1.8x)

Using a weighting function that is bounded at the record location, such as the one proposed by Tabellion and Lamorlette [2004], the interpolated illumination becomes much smoother.



Course #16: Practical global illumination with irradiance caching - Implementation details

- ***Problem***:

  slow interpolation

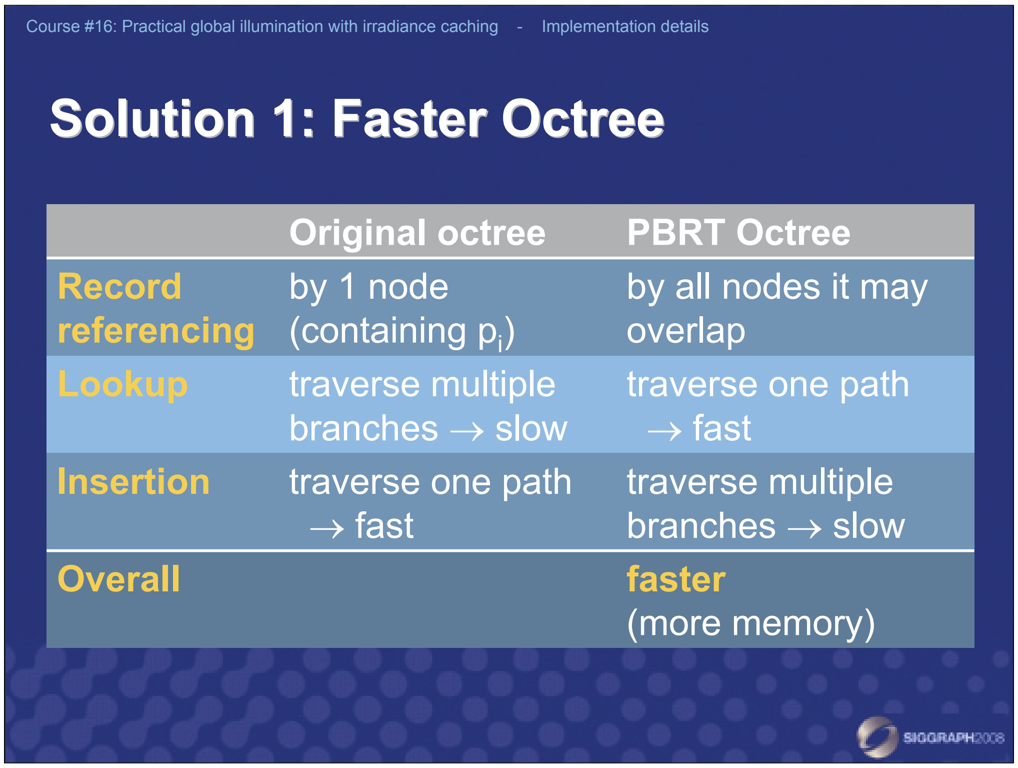
- *Solution*:

  (1) faster octree

  (2) reuse last query result

SIGGRAPH2008

Tens of millions of irradiance cache queries are used to generate a production-quality image. Therefore, it is important to keep the interpolation time as low as possible. We use two techniques to improve the interpolation speed: faster octree for indexing the records and reuse of last query result.



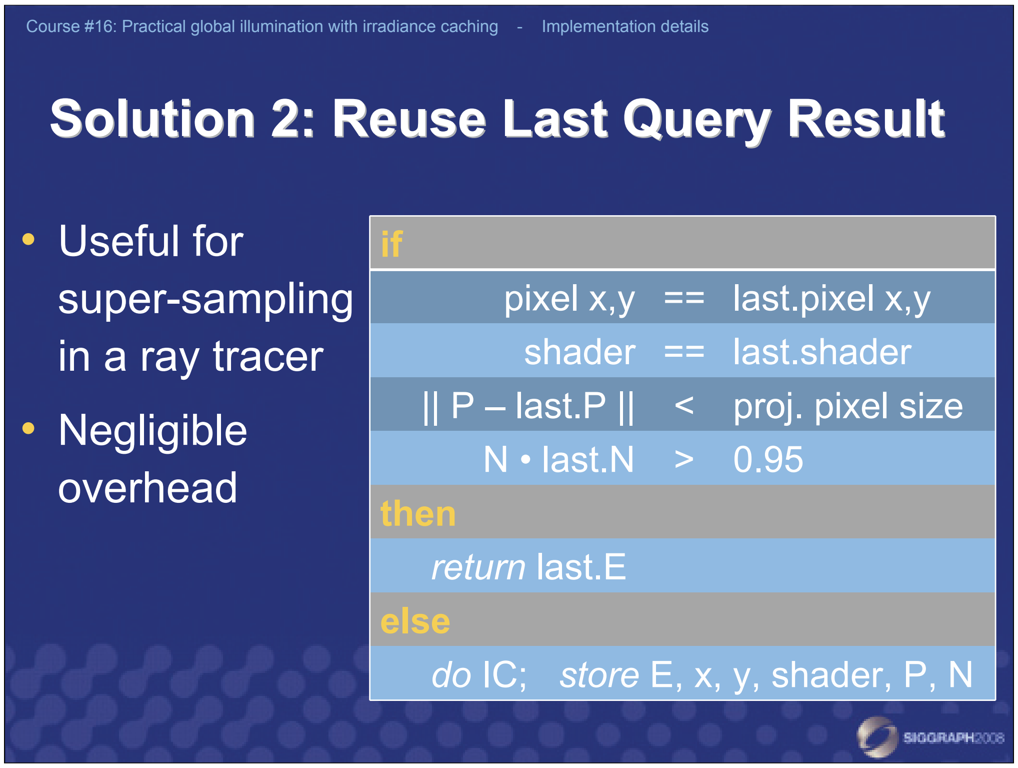
Course #16: Practical global illumination with irradiance caching - Implementation details

Solution 1: Faster Octree

	Original octree	PBRT Octree
Record referencing	by 1 node (containing p_i)	by all nodes it may overlap
Lookup	traverse multiple branches $\rightarrow$ slow	traverse one path $\rightarrow$ fast
Insertion	traverse one path $\rightarrow$ fast	traverse multiple branches $\rightarrow$ slow
Overall		**faster** (more memory)

SIGGRAPH2008

The octree structure proposed for irradiance caching by Ward et al. [1988] (used in the *Radiance* implementation) references one record from exactly one node. As a consequence, searching for all records contributing to interpolation at a given location involves traversing several path in the tree, which means recursion. This makes the tree lookup relatively slow. A better option is to reference each record from all octree nodes that the record may overlap. The tree lookup then reduces to a simple traversal form the root to the leaf. A simple implementation of this improved octree is a part of PBRT [Pharr and Humphreys 2004]. The price to pay for the faster lookup is a slightly increased memory consumption and slower insertion.



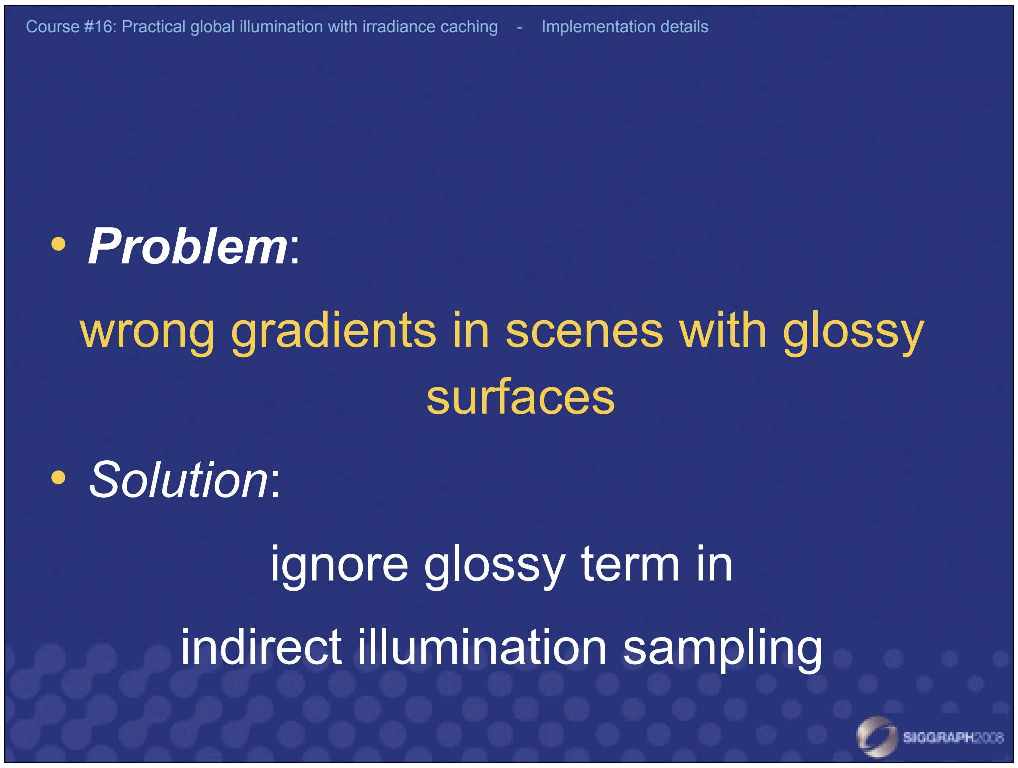
Course #16: Practical global illumination with irradiance caching - Implementation details

Solution 2: Reuse Last Query Result

- Useful for super-sampling in a ray tracer
- Negligible overhead

```
if
        pixel x,y  ==  last.pixel x,y
           shader  ==  last.shader
   || P – last.P ||  <  proj. pixel size
      N • last.N   >   0.95
then
    return last.E
else
    do IC;   store E, x, y, shader, P, N
```

Our second trick to amortize the cost of irradiance cache lookups only applies to ray tracing-based renderers (unlike REYES & comp.). When generating a production-quality image with ray tracing, 16x or higher pixel super sampling is quite common in practice. However, the color returned by each irradiance cache query within the same pixel is usually virtually identical. We exploit this observation by retaining the result of the IC query and returning the retained value for all other queries within the same pixel (subject to some further checks as shown on the slide).



- ***Problem***:

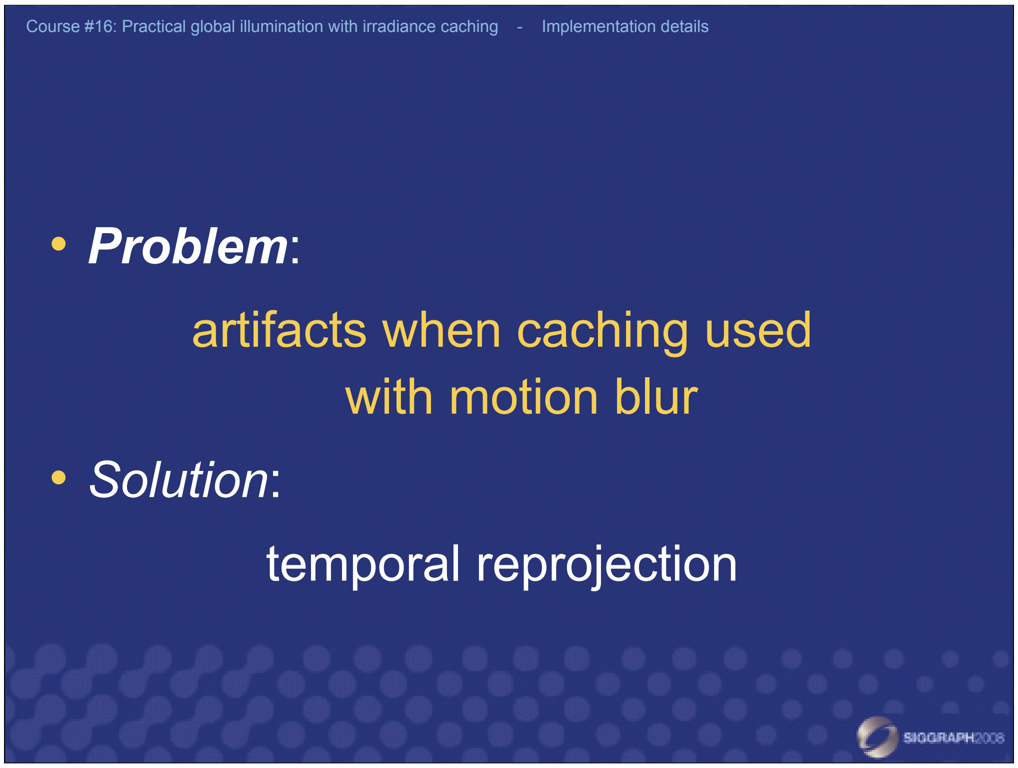
  wrong gradients in scenes with glossy surfaces

- *Solution*:

  ignore glossy term in

  indirect illumination sampling

When rendering scenes with some glossy or specular surfaces, the results of the translational gradient computation are often incorrect, resulting in artifacts in the image. The most common solution to this problem is to ignore the glossy term for shader evaluations launched by the rays generated by hemisphere sampling. Doing so ignores a part of indirect illumination but most often this is preferable to visible image artifacts.


Course #16: Practical global illumination with irradiance caching - Implementation details

- ***Problem***:

  artifacts when caching used with motion blur

- *Solution*:

  temporal reprojection

SIGGRAPH2008

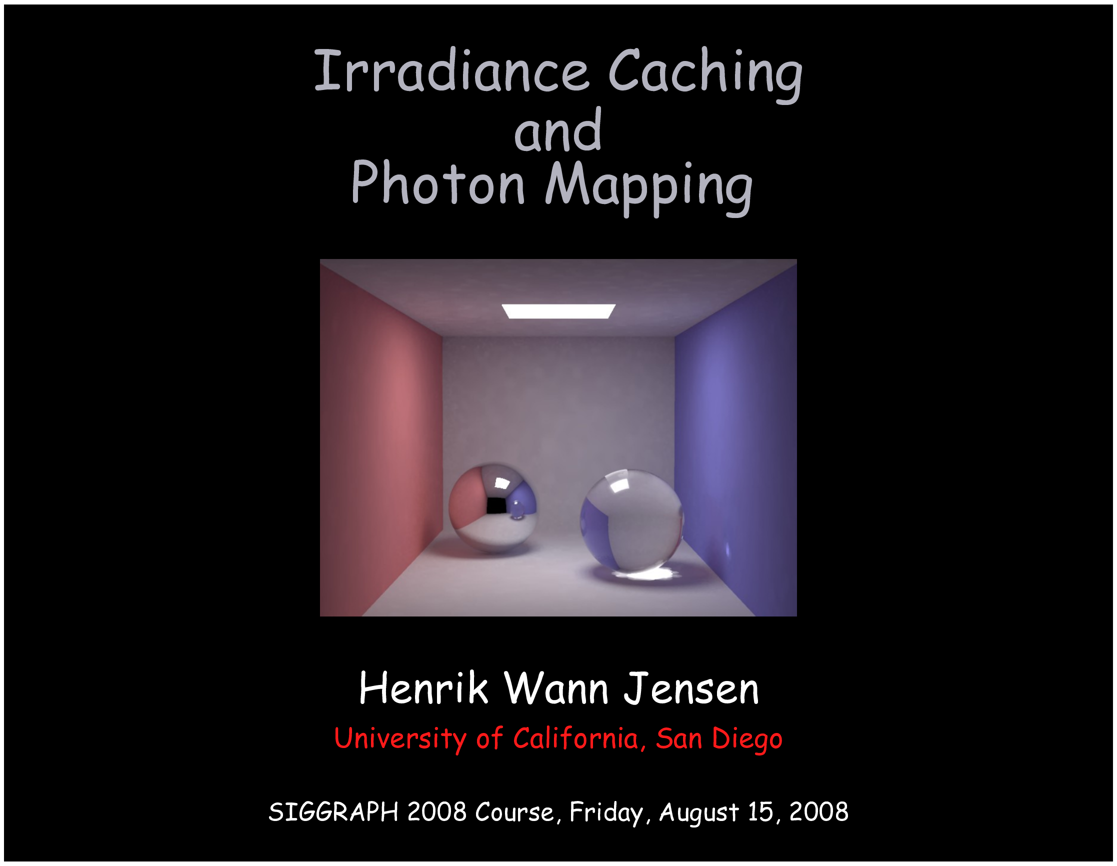
In many scanline renderers, such as Pixar's PRMan, shading is computed before resolving visibility and motion blur. It means that irradiance caching is not affected by the presence of the motion blur in the scene and works fine with motion blurred sequences.

However, for ray tracing-based renderers, the situation is more complicated. Each generated primary ray is assigned a randomly generated time. When tracing this ray, transformation / deformation of animated geometry used for intersection testing is adjusted to that time. If an irradiance cache record is generated for some time value, its irradiance will be incorrect for any other time during the frame. Moreover, lots of records are created because of the movement of the animated geometry.

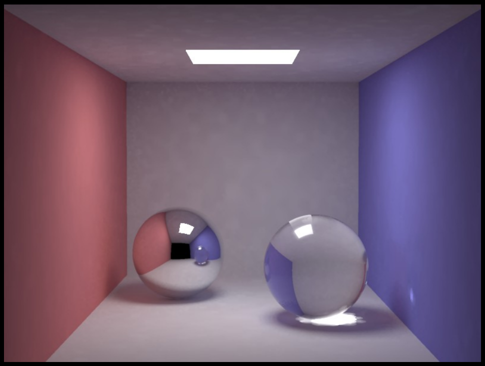
The workaround for this problem is to reproject any intersection position to time t=0 before the irradiance cache query. This is somewhat equivalent to what happens in the PRMan.

Course #16: Practical global illumination with irradiance caching - Implementation details

?

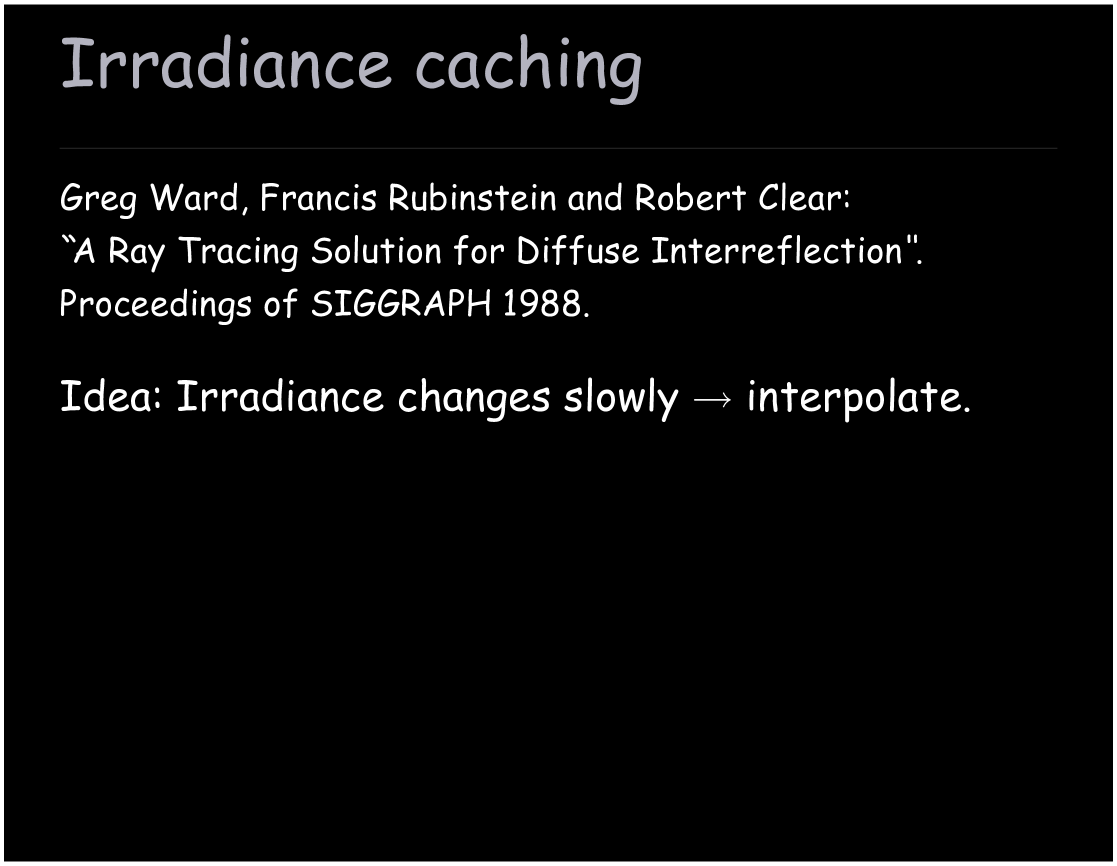


Irradiance Caching and Photon Mapping



Henrik Wann Jensen
University of California, San Diego

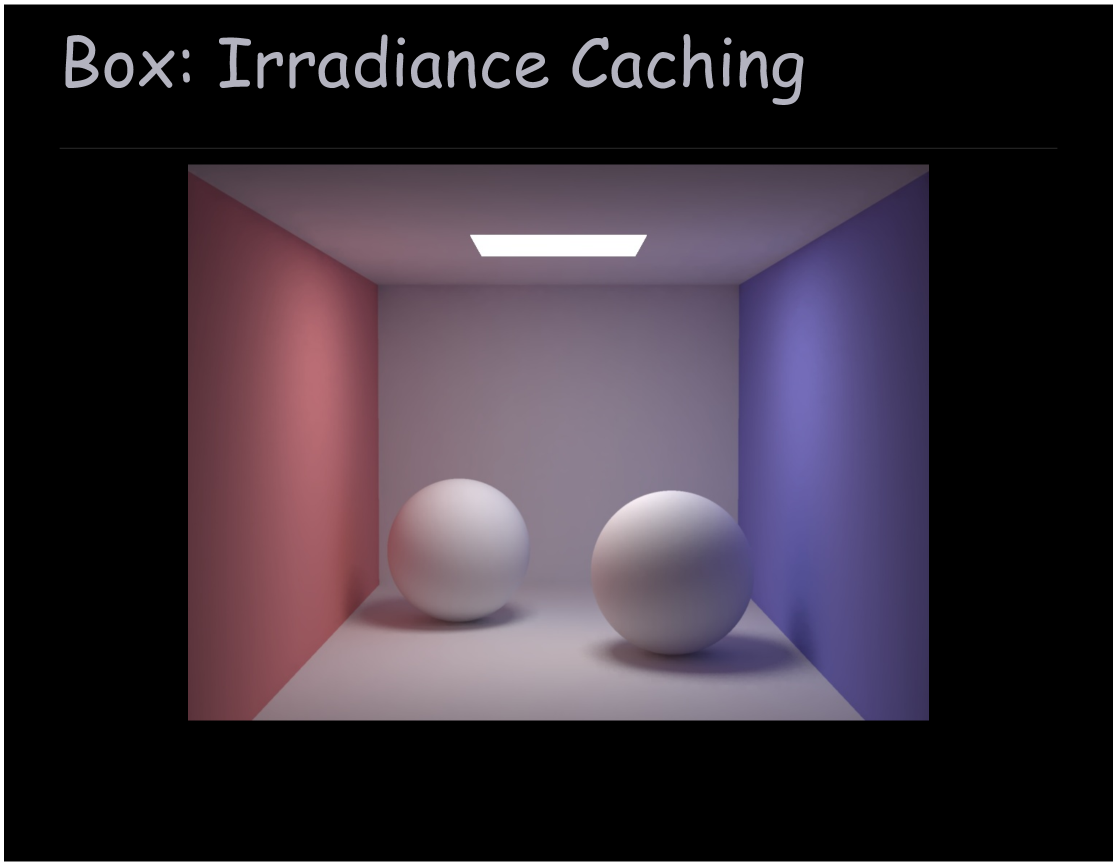
SIGGRAPH 2008 Course, Friday, August 15, 2008



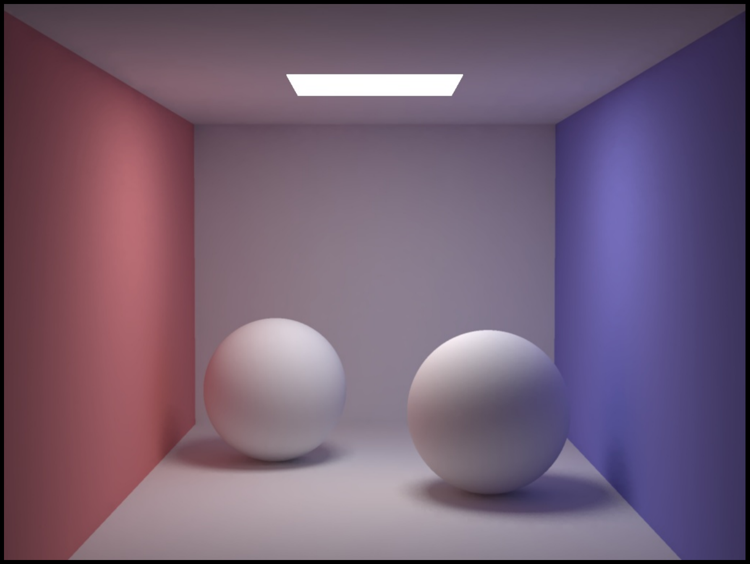
Irradiance caching

Greg Ward, Francis Rubinstein and Robert Clear:
"A Ray Tracing Solution for Diffuse Interreflection".
Proceedings of SIGGRAPH 1988.

Idea: Irradiance changes slowly → interpolate.

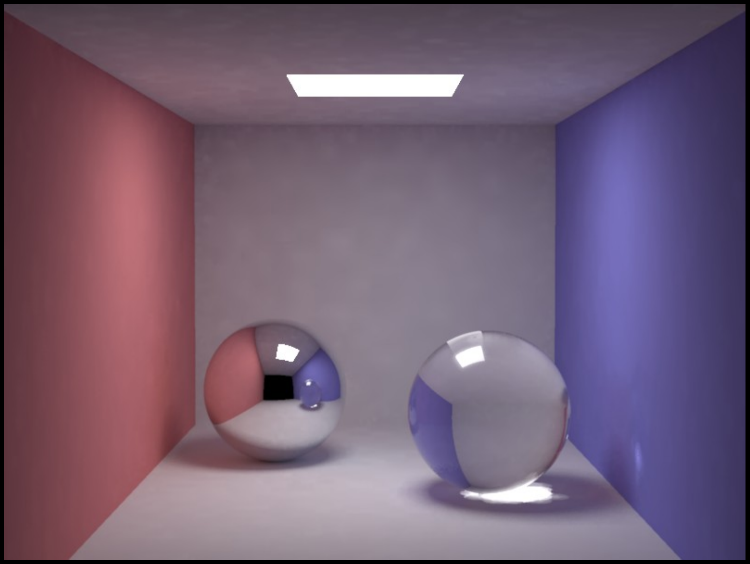


Box: Irradiance Caching



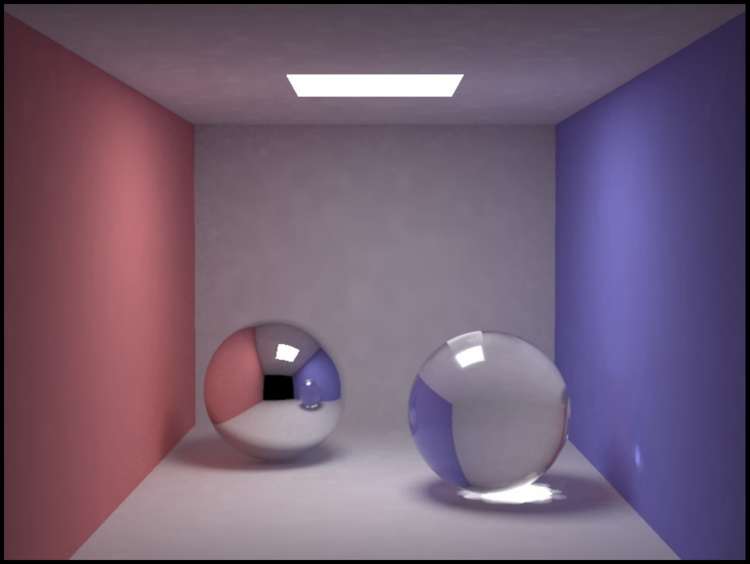
H. Wann Jensen

Box: Irradiance Caching

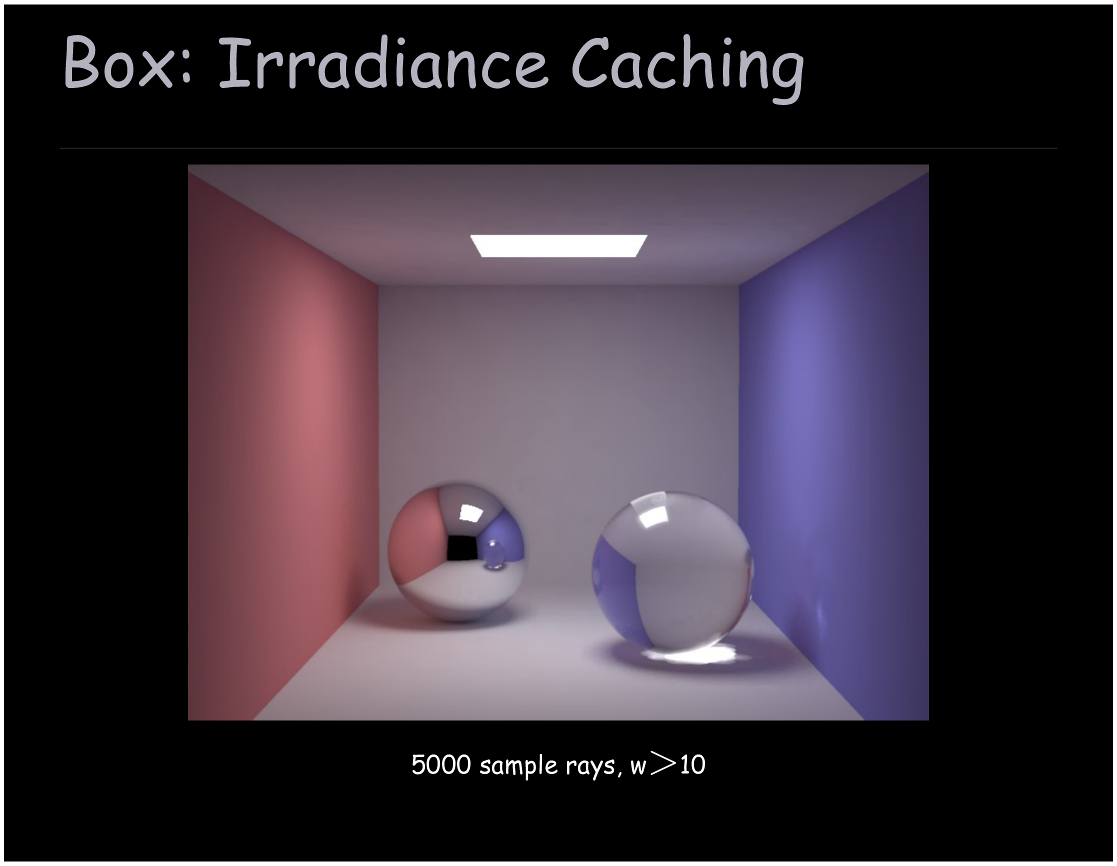


1000 sample rays, $w > 10$

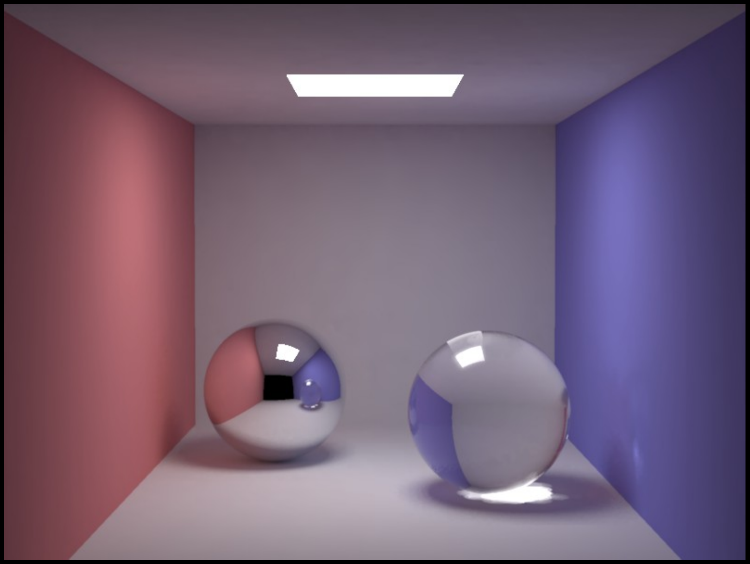
Box: Irradiance Caching



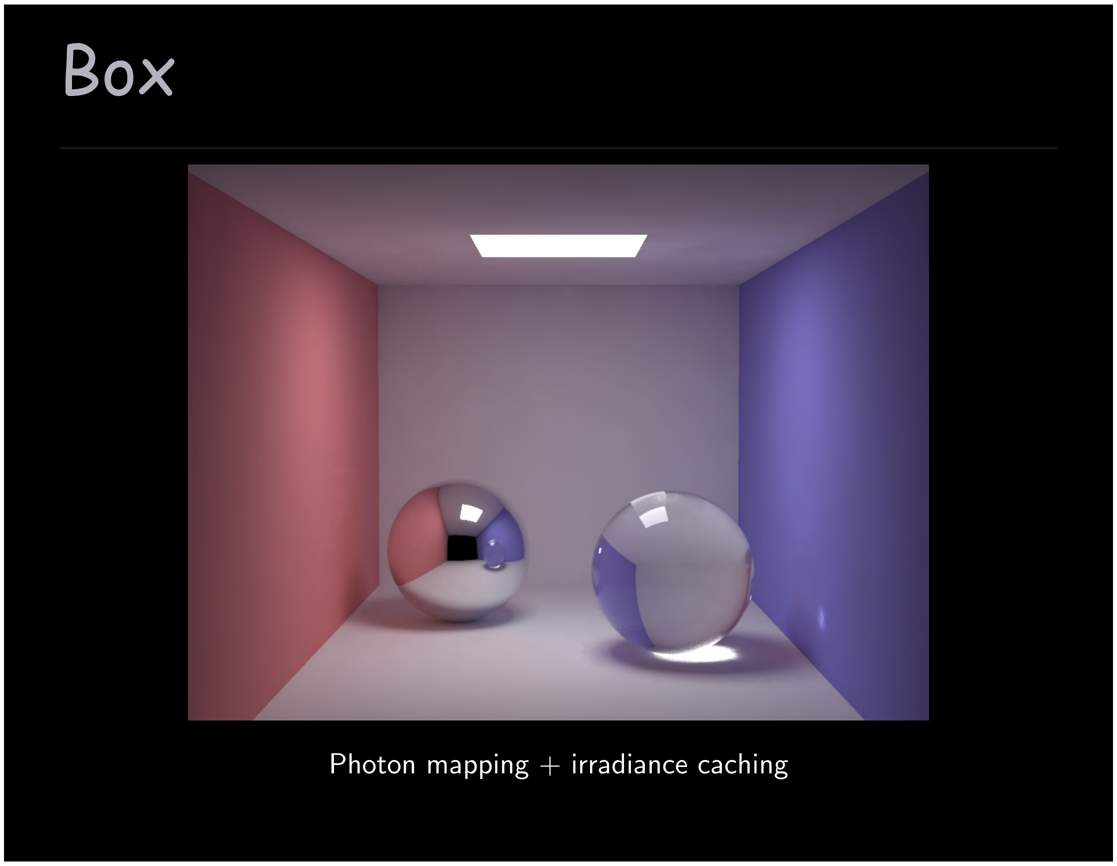
1000 sample rays, w>20



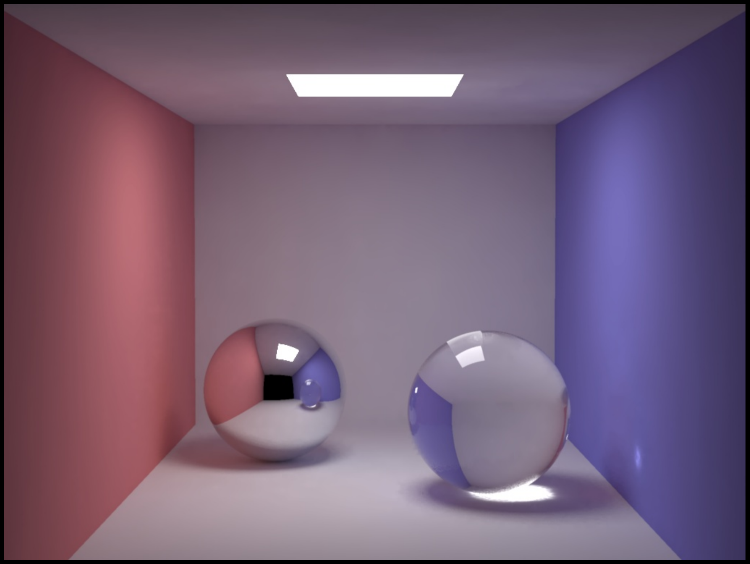
Box: Irradiance Caching



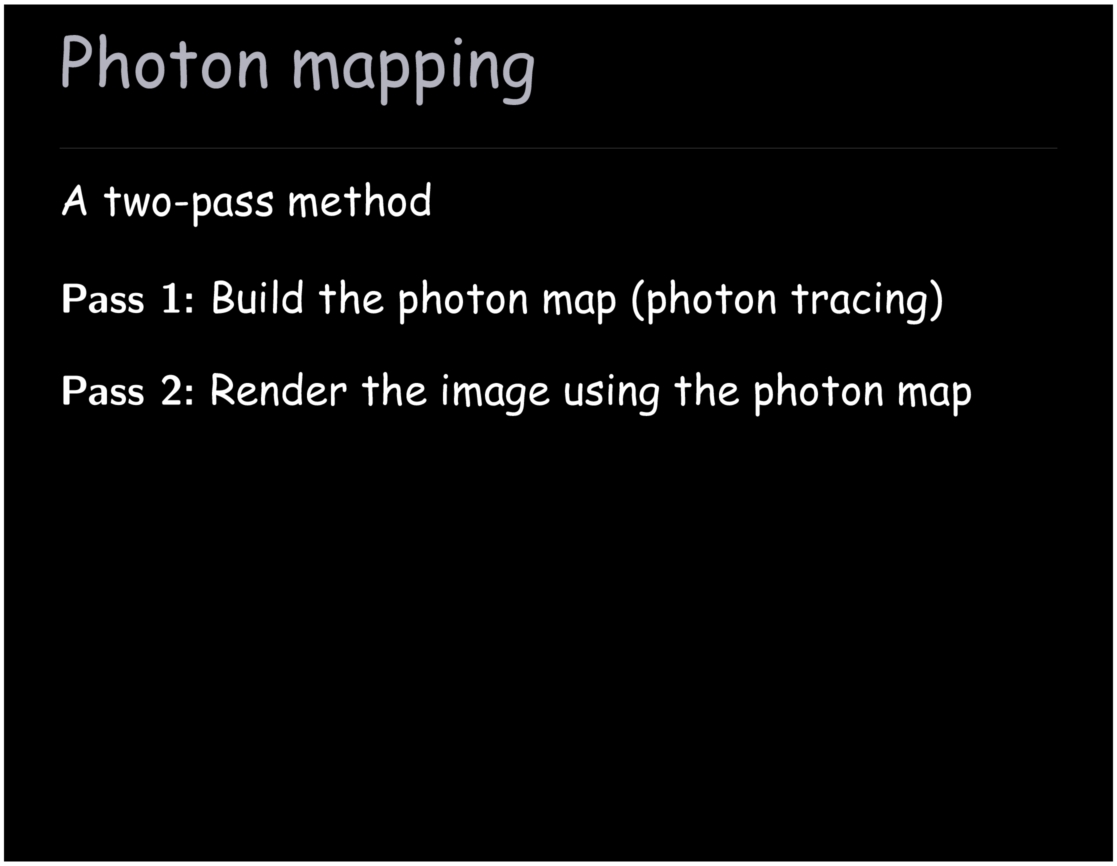
5000 sample rays, w>10



Box



Photon mapping + irradiance caching



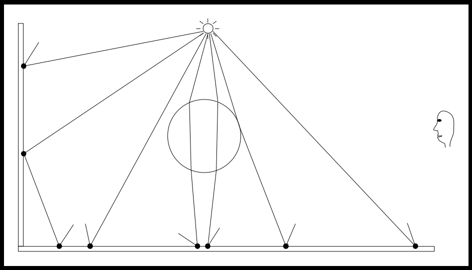
Photon mapping

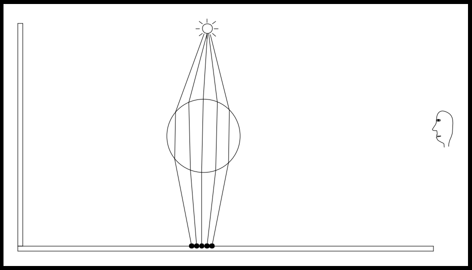
A two-pass method

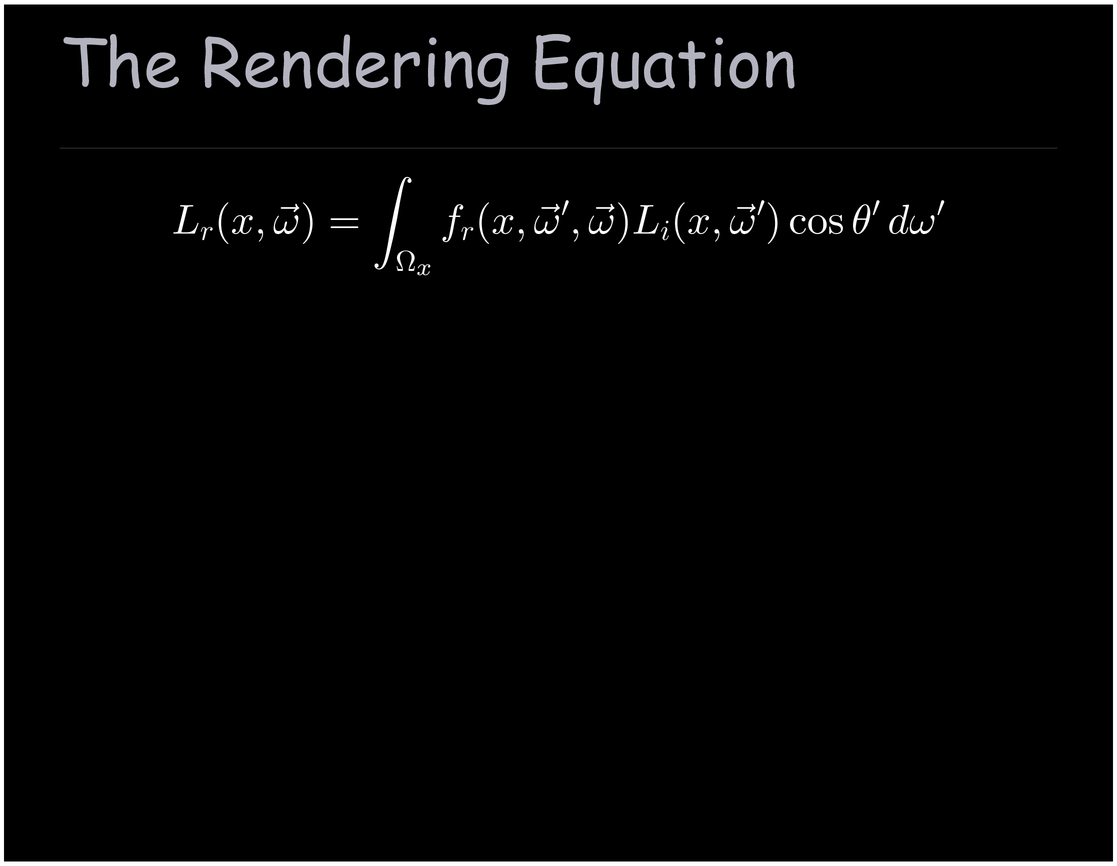
Pass 1: Build the photon map (photon tracing)

Pass 2: Render the image using the photon map

Two photon maps

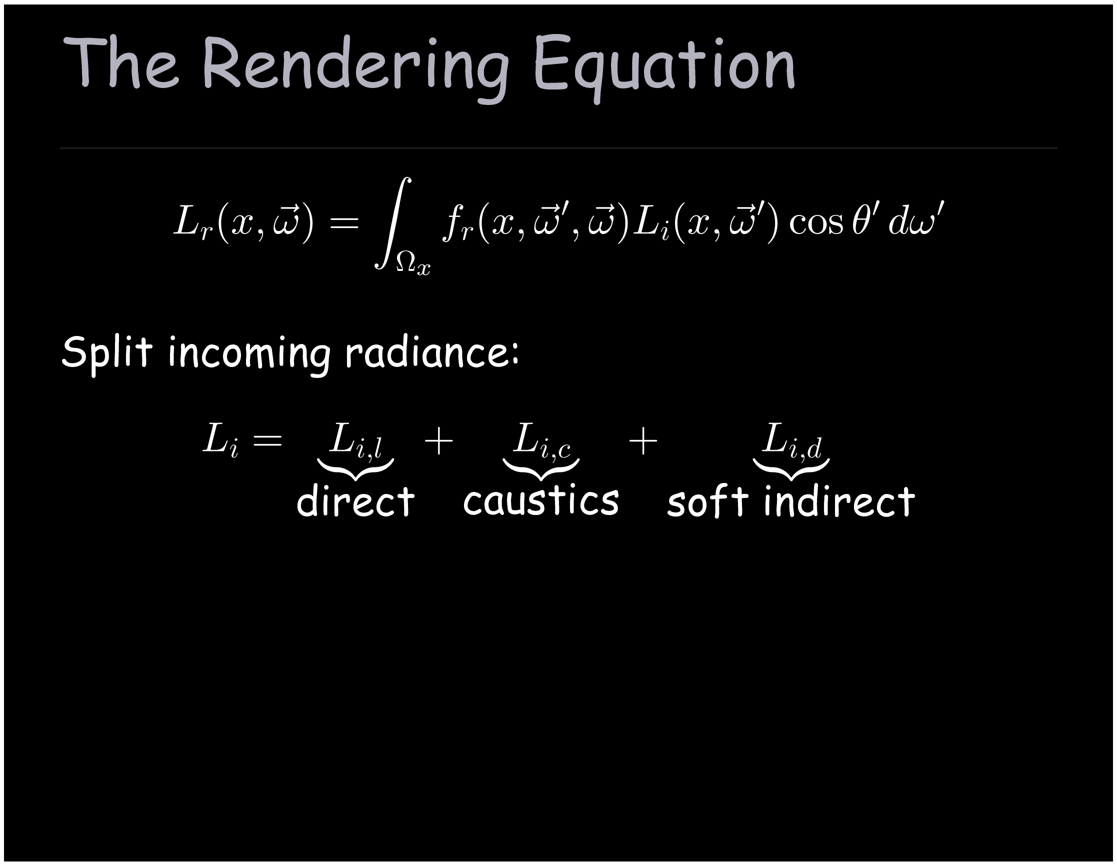

global photon map


caustics photon map



The Rendering Equation

$$L_r(x,\vec{\omega}) = \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) L_i(x,\vec{\omega}') \cos\theta' \, d\omega'$$

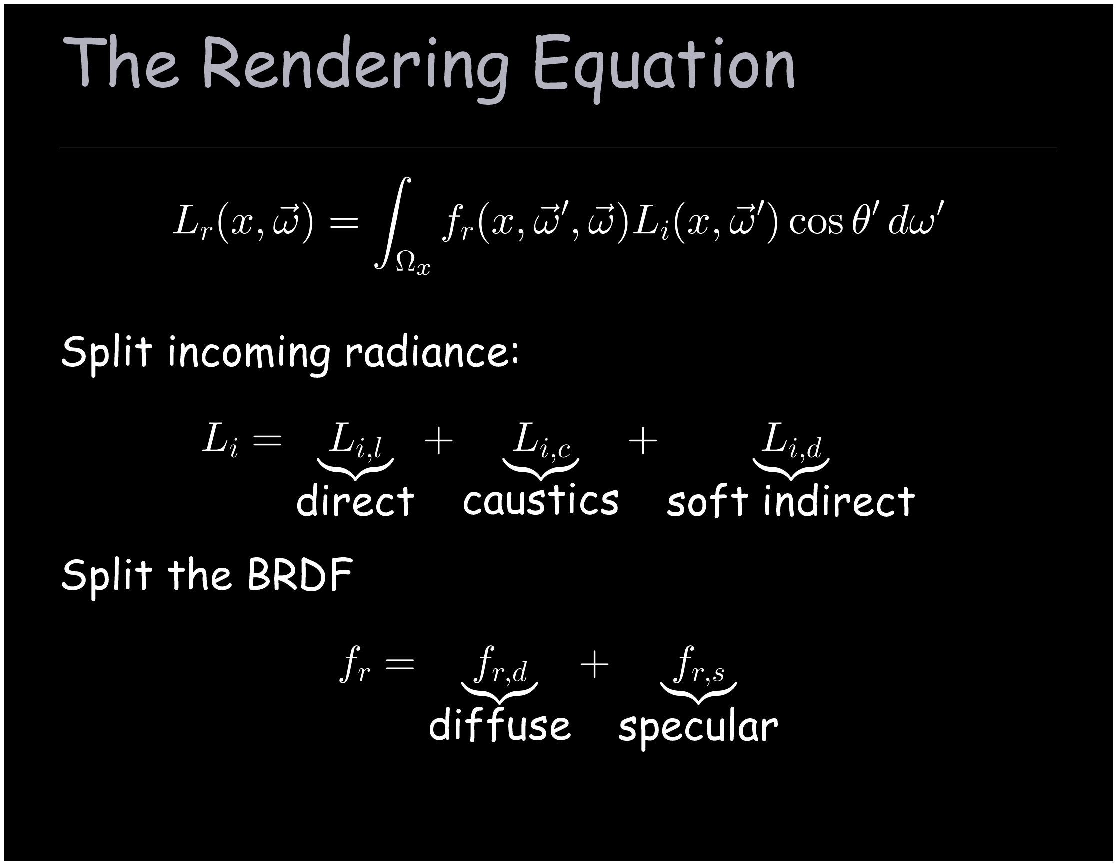


The Rendering Equation

$$L_r(x,\vec{\omega}) = \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) L_i(x,\vec{\omega}') \cos\theta' \, d\omega'$$

Split incoming radiance:

$$L_i = \underbrace{L_{i,l}}_{\text{direct}} + \underbrace{L_{i,c}}_{\text{caustics}} + \underbrace{L_{i,d}}_{\text{soft indirect}}$$



The Rendering Equation

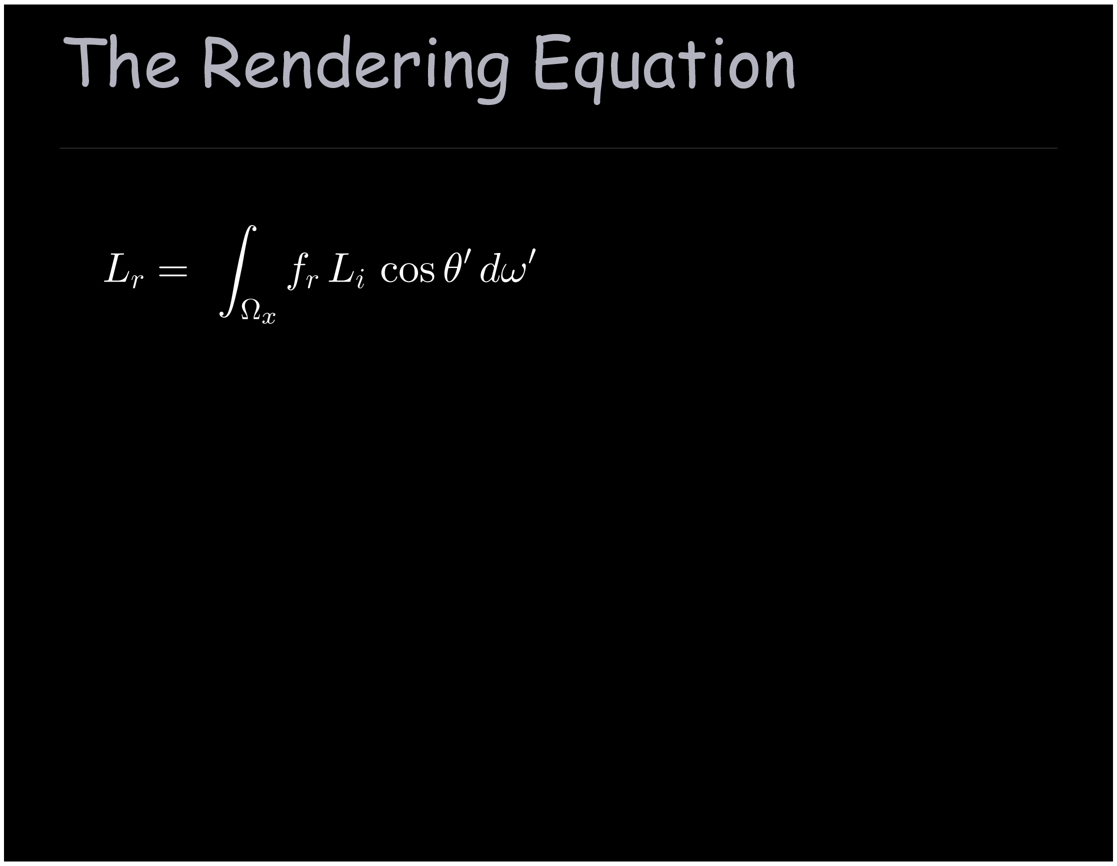
$$L_r(x,\vec{\omega}) = \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) L_i(x,\vec{\omega}') \cos\theta' \, d\omega'$$

Split incoming radiance:

$$L_i = \underbrace{L_{i,l}}_{\text{direct}} + \underbrace{L_{i,c}}_{\text{caustics}} + \underbrace{L_{i,d}}_{\text{soft indirect}}$$

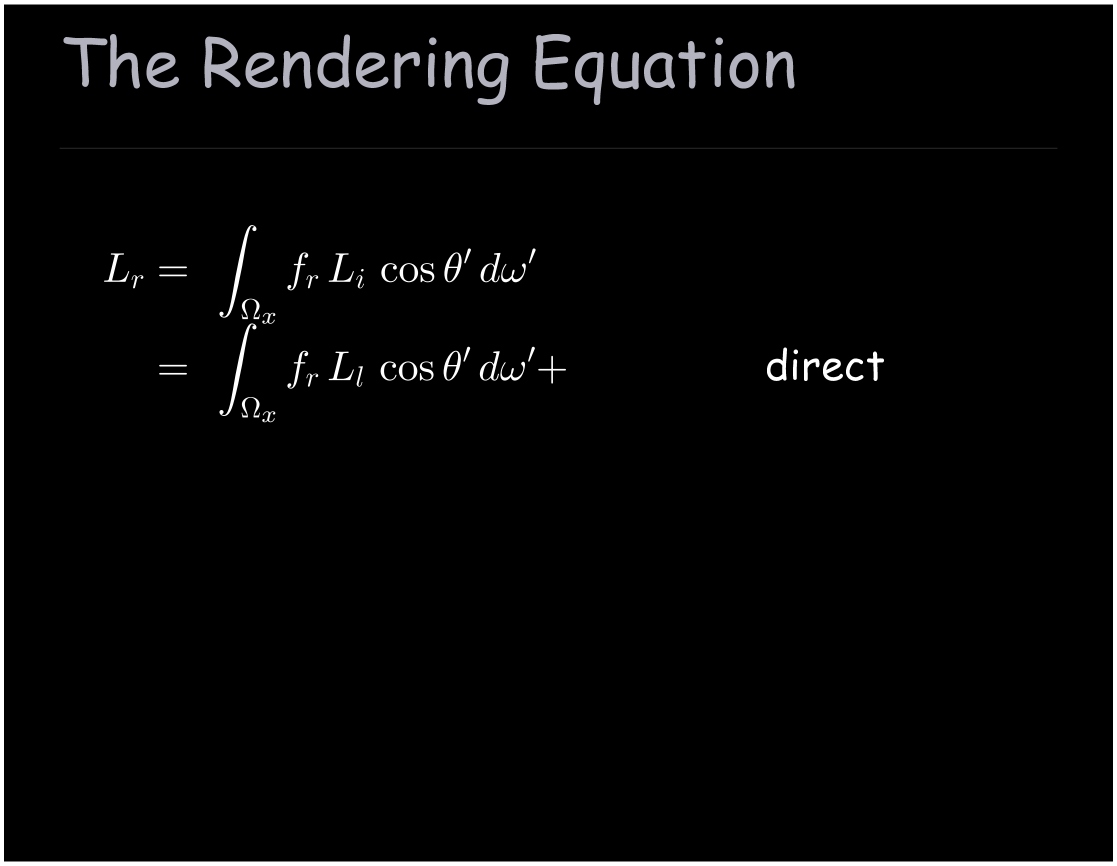
Split the BRDF

$$f_r = \underbrace{f_{r,d}}_{\text{diffuse}} + \underbrace{f_{r,s}}_{\text{specular}}$$



The Rendering Equation

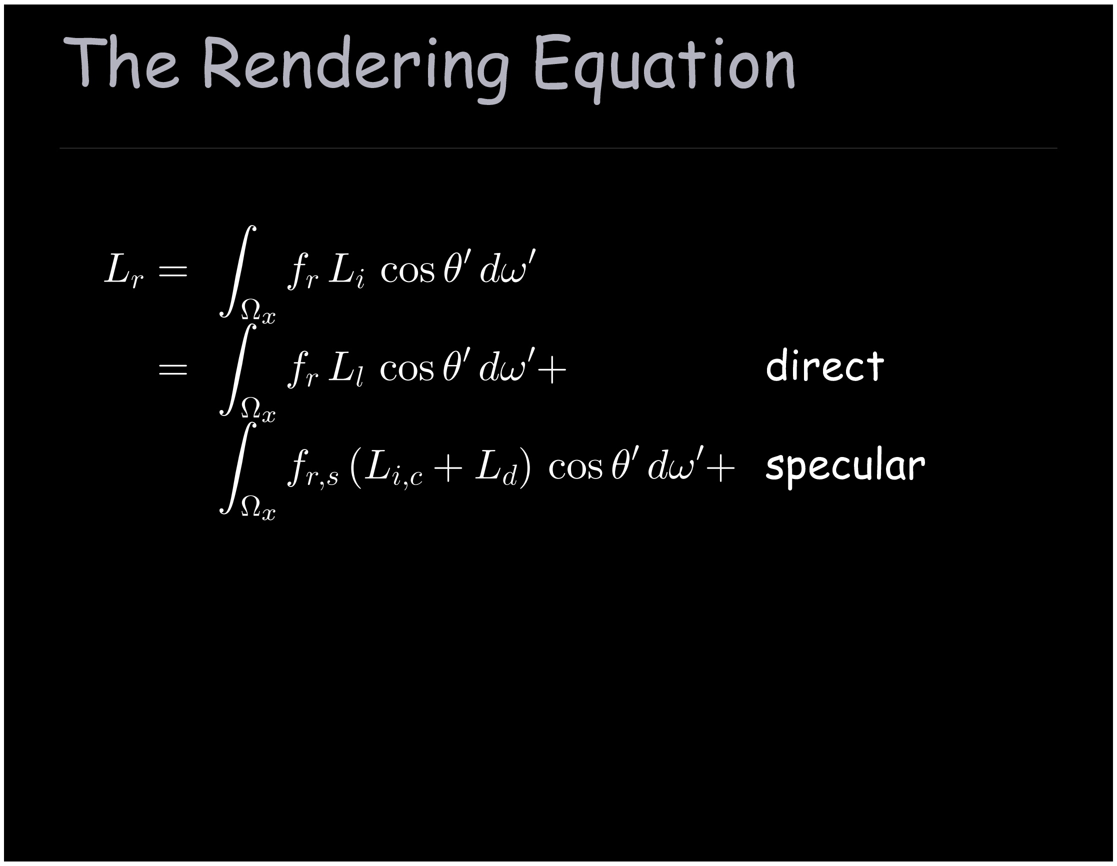
$$L_r = \int_{\Omega_x} f_r \, L_i \, \cos\theta' \, d\omega'$$



The Rendering Equation

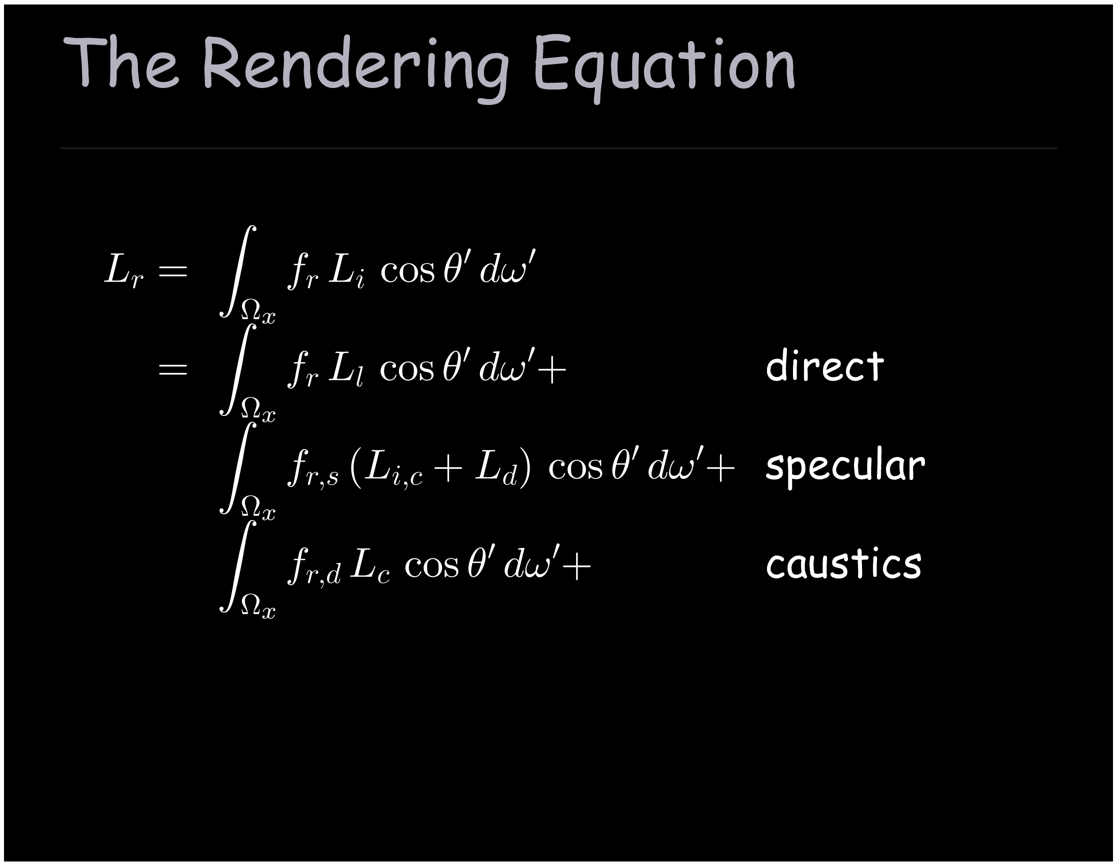
$$\begin{aligned} L_r &= \int_{\Omega_x} f_r\, L_i\, \cos\theta'\, d\omega' \\ &= \int_{\Omega_x} f_r\, L_l\, \cos\theta'\, d\omega' + \end{aligned}$$

direct



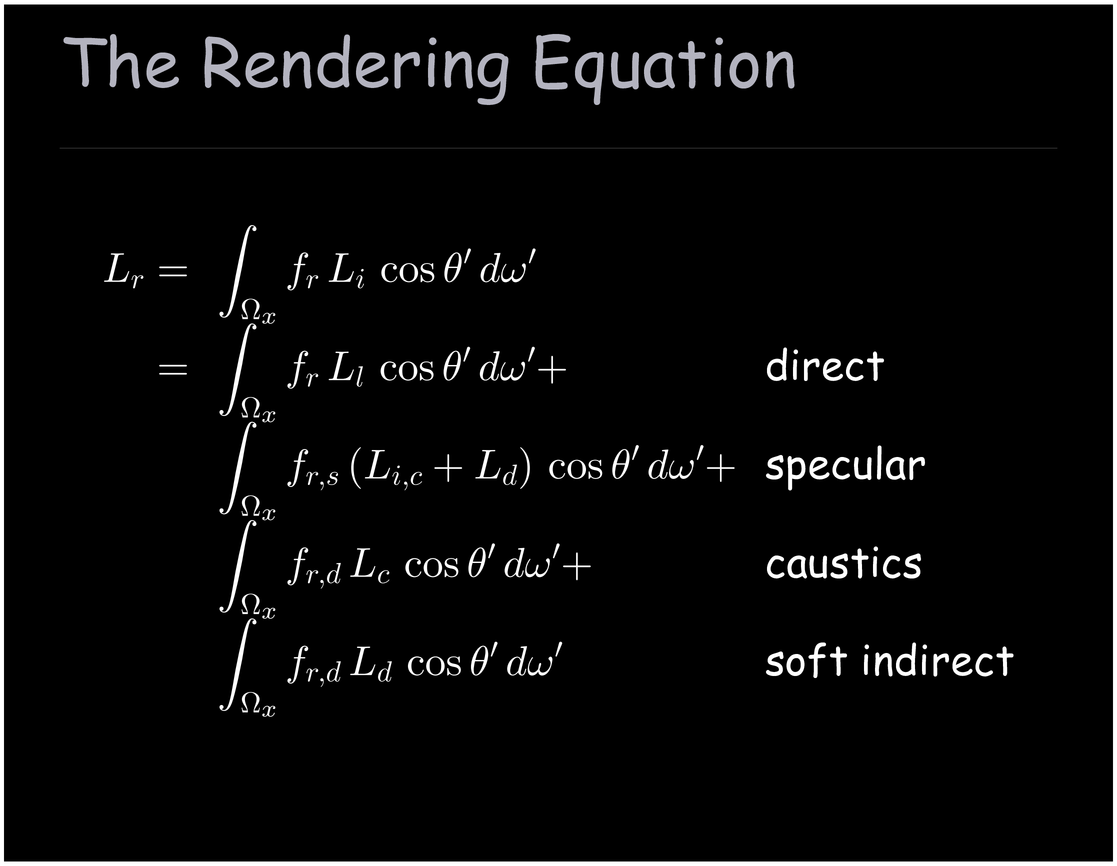
The Rendering Equation

$$\begin{aligned} L_r &= \int_{\Omega_x} f_r \, L_i \, \cos\theta' \, d\omega' \\ &= \int_{\Omega_x} f_r \, L_l \, \cos\theta' \, d\omega' + && \text{direct} \\ &\quad \int_{\Omega_x} f_{r,s} \, (L_{i,c} + L_d) \, \cos\theta' \, d\omega' + && \text{specular} \end{aligned}$$



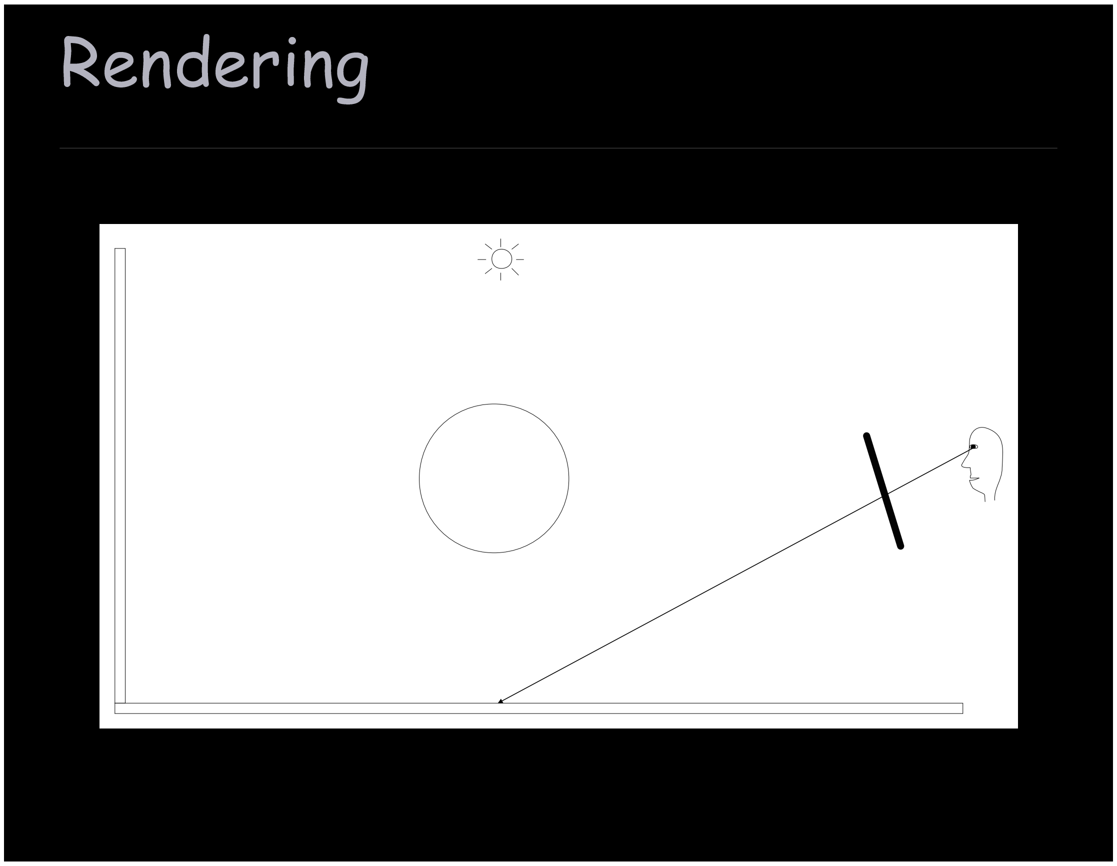
The Rendering Equation

$$
\begin{aligned}
L_r = & \int_{\Omega_x} f_r \, L_i \, \cos\theta' \, d\omega' \\
= & \int_{\Omega_x} f_r \, L_l \, \cos\theta' \, d\omega' + && \text{direct} \\
& \int_{\Omega_x} f_{r,s} \, (L_{i,c} + L_d) \, \cos\theta' \, d\omega' + && \text{specular} \\
& \int_{\Omega_x} f_{r,d} \, L_c \, \cos\theta' \, d\omega' + && \text{caustics}
\end{aligned}
$$

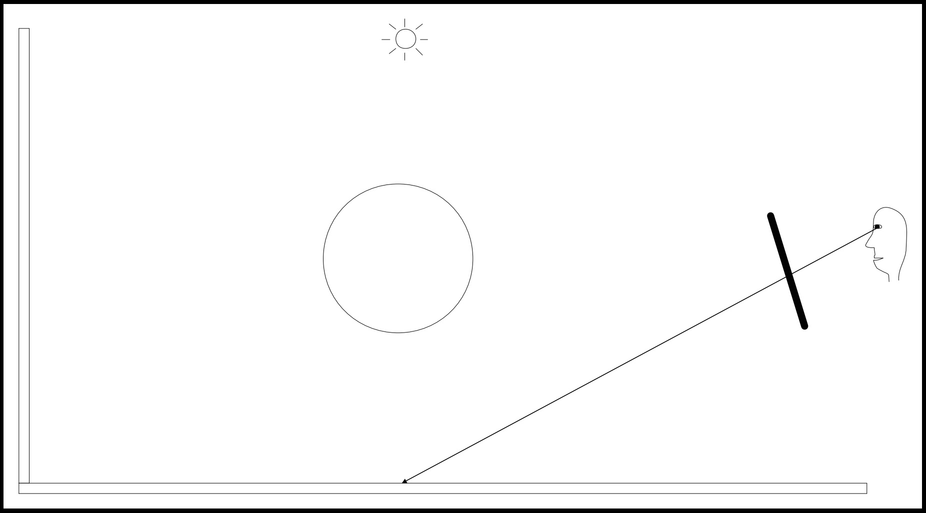


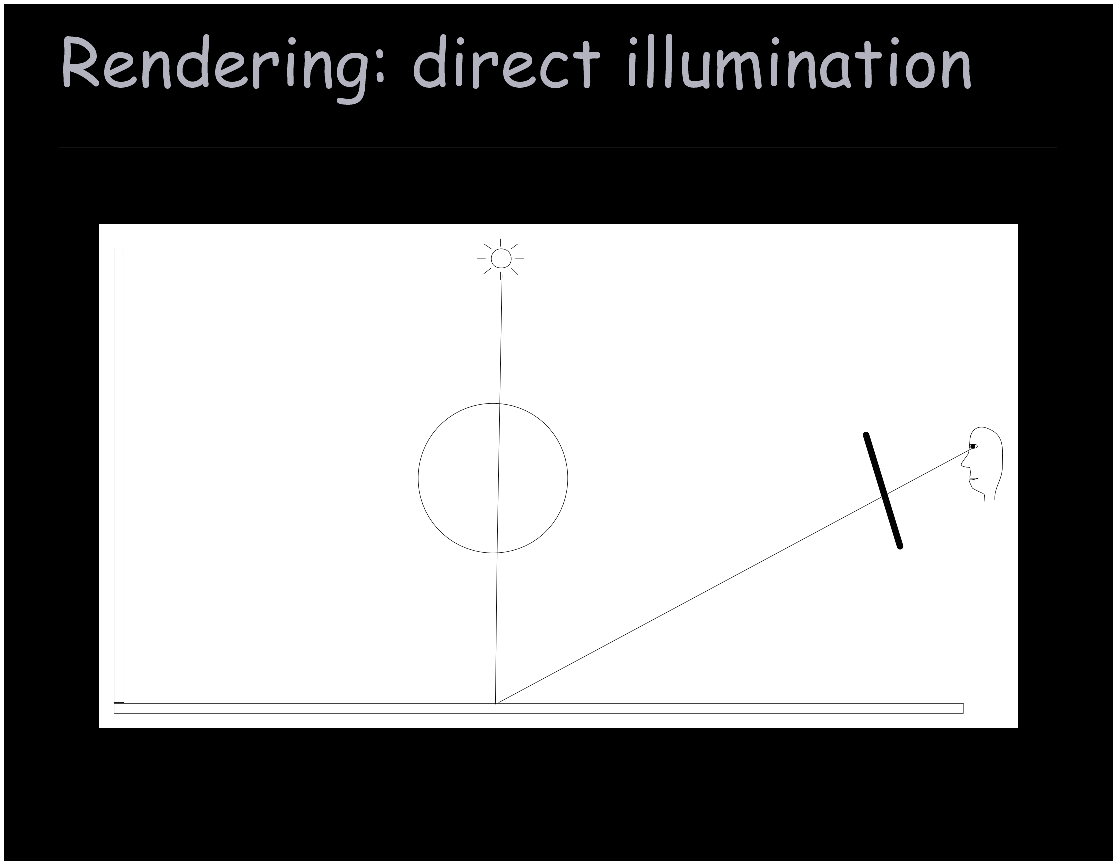
The Rendering Equation

$$
\begin{aligned}
L_r &= \int_{\Omega_x} f_r\, L_i\, \cos\theta'\, d\omega' \\
&= \int_{\Omega_x} f_r\, L_l\, \cos\theta'\, d\omega' + && \text{direct} \\
&\quad \int_{\Omega_x} f_{r,s}\, (L_{i,c} + L_d)\, \cos\theta'\, d\omega' + && \text{specular} \\
&\quad \int_{\Omega_x} f_{r,d}\, L_c\, \cos\theta'\, d\omega' + && \text{caustics} \\
&\quad \int_{\Omega_x} f_{r,d}\, L_d\, \cos\theta'\, d\omega' && \text{soft indirect}
\end{aligned}
$$

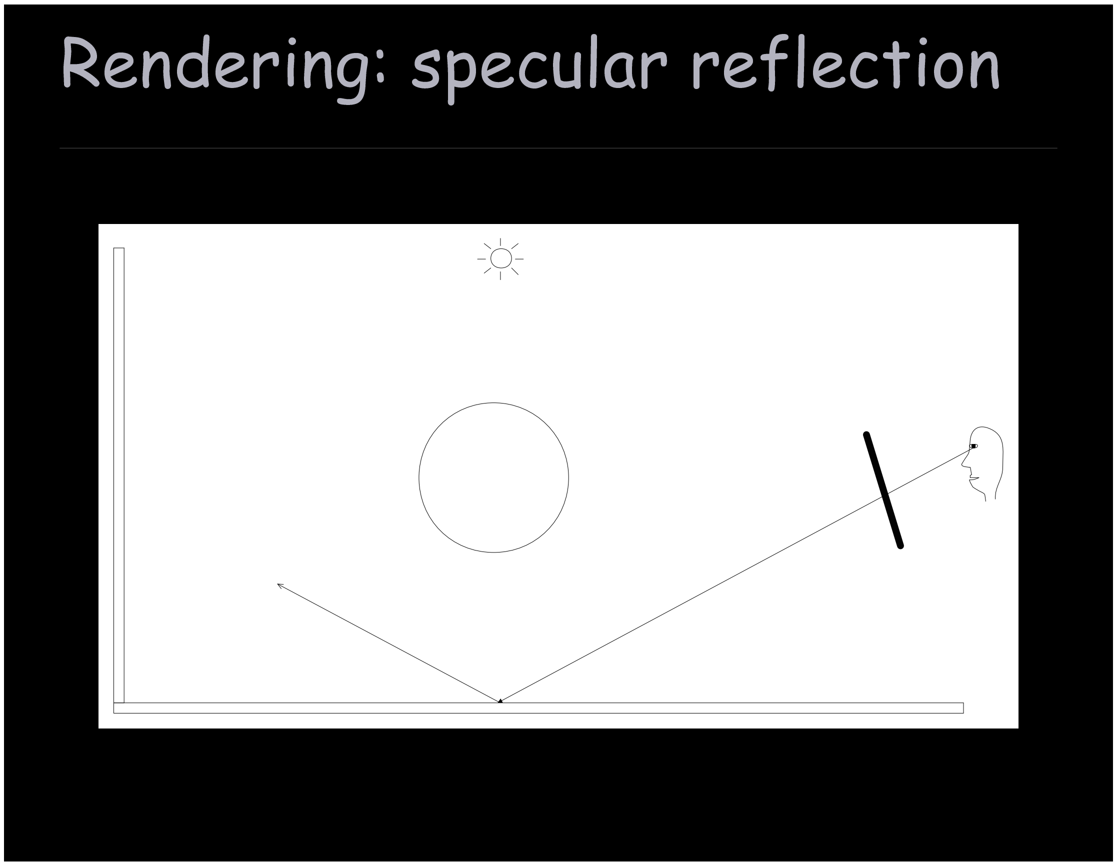


Rendering

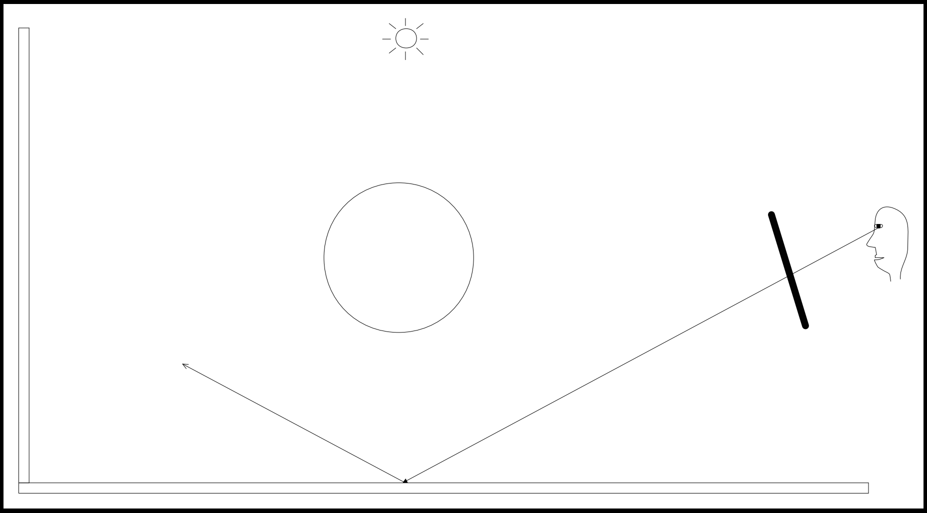


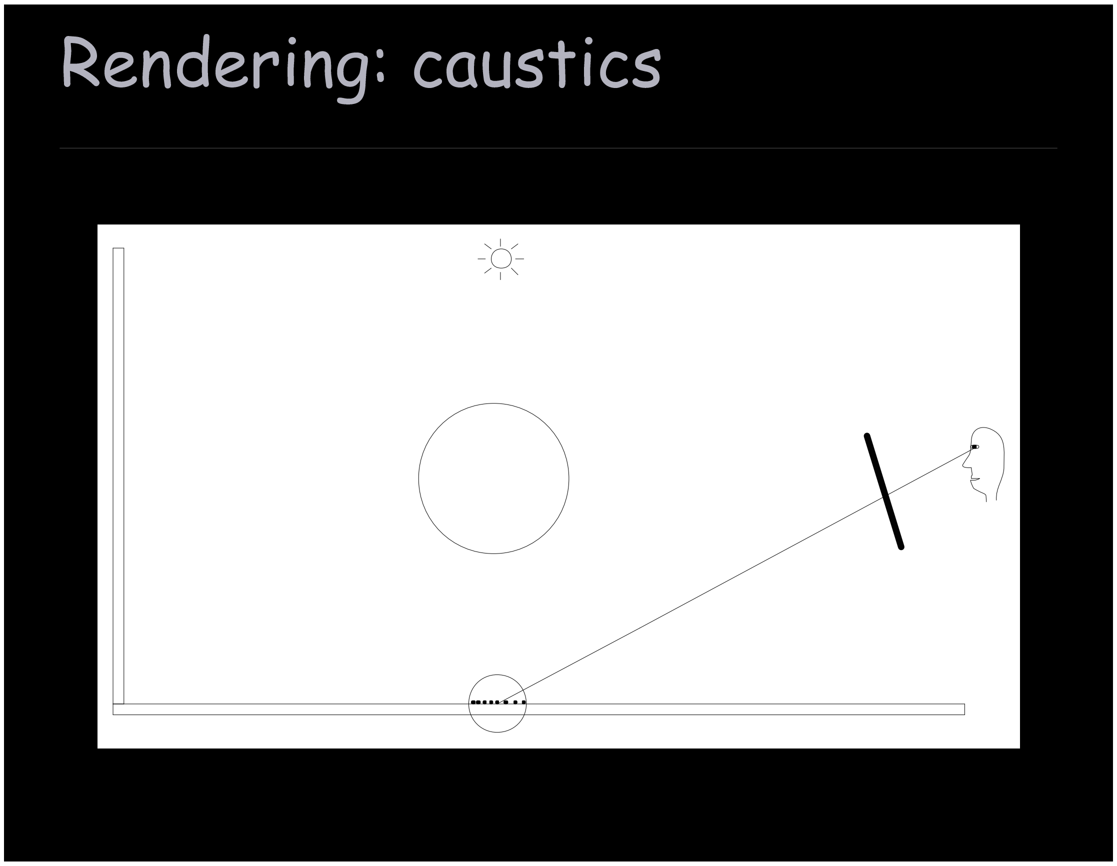


Rendering: direct illumination

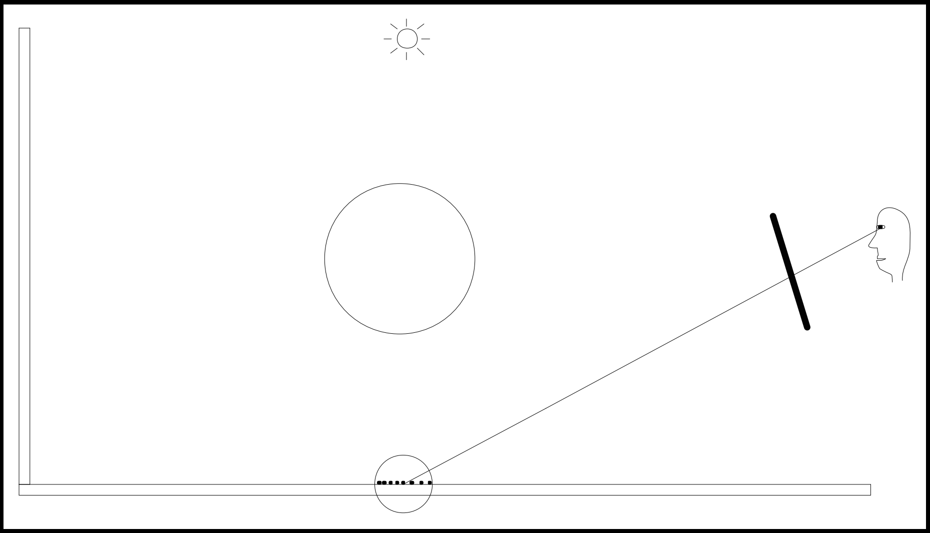


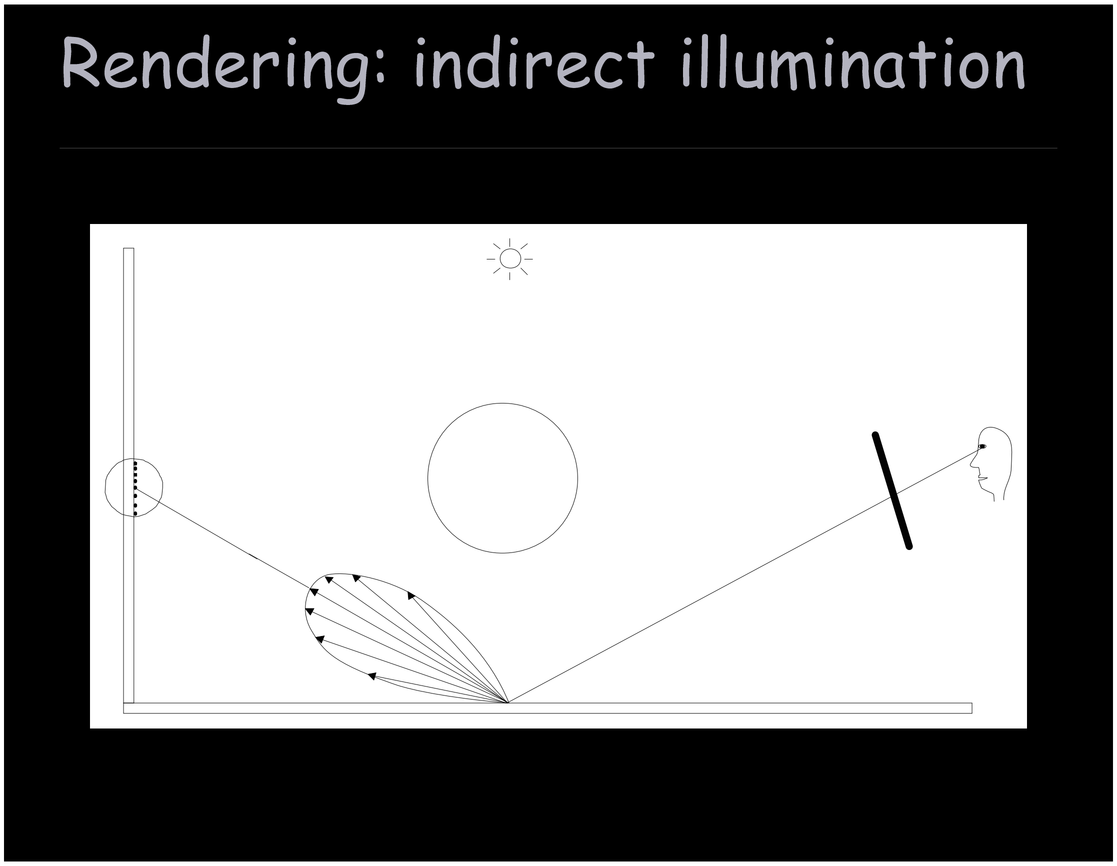
Rendering: specular reflection





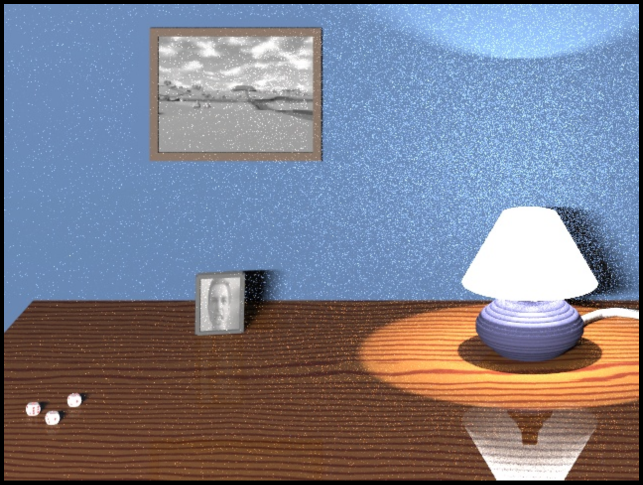
Rendering: caustics





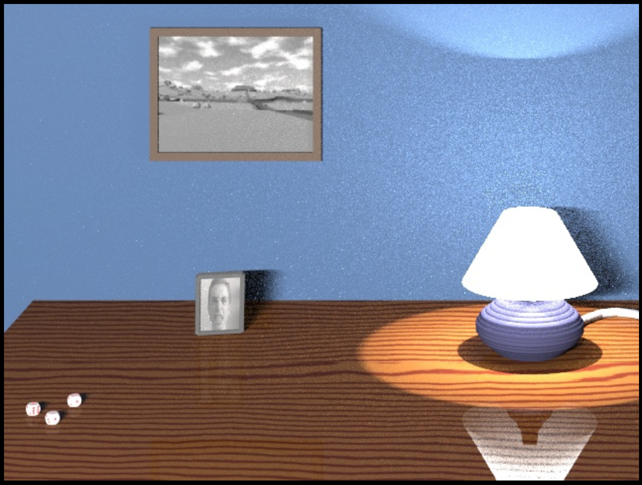
Rendering: indirect illumination

No Importance Sampling



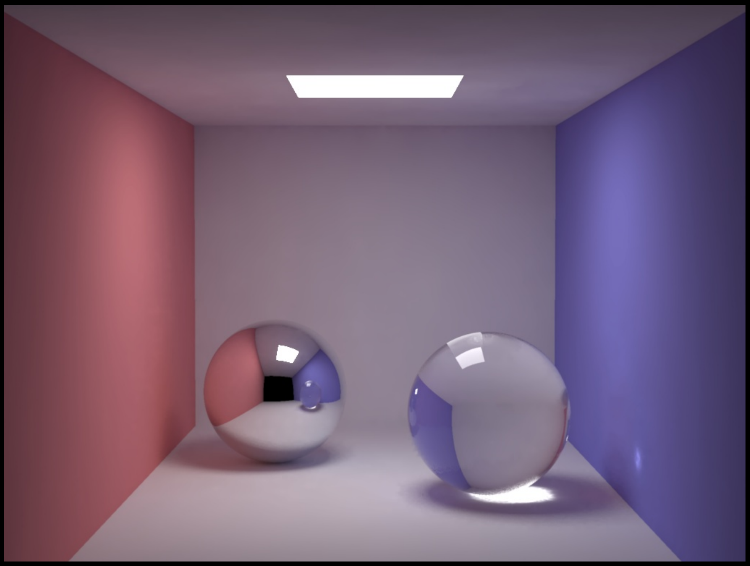
H. Wann Jensen

Importance Sampling



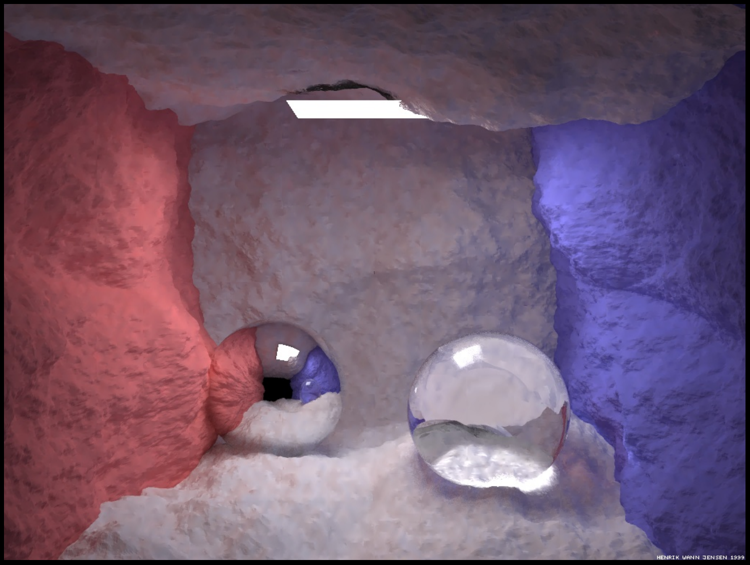
(Using the 50 nearest photons)

Box

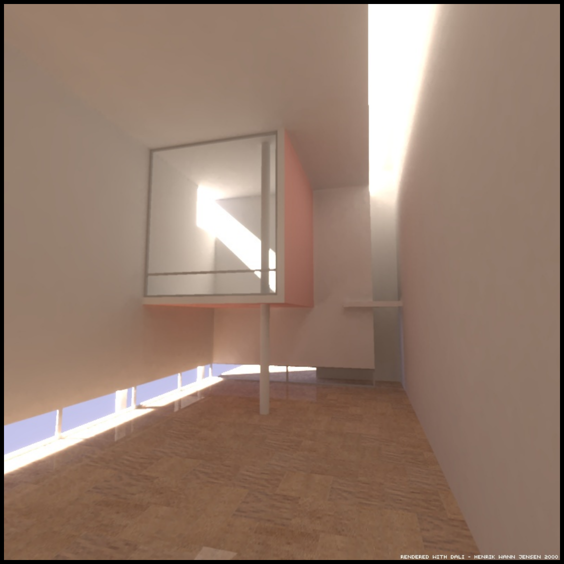


Photon mapping + irradiance caching

Fractal box

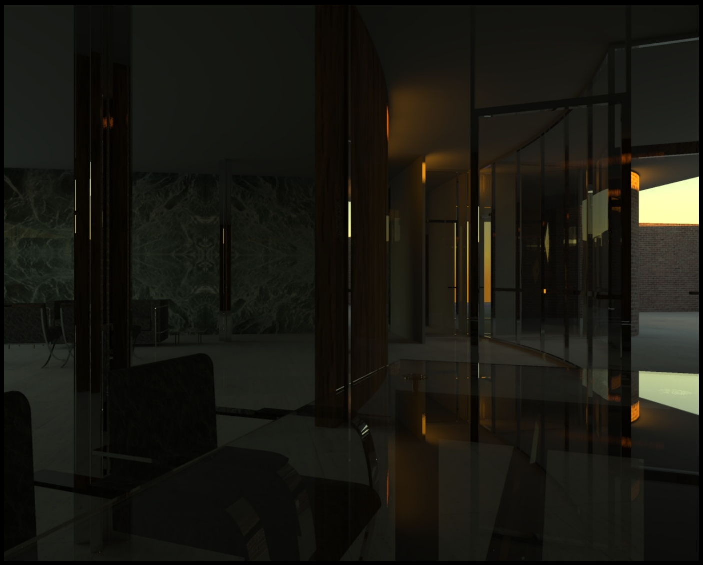


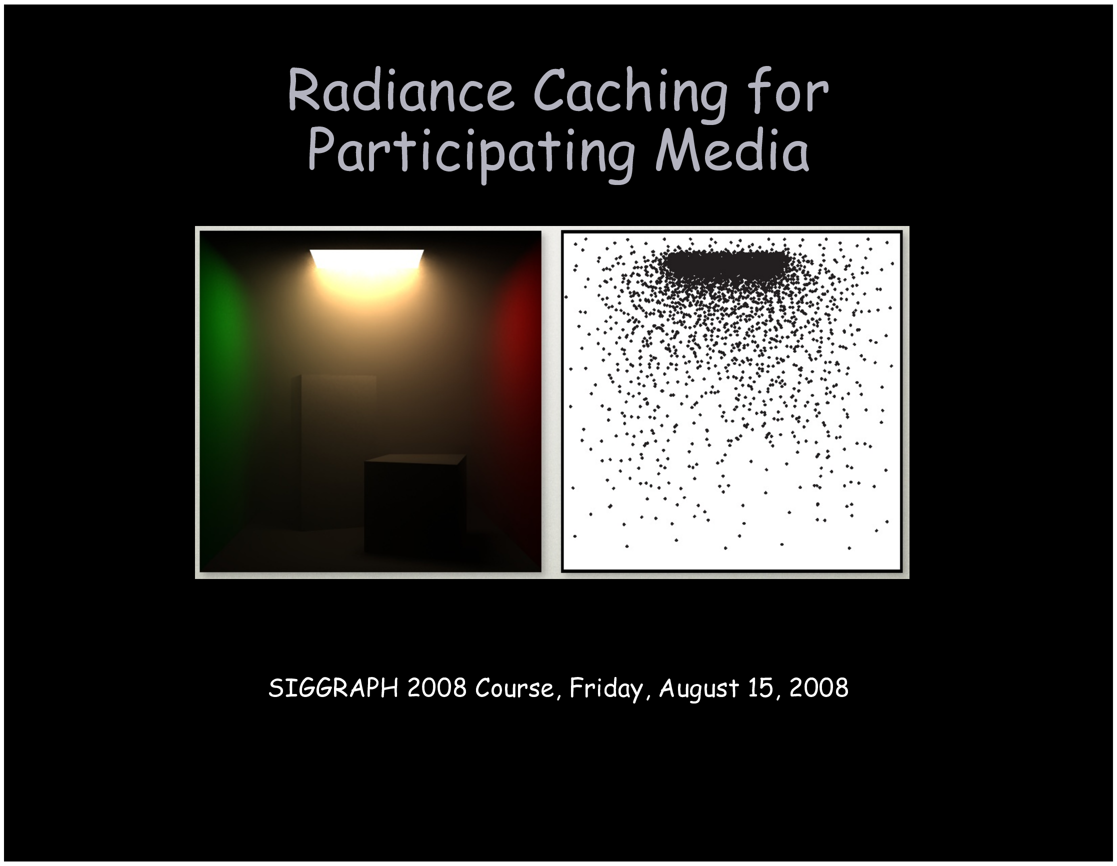
Expo



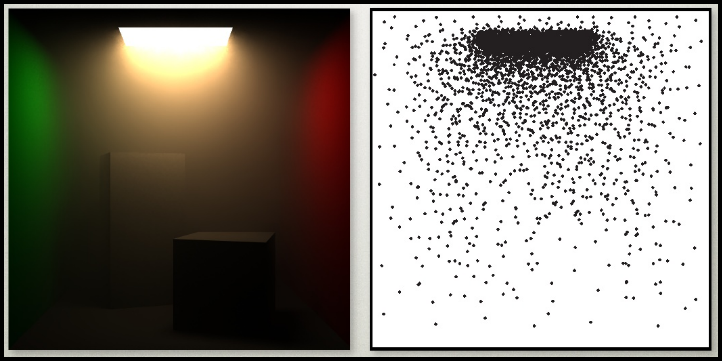
Mies house (2pm)

Mies house (7pm)



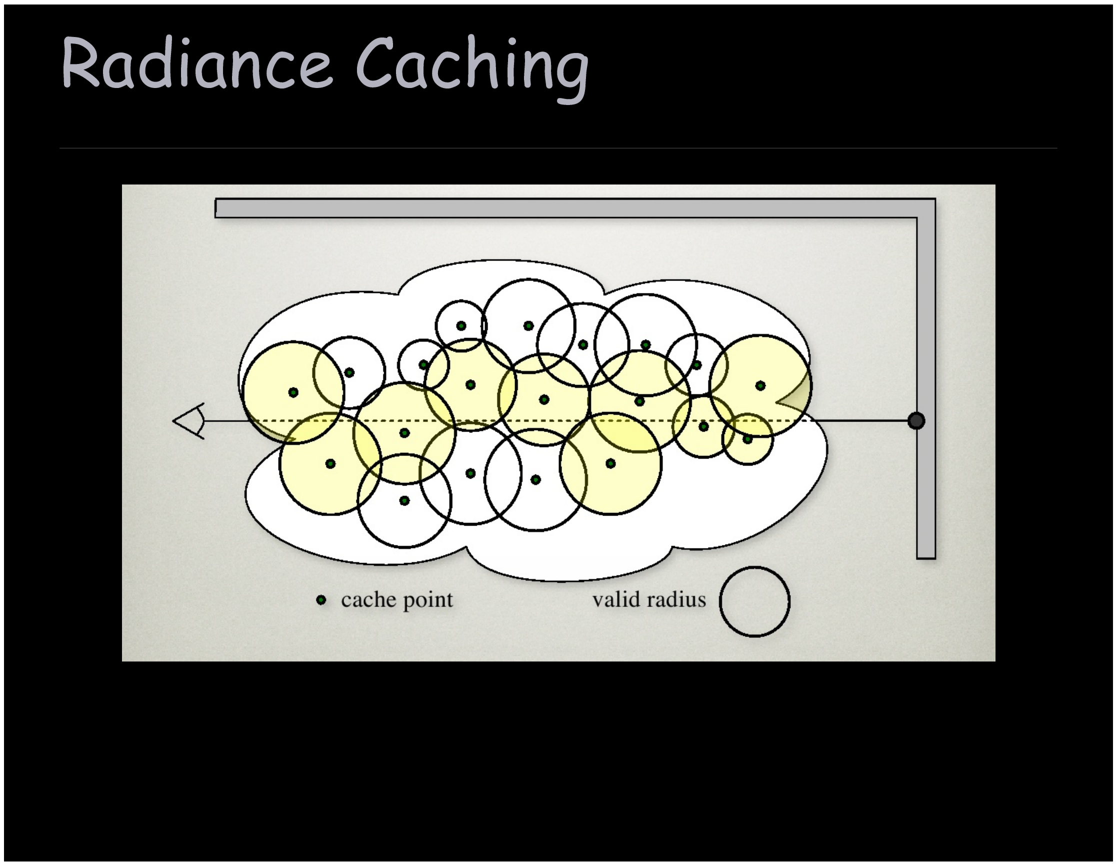


Radiance Caching for Participating Media

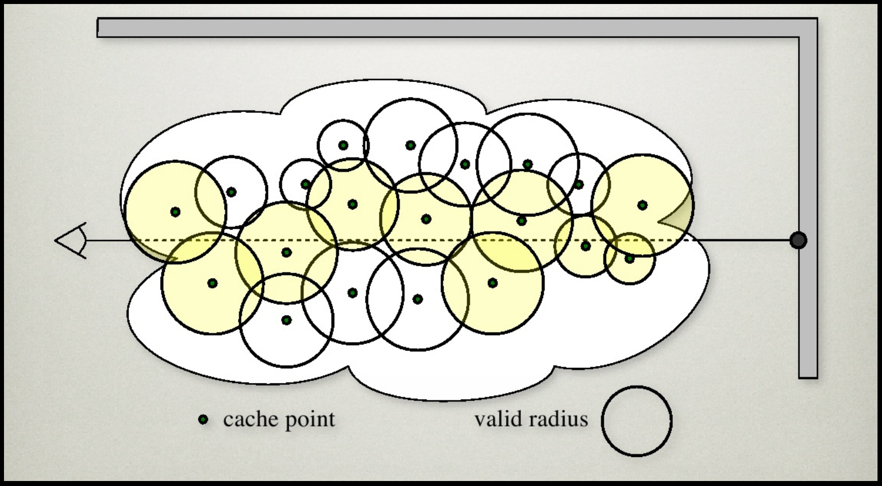


SIGGRAPH 2008 Course, Friday, August 15, 2008

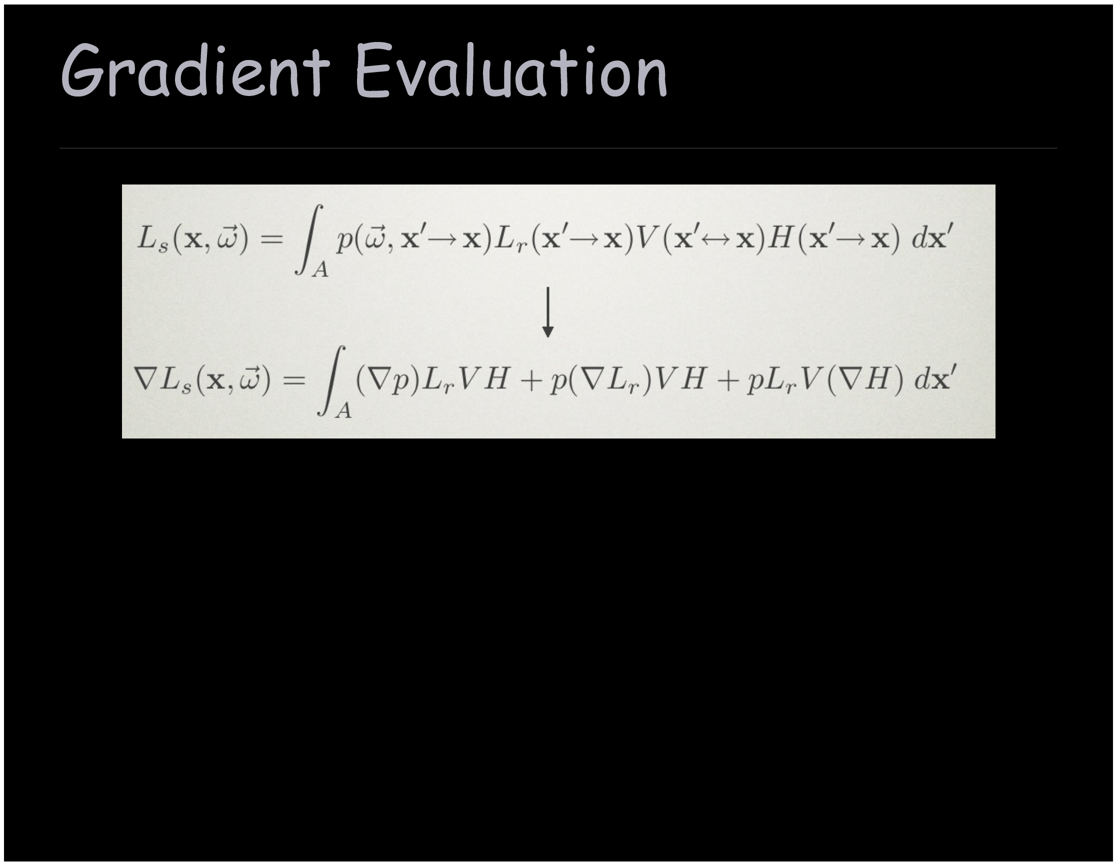
Participating Media



Radiance Caching



H. Wann Jensen



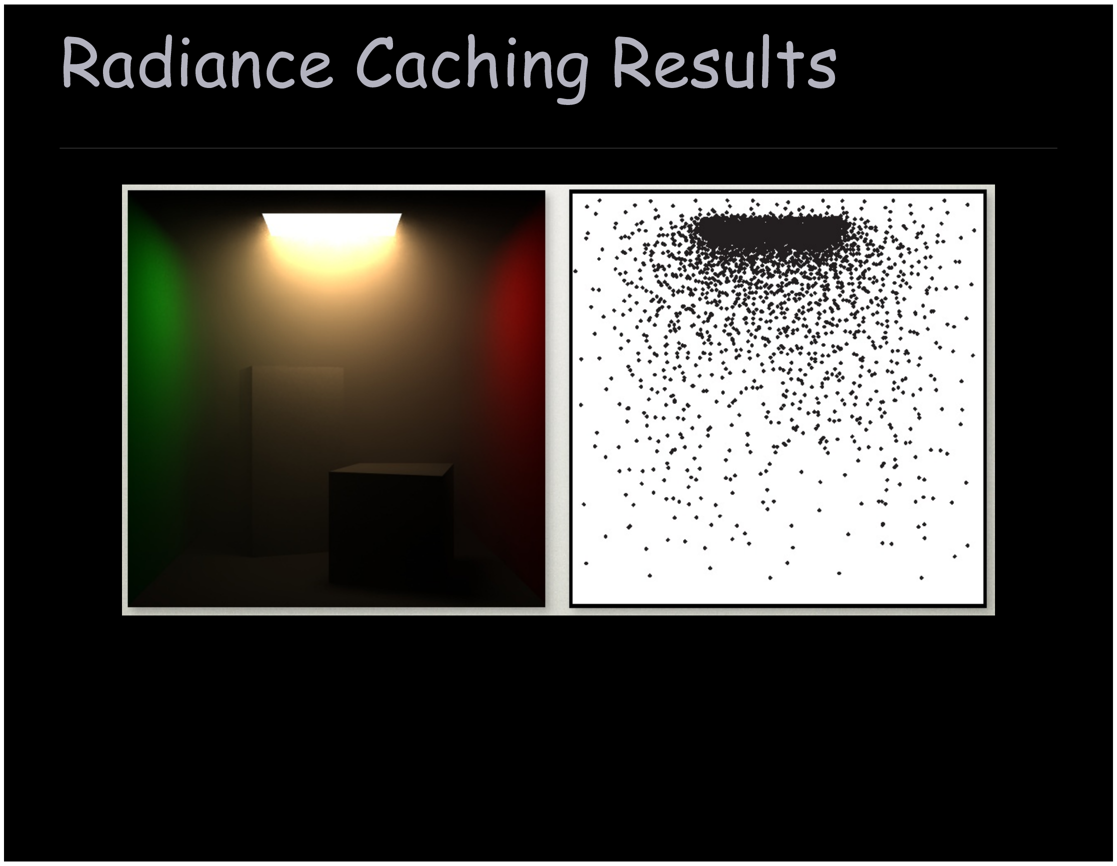
Gradient Evaluation

$$L_s(\mathbf{x},\vec{\omega}) = \int_A p(\vec{\omega},\mathbf{x}'\!\rightarrow\mathbf{x})L_r(\mathbf{x}'\!\rightarrow\mathbf{x})V(\mathbf{x}'\!\leftrightarrow\mathbf{x})H(\mathbf{x}'\!\rightarrow\mathbf{x})\;d\mathbf{x}'$$

$$\downarrow$$

$$\nabla L_s(\mathbf{x},\vec{\omega}) = \int_A (\nabla p)L_rVH + p(\nabla L_r)VH + pL_rV(\nabla H)\;d\mathbf{x}'$$

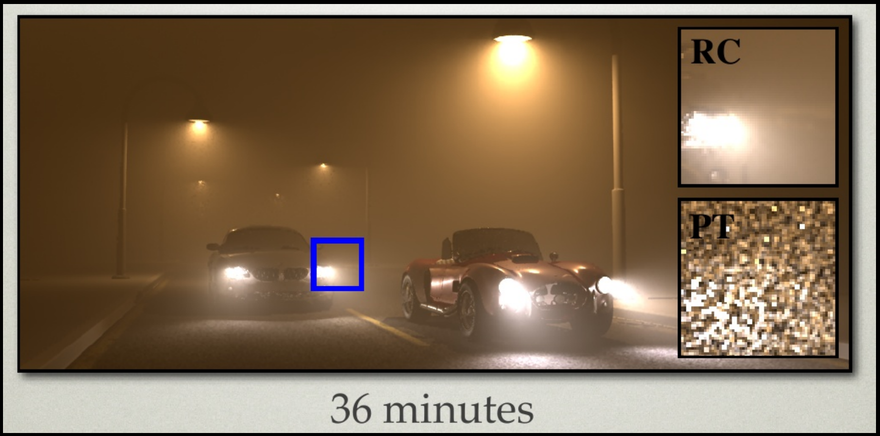
H. Wann Jensen

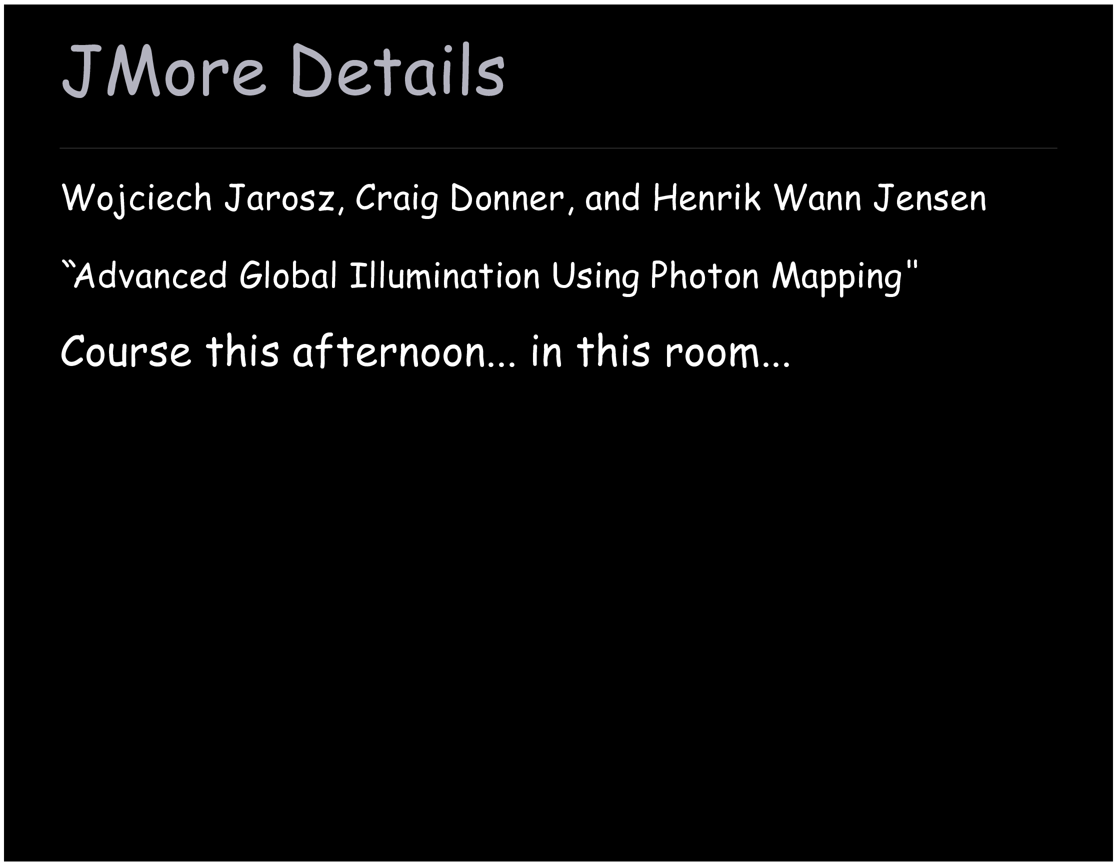


Radiance Caching Results

H. Wann Jensen

Cars in Fog





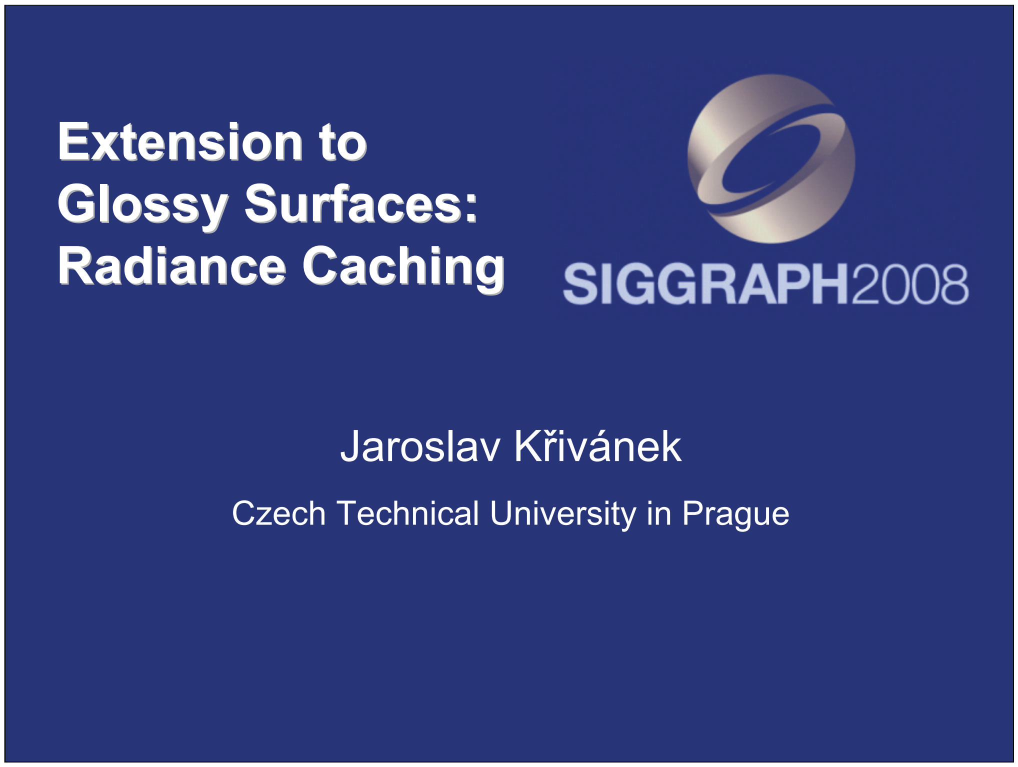
JMore Details

Wojciech Jarosz, Craig Donner, and Henrik Wann Jensen

"Advanced Global Illumination Using Photon Mapping"

Course this afternoon... in this room...

SIGGRAPH2008



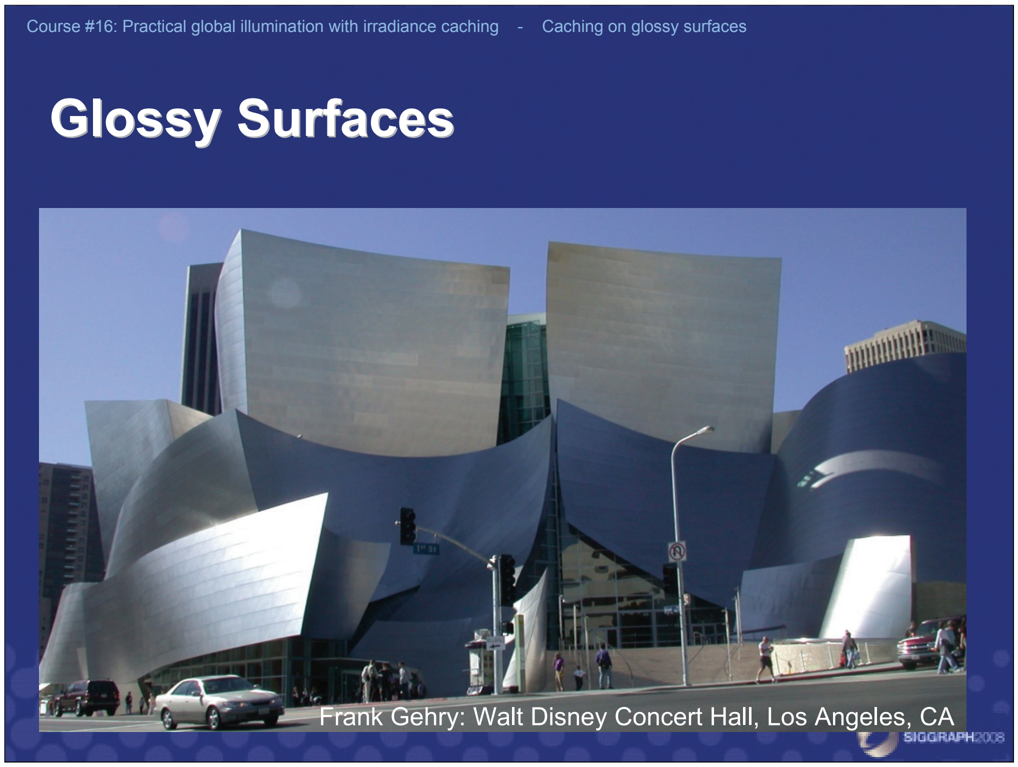
Extension to Glossy Surfaces: Radiance Caching

SIGGRAPH2008

Jaroslav Křivánek

Czech Technical University in Prague

The interpolation scheme used in irradiance caching for diffuse surfaces can be also used on glossy surfaces. However, some modifications are necessary due to the view-dependence of reflection on glossy surfaces. These modifications will be discussed in this part of the course.

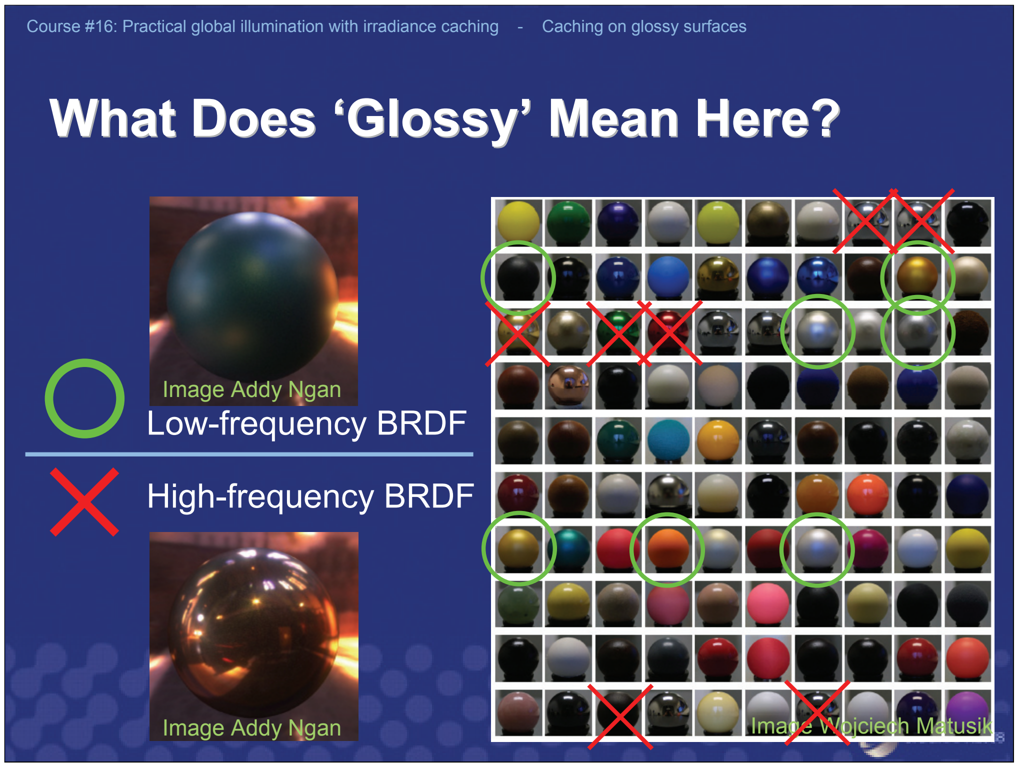


Glossy Surfaces

Frank Gehry: Walt Disney Concert Hall, Los Angeles, CA

SIGGRAPH2008

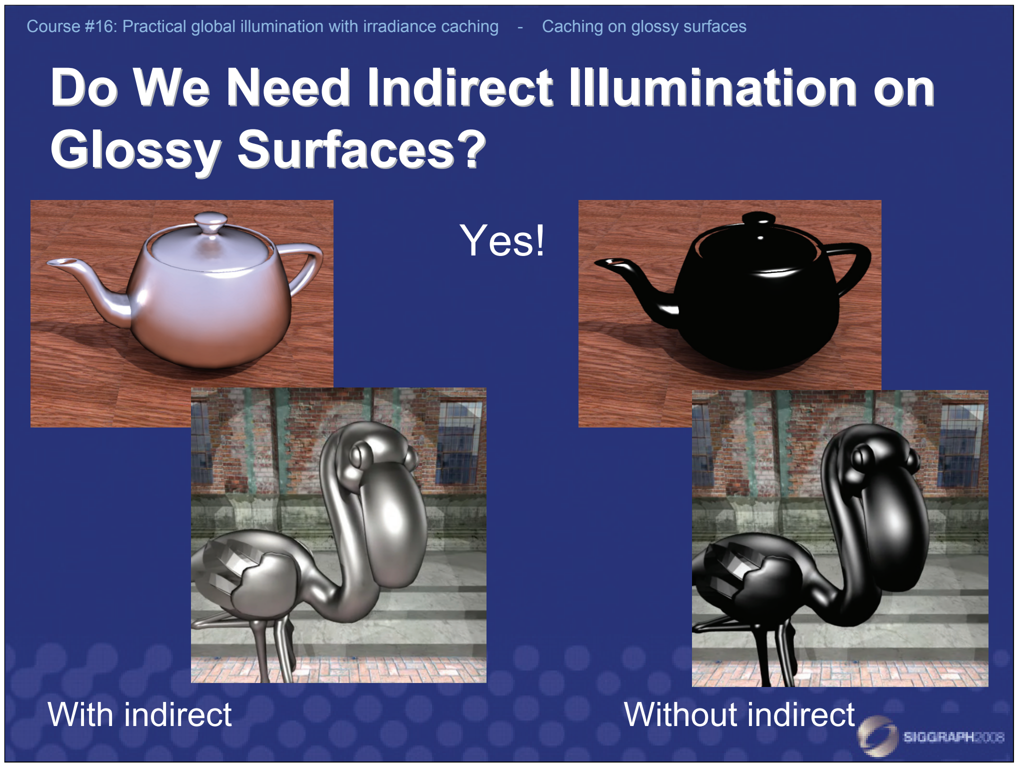
A nice real-world example of glossy surfaces are the walls of Frank Gehry's Walt Disney Concert Hall in Los Angeles (as well as Gehry's Guggenheim Museum in Bilbao, Spain). Other common examples of glossy surfaces include rough plastic surfaces or rough pottery.



Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

What Does 'Glossy' Mean Here?

What do we exactly mean by 'glossy surface' for the purpose of this talk? We are referring to rough surfaces that reflect their environment but the reflections are blurry. This is in contrast to 'specular' surfaces that show sharp reflections of their environment. Glossy surfaces have 'low-frequency' BRDFs while specular surfaces have 'high-frequency' BRDFs. Indirect illumination on specular surfaces can be quite efficiently computed by stochastic ray tracing with importance sampling (proportional to the BRDF) using only a couple of secondary rays. In what follows, we will focus on computing indirect illumination on glossy surfaces (with low-frequency BRDFs), for which importance sampling is not very effective.



Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

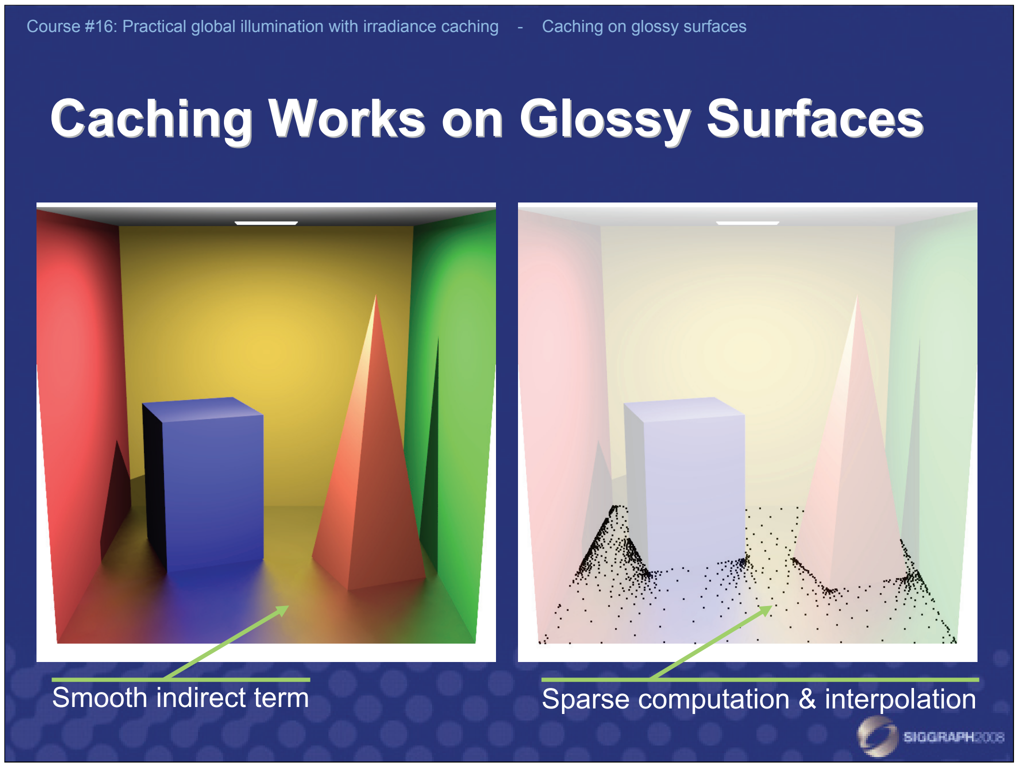
Do We Need Indirect Illumination on Glossy Surfaces?

Yes!

With indirect

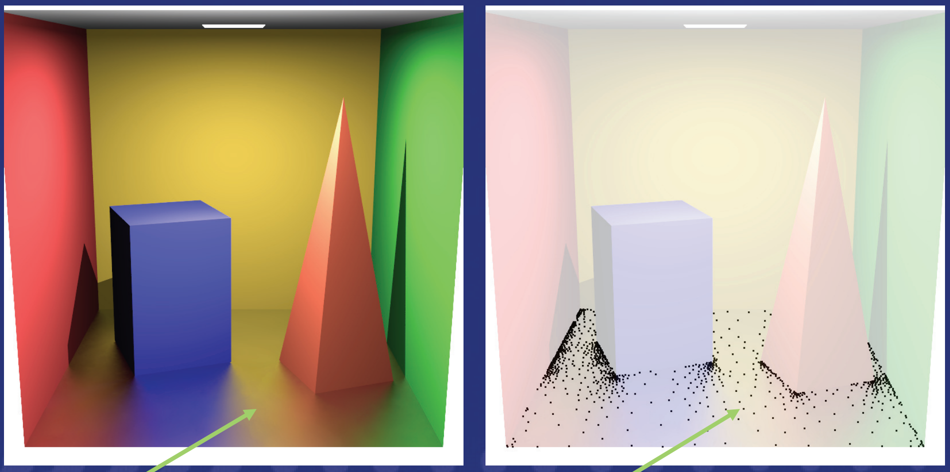
Without indirect

Why do we bother computing indirect illumination on glossy surfaces? Is it worth the effort? Yes, it certainly is. Actually, Fleming et al. [2003] have shown that correct perception of material properties depends very much on the illumination. Under a point light, we don't really know what an object is made of, while under natural illumination, it is much easier to recognize the object's material.



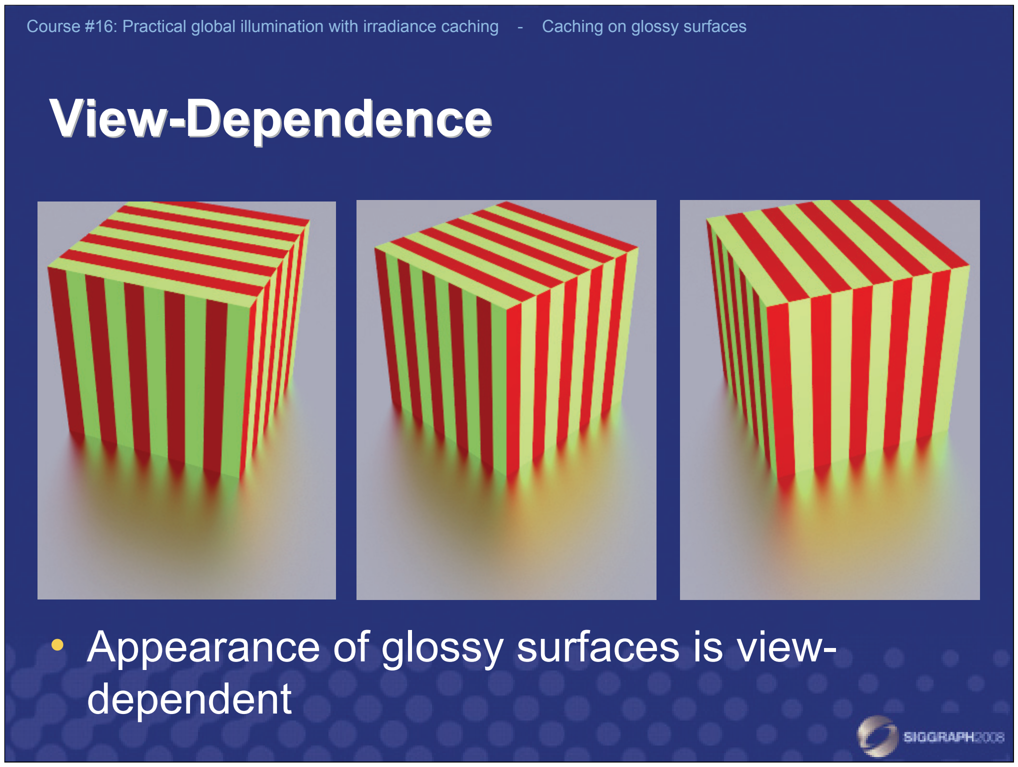
Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Caching Works on Glossy Surfaces



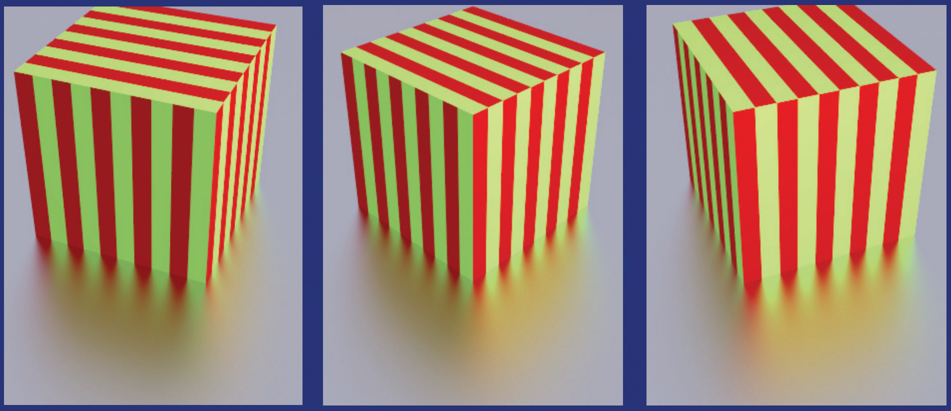
SIGGRAPH2008

Fortunately, glossy surfaces show a very blurry reflections of their environment. In other words, indirect illumination on glossy surfaces changes slowly, similar to diffuse ones. This allows to use the same trick as in irradiance caching – compute illumination lazily at sparse locations in the scene and interpolate elsewhere.



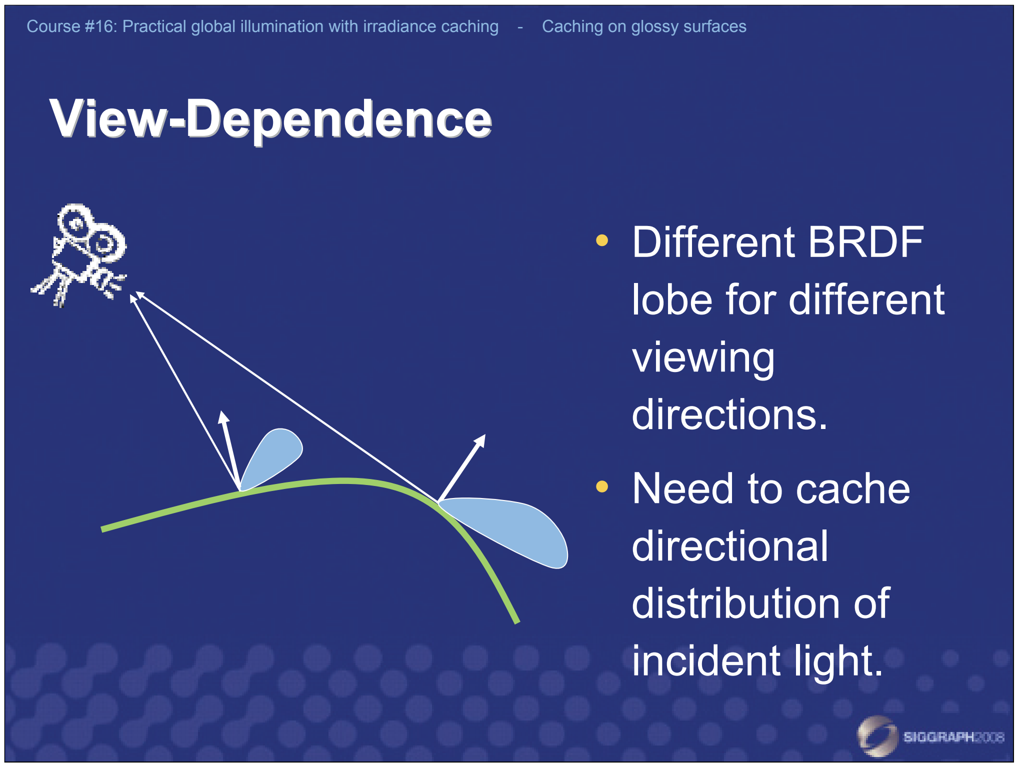
Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

View-Dependence



- Appearance of glossy surfaces is view-dependent

Unlike for diffuse surfaces, appearance of glossy surfaces is view-dependent. In other words, the surface looks different when observed from different viewing angles. This makes interpolation of illumination more complicated than on purely diffuse surfaces.

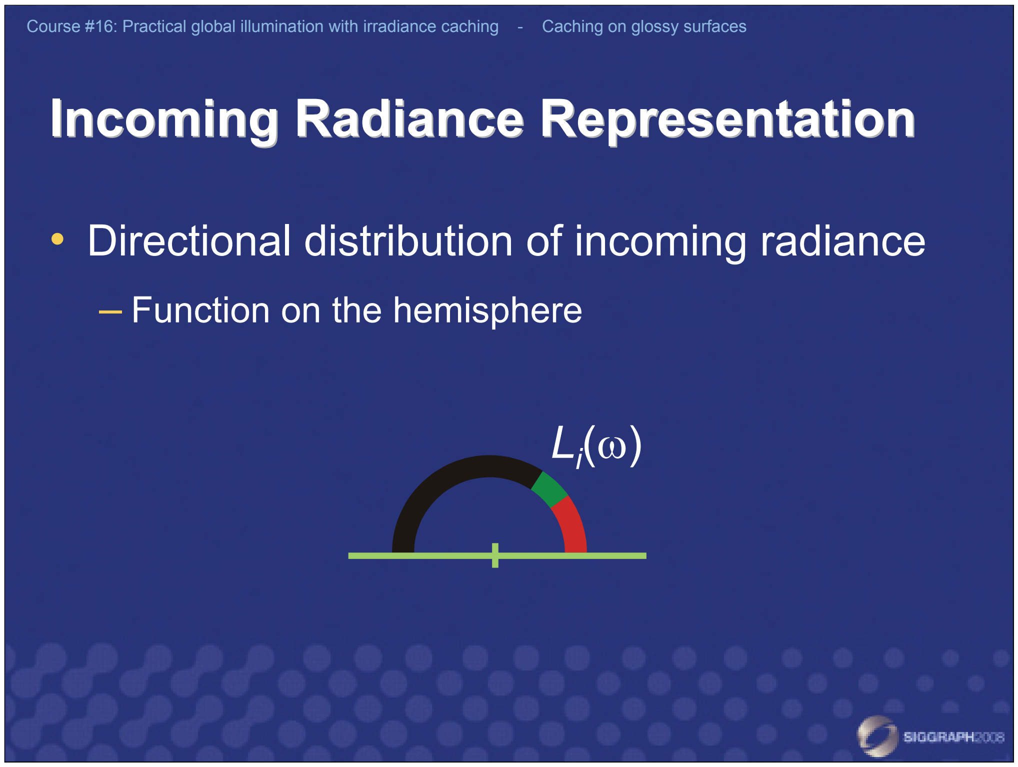


Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

View-Dependence

- Different BRDF lobe for different viewing directions.
- Need to cache directional distribution of incident light.

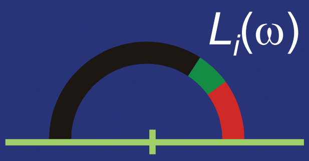
The view-dependence of glossy surfaces is due to the fact that different part of incident light is reflected towards the camera for different viewing directions. So if we want to interpolate illumination on glossy surfaces, we need to cache the full directional distribution of incoming radiance – i.e. we cache a representation of incoming radiance at a point for all possible directions on the hemisphere. Then, at each interpolation point, we ‘extract’ the part of the incoming light which is reflected towards the camera. This ‘extraction’ is done by evaluating the illumination integral, i.e. by integrating incoming light multiplied by the BRDF lobe over the hemisphere. This will be discussed in more detail later.



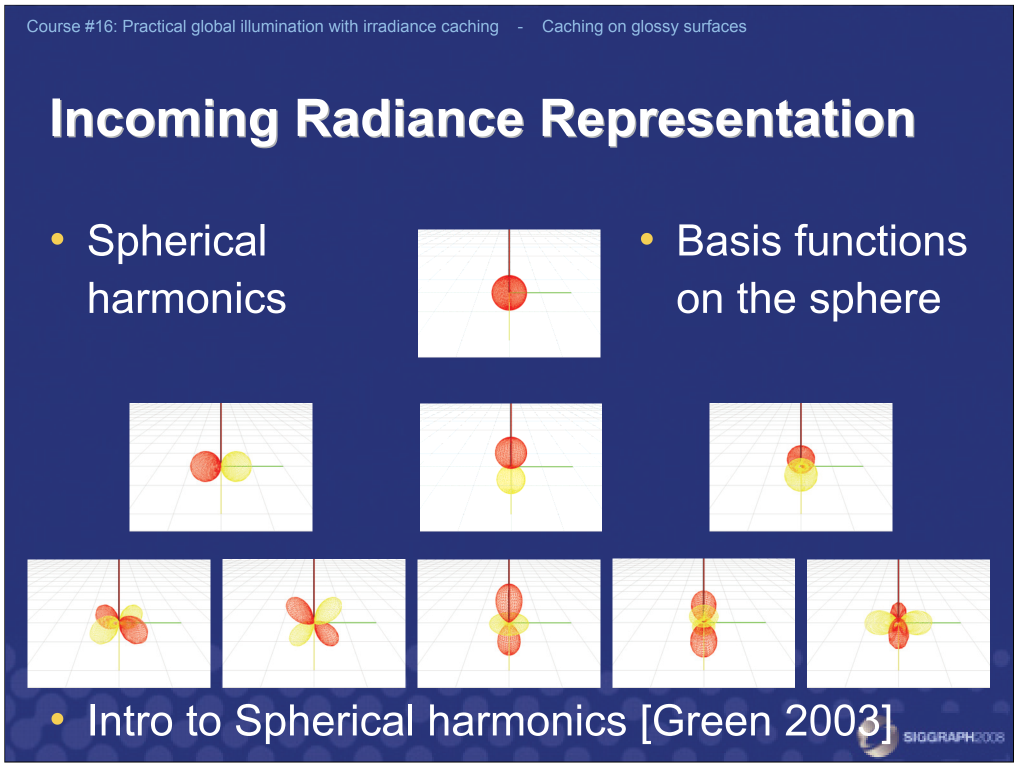
Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Incoming Radiance Representation

- Directional distribution of incoming radiance
  - Function on the hemisphere

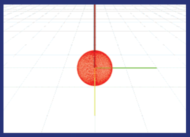


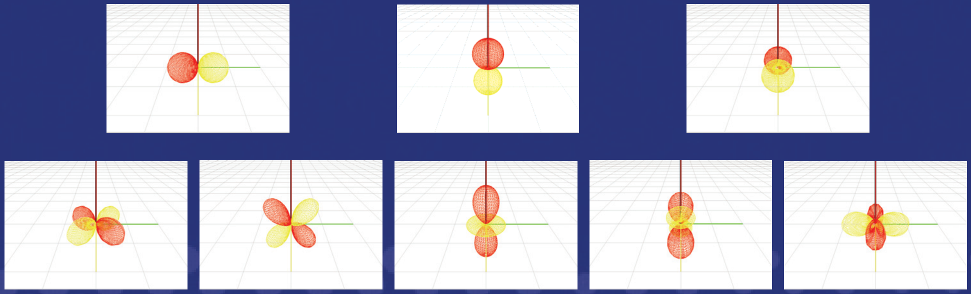
The cached quantity – directional distribution of incoming radiance - is a function defined on the hemisphere. We need to find a suitable representation of this function to make caching possible.



Incoming Radiance Representation

- Spherical harmonics
- Basis functions on the sphere

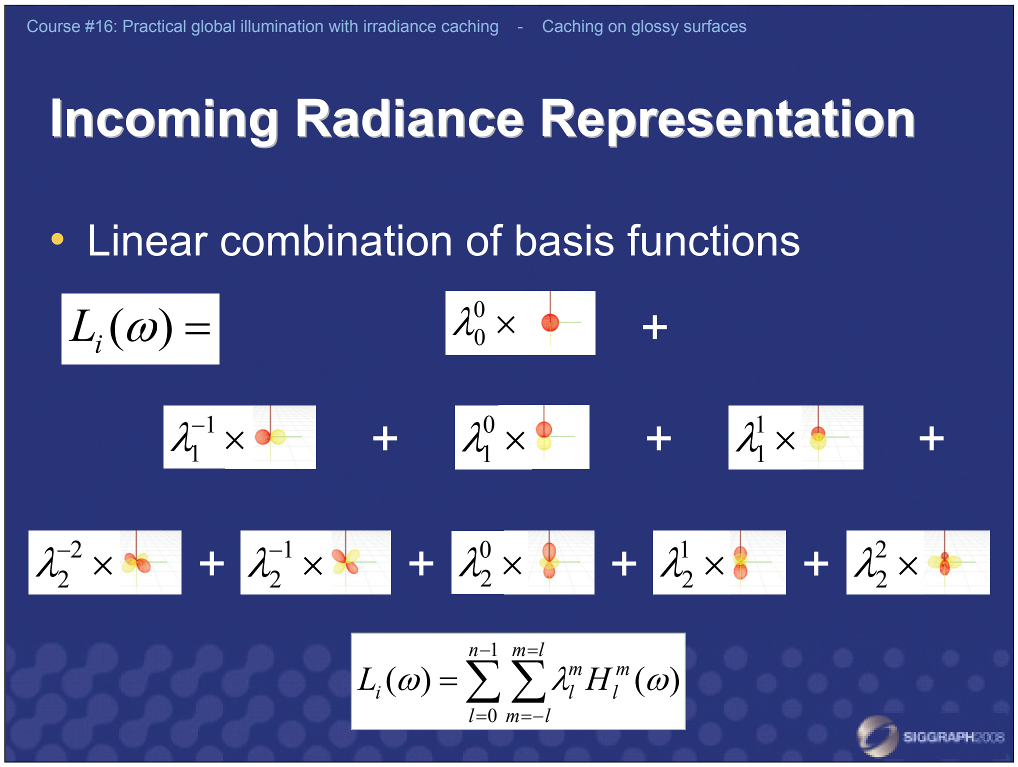




- Intro to Spherical harmonics [Green 2003]

SIGGRAPH2008

For various reasons that will become clearer later, we use spherical harmonics to represent the incoming radiance at a point. Spherical harmonics are basis functions defined on the unit sphere.


Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Incoming Radiance Representation

- Linear combination of basis functions

$L_i(\omega) =$ $\lambda_0^0 \times$ +

$\lambda_1^{-1} \times$ + $\lambda_1^0 \times$ + $\lambda_1^1 \times$ +

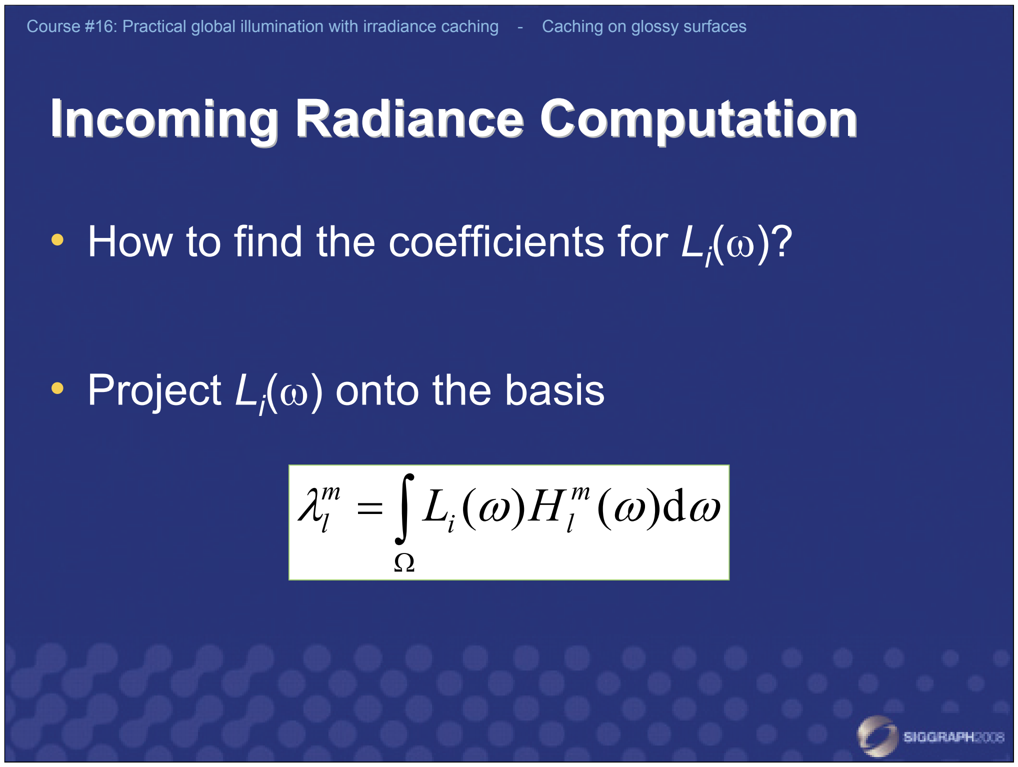
$\lambda_2^{-2} \times$ + $\lambda_2^{-1} \times$ + $\lambda_2^0 \times$ + $\lambda_2^1 \times$ + $\lambda_2^2 \times$

$$L_i(\omega) = \sum_{l=0}^{n-1} \sum_{m=-l}^{m=l} \lambda_l^m H_l^m(\omega)$$

Incoming radiance, as well as any other function defined on the hemisphere, can be represented as a linear combination of the basis functions. The coefficients in this linear combination make up the representation of our function.

For spherical harmonics, the coefficients are indexed by two indices, *l* (lower index) and *m* (upper index). The *l*-index runs from 0 to *n*-1, where *n* is the order of the spherical harmonics representation. The higher the order, the better (more accurate) representation. Spherical harmonics with the same *l*-index form a band (a row on the slide above). Within one band, the *m* index goes from –*l* to +*l*. So there is one harmonic in the first band, three harmonics in the second band, five in the third band etc. There are n^2 coefficients in total for an order-*n* representation. The double sum in the formula on the slide simply stands for summing over all harmonics up to order *n*. It could also be expressed as a single sum for *i* from 0 to n^2-1, with *i* given by : $i = l(l+1) + m$.

To summarize, an order-*n* representation of a function consists of n^2 coefficients, which is what we to store in the cache. (Actually, we store one coefficient vector for each color component, R, G, and B). Typically, we use order up to 10 in radiance caching, corresponding to 100 coefficients.



Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

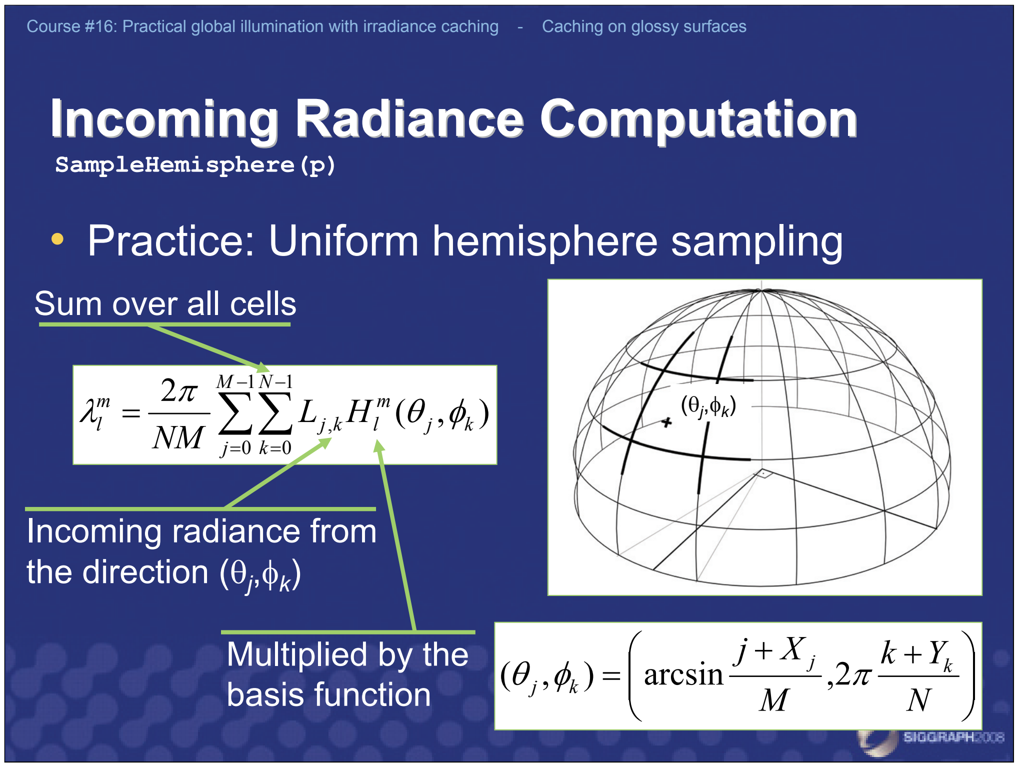
Incoming Radiance Computation

- How to find the coefficients for $L_i(\omega)$?
- Project $L_i(\omega)$ onto the basis

$$\lambda_l^m = \int_\Omega L_i(\omega) H_l^m(\omega) \mathrm{d}\omega$$

SIGGRAPH2008

The coefficients representing a function are found by *projection*. For our purposes, it is sufficient to say that the projection is computed by integrating the product of the function being projected with the basis function, as shown on the slide.



Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Incoming Radiance Computation

`SampleHemisphere(p)`

- Practice: Uniform hemisphere sampling

Sum over all cells

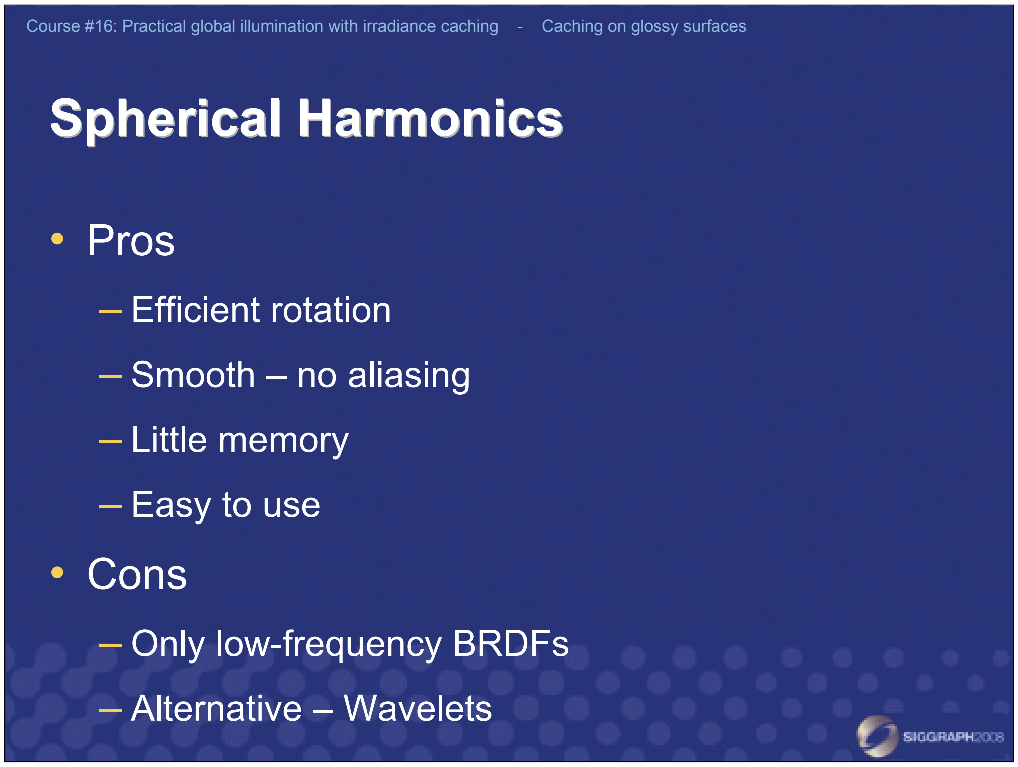
$$\lambda_l^m = \frac{2\pi}{NM}\sum_{j=0}^{M-1}\sum_{k=0}^{N-1} L_{j,k} H_l^m(\theta_j,\phi_k)$$

Incoming radiance from the direction (θ_j,ϕ_k)

Multiplied by the basis function

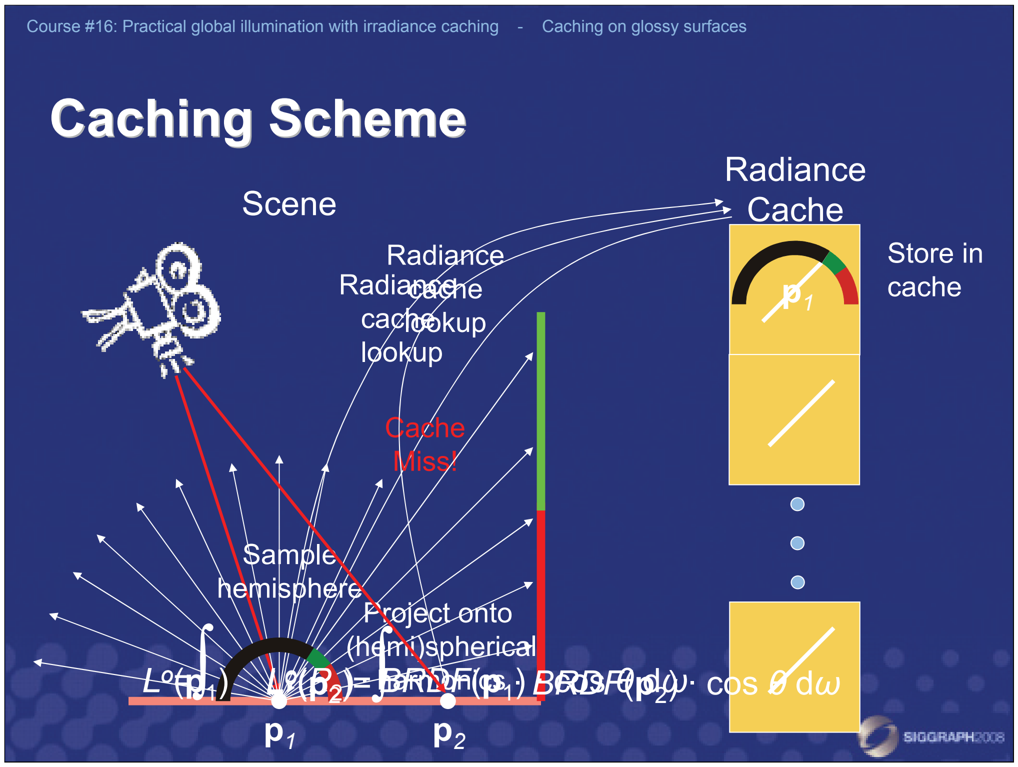
$$(\theta_j,\phi_k) = \left(\arcsin\frac{j+X_j}{M}, 2\pi\frac{k+Y_k}{N}\right)$$

To find the coefficients for the incoming radiance at a point, we need to numerically estimate the projection integral. Since our only means to evaluate the incoming radiance is point sampling – i.e. casting secondary rays – the obvious choice is Monte Carlo quadrature. We divide the hemisphere *uniformly* into *M* x *N* cells, where *M.N* is the desired number of rays (we use 500 – 8000 our renderings), and $N \approx 4M$. For each cell [j,k] we pick a random direction using the formula above. Note that this is slightly different from irradiance caching, since we are using uniform sampling (as opposed to cosine-proportional sampling in irradiance caching). For each sampling direction, we evaluate the basis functions, multiply them by the incoming radiance from that direction and accumulate to the coefficient vector.



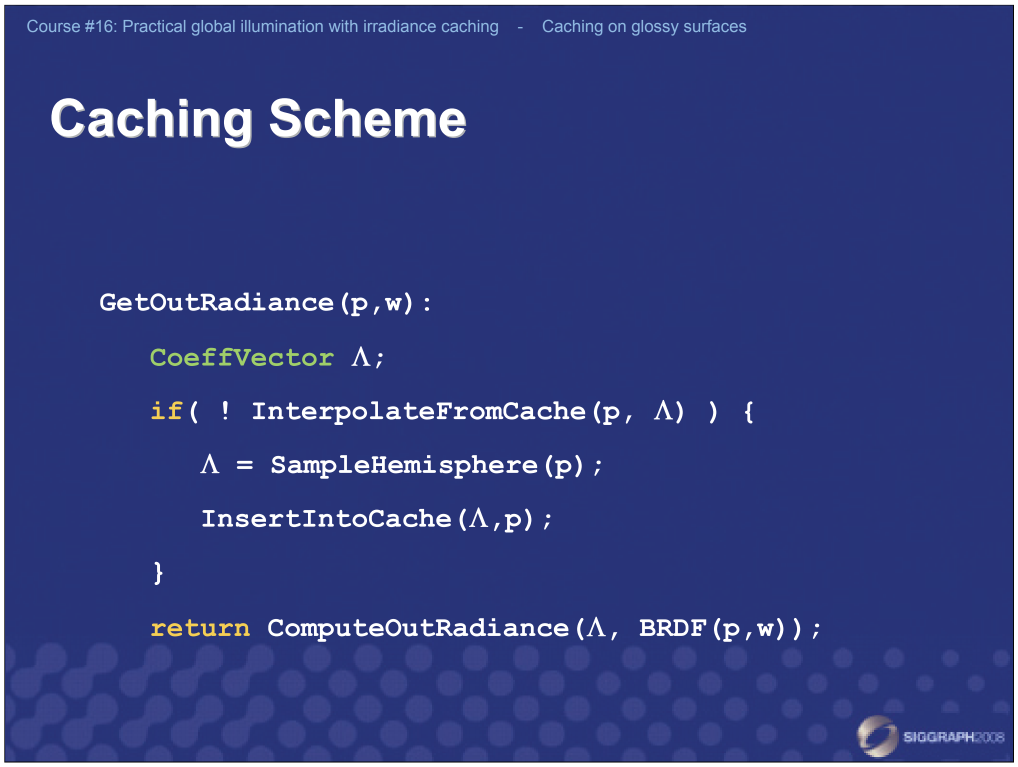
Spherical Harmonics

- Pros
  - Efficient rotation
  - Smooth – no aliasing
  - Little memory
  - Easy to use
- Cons
  - Only low-frequency BRDFs
  - Alternative – Wavelets


Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Caching Scheme

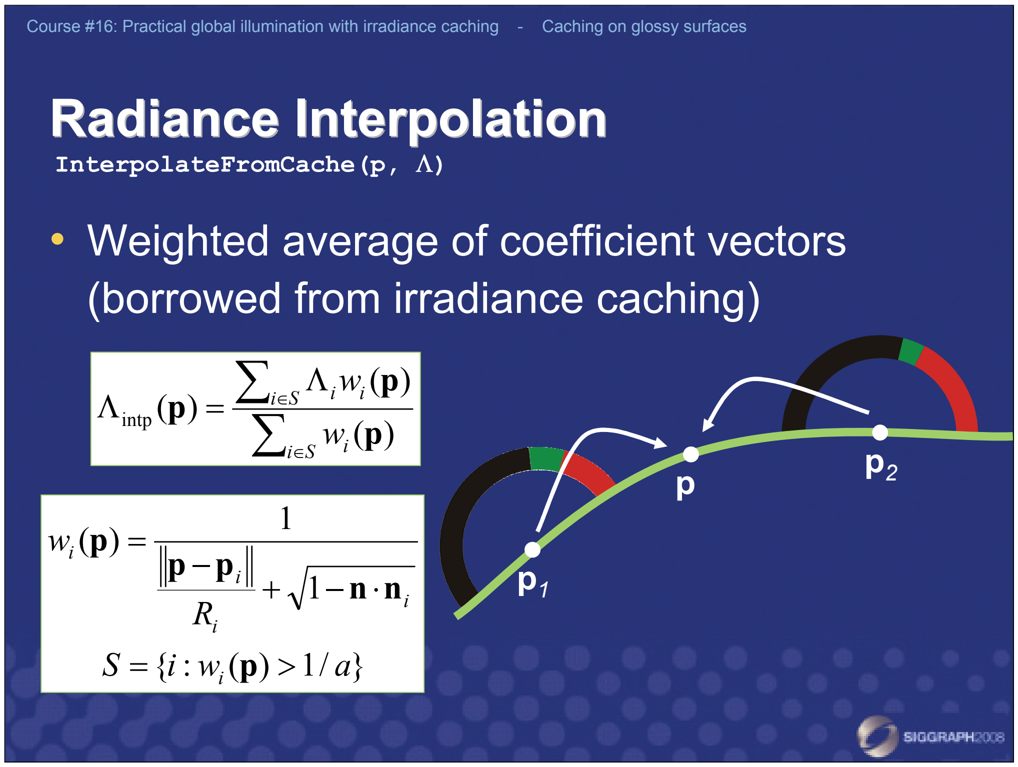
Radiance Cache
Scene
Radiance cache lookup
Radiance cache lookup
Store in cache
p_1
Cache Miss!
Sample hemisphere
Project onto (hemi)spherical
$L^o(p_1) = \int L^i(\omega) \cdot BRDF(p_1) \cdot \cos\theta \, d\omega$
$L^o(p_2) = \int L^i(\omega) \cdot BRDF(p_2) \cdot \cos\theta \, d\omega$
p_1
p_2
SIGGRAPH2008



Caching Scheme

```
GetOutRadiance(p,w):
    CoeffVector Λ;
    if( ! InterpolateFromCache(p, Λ) ) {
        Λ = SampleHemisphere(p);
        InsertIntoCache(Λ,p);
    }
    return ComputeOutRadiance(Λ, BRDF(p,w));
```

The overall caching scheme on glossy surfaces is the same as on diffuse ones. There are some important differences, though, that have to be taken care of. The procedure to compute outgoing radiance at a point due to glossy indirect illumination in a given (viewing) direction proceeds as shown on the slide. First, we query the radiance cache for previously computed records available for interpolation. If some are available, the coefficients for incoming radiance are interpolated from the cache. Otherwise, they are computed using hemisphere sampling as described on previous slides and stored in the cache. Finally, the incoming radiance is integrated against the BRDF to get the outgoing radiance. Various sub-routines of this procedure will be discussed on the following slides.



Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Radiance Interpolation

`InterpolateFromCache(p, Λ)`

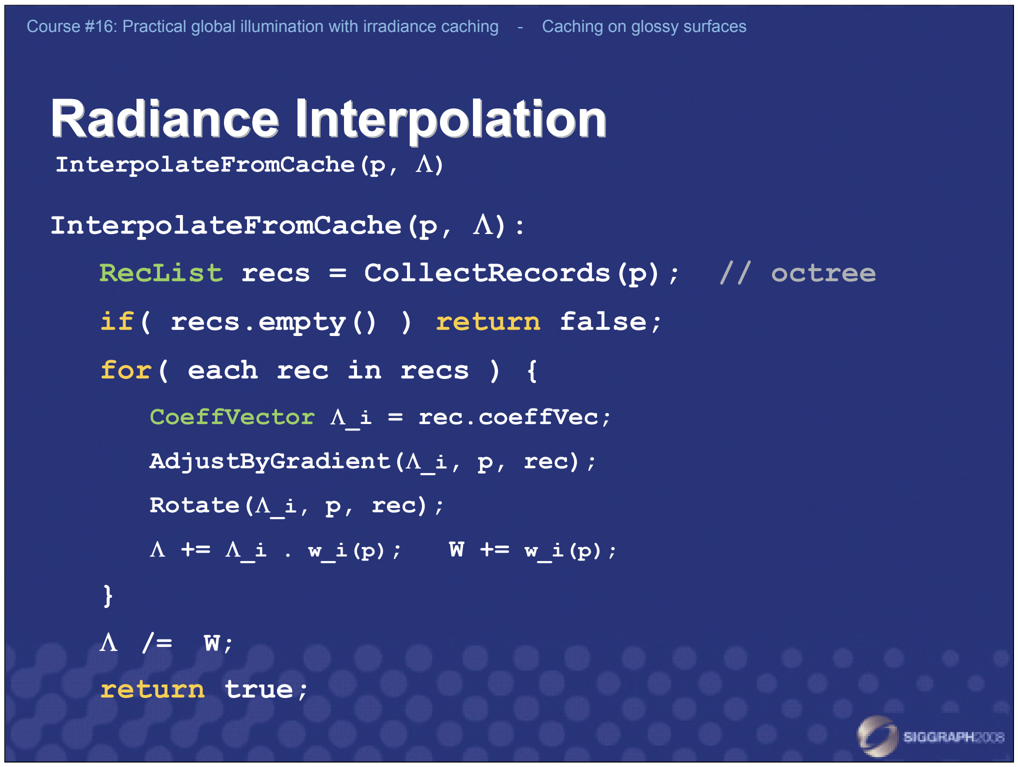
- Weighted average of coefficient vectors (borrowed from irradiance caching)

$$\Lambda_{\text{intp}}(\mathbf{p}) = \frac{\sum_{i \in S} \Lambda_i w_i(\mathbf{p})}{\sum_{i \in S} w_i(\mathbf{p})}$$

$$w_i(\mathbf{p}) = \frac{1}{\frac{\|\mathbf{p}-\mathbf{p}_i\|}{R_i} + \sqrt{1-\mathbf{n}\cdot\mathbf{n}_i}}$$

$$S = \{i : w_i(\mathbf{p}) > 1/a\}$$

To get the coefficient of interpolated incoming radiance at a point, we use the same weighted sum as used in irradiance caching. Instead of interpolating cached irradiance values, we interpolate the coefficient vectors. However the coefficient vectors have to be adjusted by translational gradients and by rotation as described later.



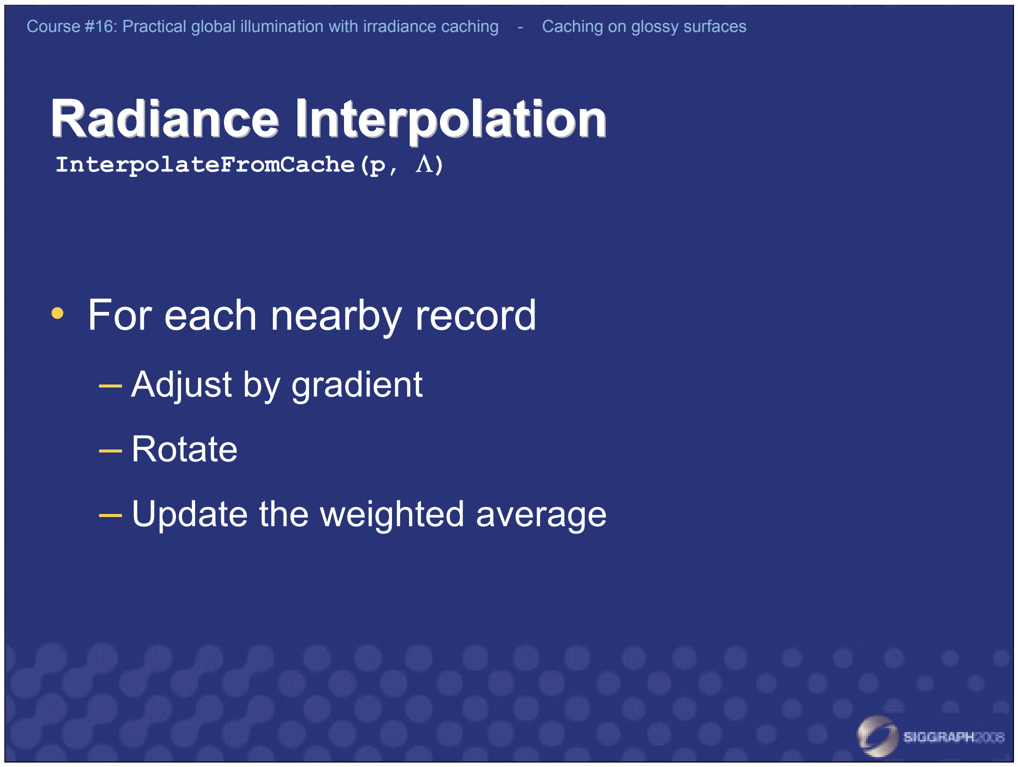
Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Radiance Interpolation

InterpolateFromCache(p, Λ)

```
InterpolateFromCache(p, Λ):
    RecList recs = CollectRecords(p);   // octree
    if( recs.empty() ) return false;
    for( each rec in recs ) {
        CoeffVector Λ_i = rec.coeffVec;
        AdjustByGradient(Λ_i, p, rec);
        Rotate(Λ_i, p, rec);
        Λ += Λ_i . w_i(p);    W += w_i(p);
    }
    Λ /= W;
    return true;
```

The radiance interpolation procedure starts by locating the existing records available for interpolation at **p**. This is uses the same octree structure as the irradiance cache. Then we loop over the records available for interpolation. For each record, we first adjust its coefficient vector by the translational gradient. The gradients are computed during hemisphere sampling and stored with the records in the cache. After the gradient adjustment, we apply a rotation to align coordinate frames at the point of interpolation and the location of the record.

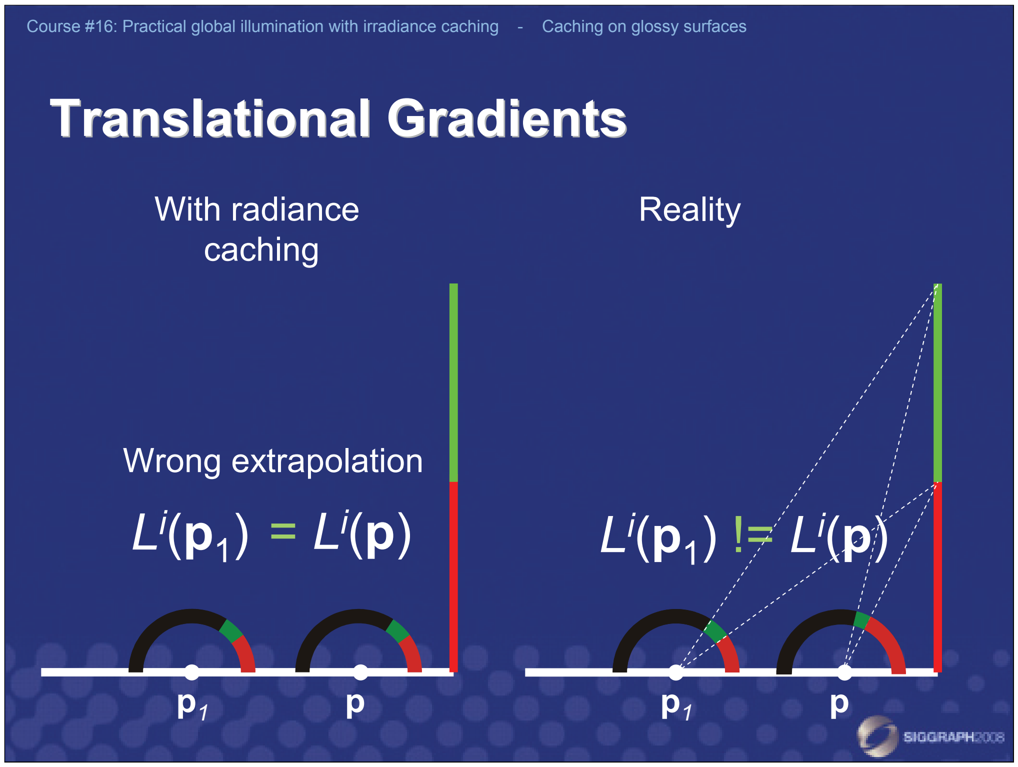


Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Radiance Interpolation

`InterpolateFromCache(p, Λ)`

- For each nearby record
  - Adjust by gradient
  - Rotate
  - Update the weighted average



Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Translational Gradients

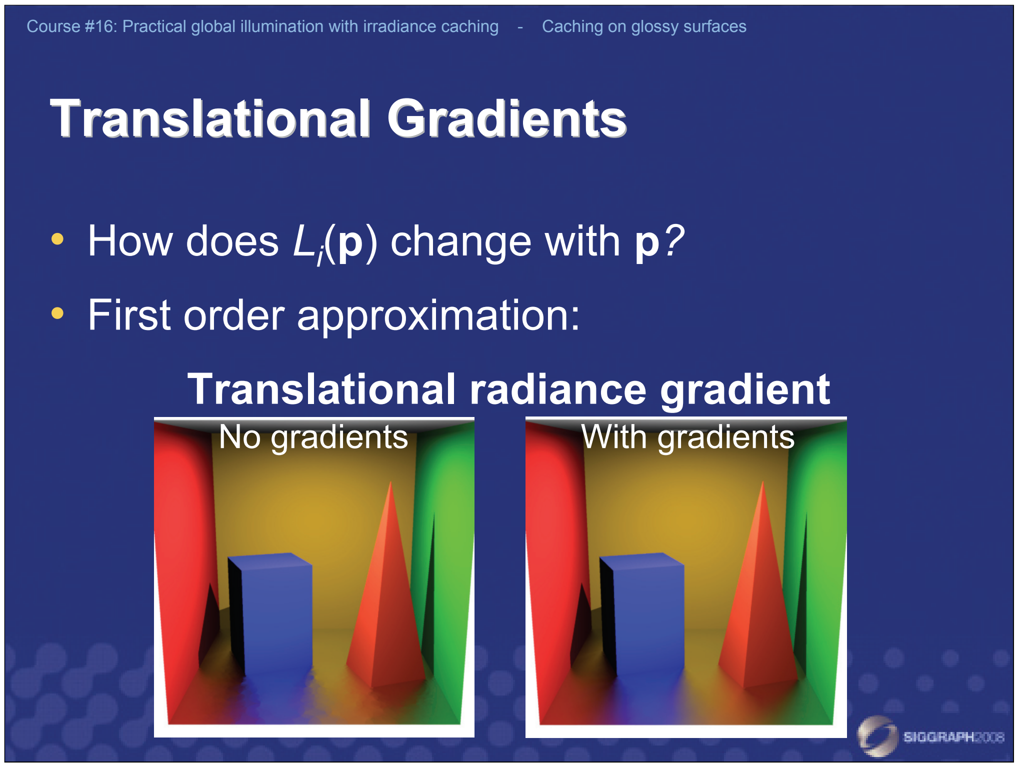
With radiance caching

Wrong extrapolation

$L^i(\mathbf{p}_1) = L^i(\mathbf{p})$

Reality

$L^i(\mathbf{p}_1) \mathrel{!=} L^i(\mathbf{p})$



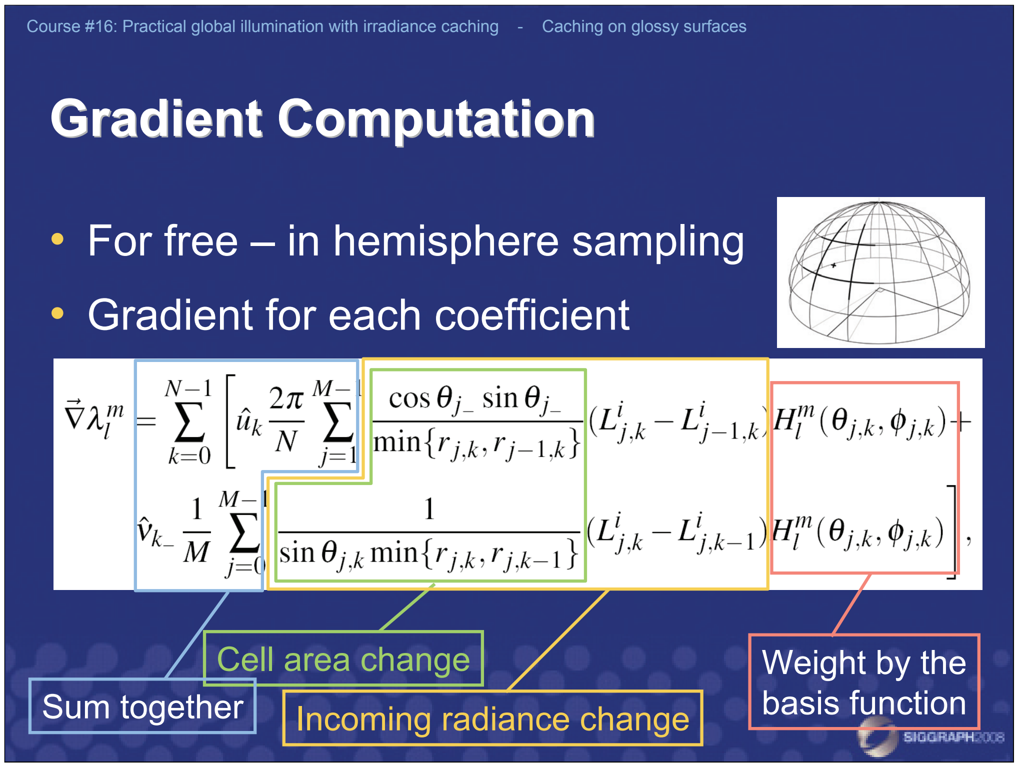
Translational Gradients

- How does $L_i(\mathbf{p})$ change with $\mathbf{p}$?
- First order approximation:

Translational radiance gradient

No gradients
With gradients

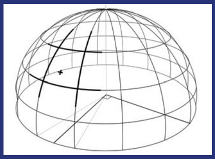
Translational gradients are used in interpolation to compensate for the change of incoming radiance with translation. If radiance is interpolated without the use of gradients, we see undesirable discontinuities in the images. The discontinuities are suppressed by the use of gradients.



Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Gradient Computation

- For free – in hemisphere sampling
- Gradient for each coefficient



$$\vec{\nabla}\lambda_l^m = \sum_{k=0}^{N-1}\left[\hat{u}_k \frac{2\pi}{N}\sum_{j=1}^{M-1}\frac{\cos\theta_{j_-}\sin\theta_{j_-}}{\min\{r_{j,k}, r_{j-1,k}\}}(L^i_{j,k} - L^i_{j-1,k})H_l^m(\theta_{j,k},\phi_{j,k}) + \hat{v}_{k_-}\frac{1}{M}\sum_{j=0}^{M-1}\frac{1}{\sin\theta_{j,k}\min\{r_{j,k}, r_{j,k-1}\}}(L^i_{j,k} - L^i_{j,k-1})H_l^m(\theta_{j,k},\phi_{j,k})\right],$$

Sum together

Cell area change

Incoming radiance change

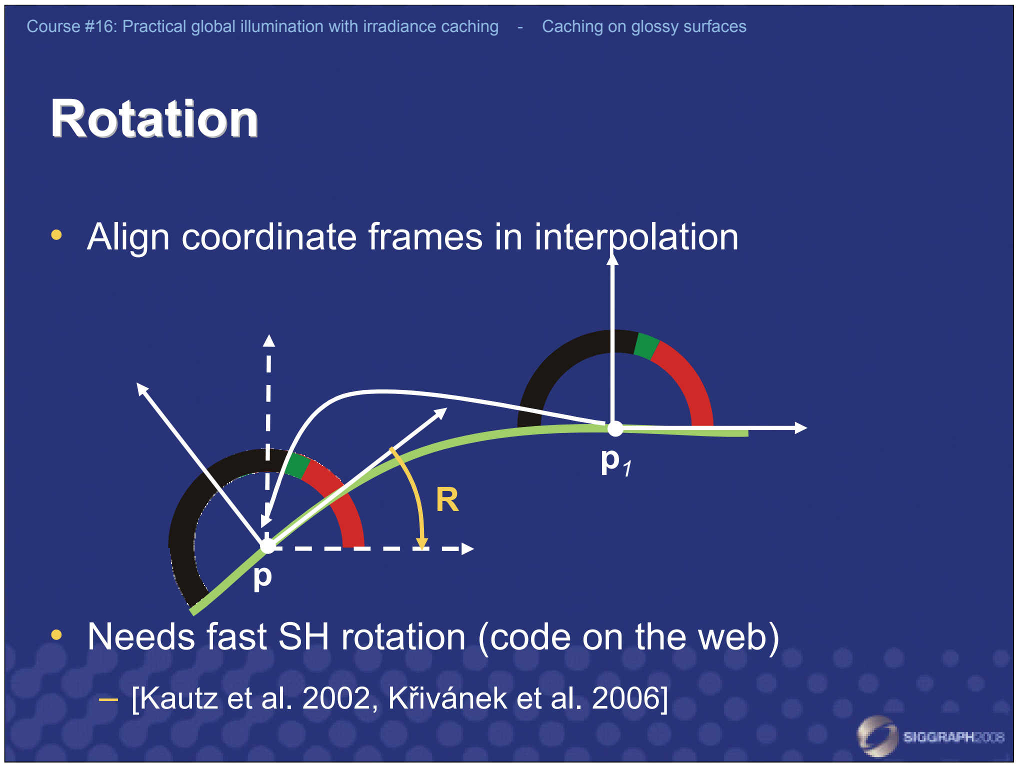
Weight by the basis function

SIGGRAPH2008

The gradients represent a first-order approximation of the change of incoming radiance with translation. For each coefficient of incoming radiance, there is one translational gradient, which describes how this coefficient changes with translation. Gradients are computed in hemisphere sampling along with the coefficient vectors using a formula shown above, and stored in the record.

To save space, we store the gradients in form of two coefficient derivative vectors – with respect to *x* and *y* axes of the local coordinate frame at the record. We disregard the gradient with respect to the local *z* axis, since the displacements along *z* are small in the interpolation. The *xy* plane is the local tangent plane and *z* is the normal vector.

The gradients are then used in interpolation to adjust the coefficient vector stored in the record.



Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Rotation

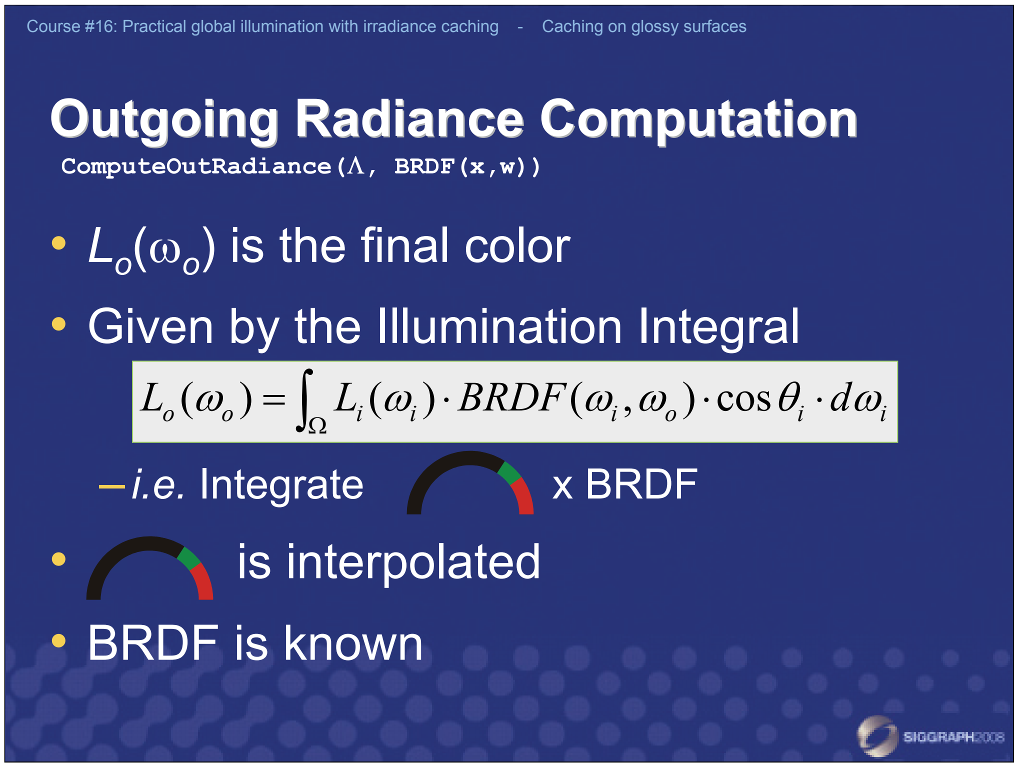
- Align coordinate frames in interpolation

- Needs fast SH rotation (code on the web)
  - [Kautz et al. 2002, Křivánek et al. 2006]

SIGGRAPH2008

After the gradient adjustment, the interpolation procedure proceeds by applying a rotation. This is required to align the coordinate frame at the point of interpolation with the coordinate frame with respect to which the coefficients of the incoming radiance were computed. The rotation procedure takes a vector of spherical harmonic coefficients for the incoming radiance and a 3x3 rotation matrix, and outputs a coefficient vector representing the rotated incoming radiance. This rotation procedure is the bottleneck of in the radiance interpolation and hence it is essential to have an optimized procedure for this. We won’t go into detail here and rather refer to the radiance caching web page where the rotation code can be downloaded.

This completes the interpolation procedure.



Outgoing Radiance Computation

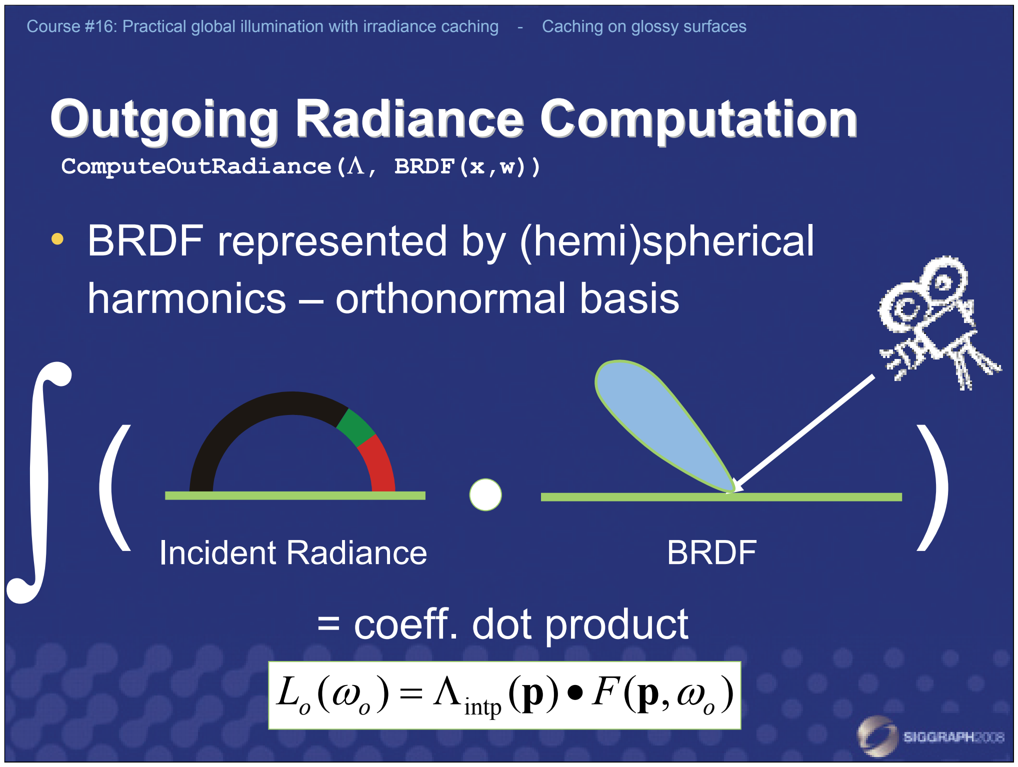
`ComputeOutRadiance(Λ, BRDF(x,w))`

- $L_o(\omega_o)$ is the final color
- Given by the Illumination Integral

$$L_o(\omega_o) = \int_{\Omega} L_i(\omega_i) \cdot BRDF(\omega_i, \omega_o) \cdot \cos\theta_i \cdot d\omega_i$$

  - *i.e.* Integrate x BRDF
- is interpolated
- BRDF is known

The final result we want to compute is the outgoing radiance at a point for a given outgoing direction – this is the color that corresponds to glossy indirect illumination. So far, we know how to compute or possibly interpolate the coefficient vector representing the directional distribution of incoming radiance at a point. Now we turn this into the outgoing radiance by evaluating the illumination integral, shown on the slide. Since the incoming radiance is interpolated and the BRDF can be looked up from the scene database, all the information required to evaluate the illumination integral is readily available.



Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Outgoing Radiance Computation

`ComputeOutRadiance(`Λ`, BRDF(x,w))`

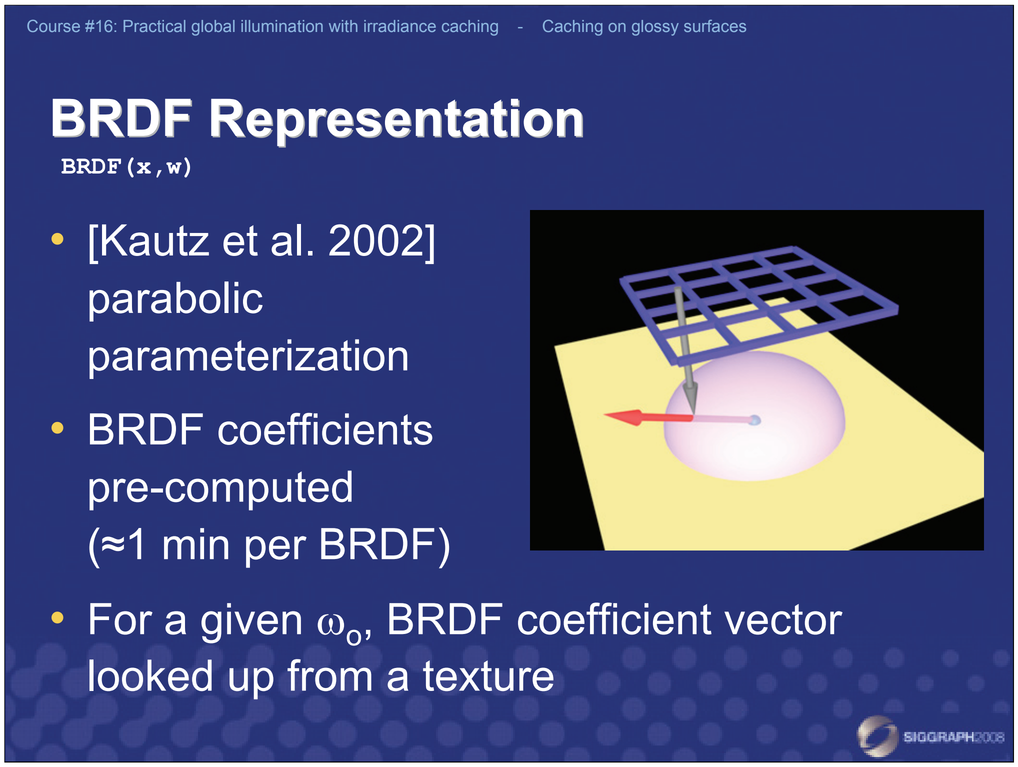
- BRDF represented by (hemi)spherical harmonics – orthonormal basis

Incident Radiance
BRDF

= coeff. dot product

$$L_o(\omega_o) = \Lambda_{\text{intp}}(\mathbf{p}) \bullet F(\mathbf{p}, \omega_o)$$

To make the evaluation of the illumination integral fast, we take advantage of the dot-product property of orthonormal bases (which spherical and hemispherical harmonics are): Integral of the product of two functions can be computed as the dot product of the coefficient vectors of these function with respect to the basis. So if we represent the BRDF times the cosine term using (hemi)spherical harmonics, all we need to do in order to compute the outgoing radiance is to dot the BRDF coefficients with the incoming radiance coefficients.

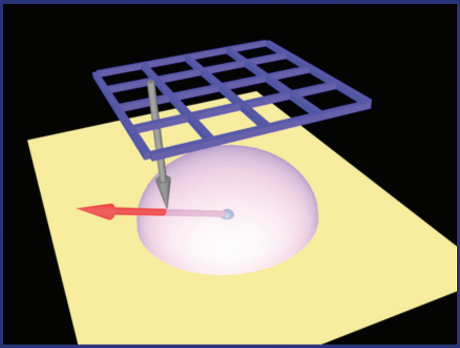


Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

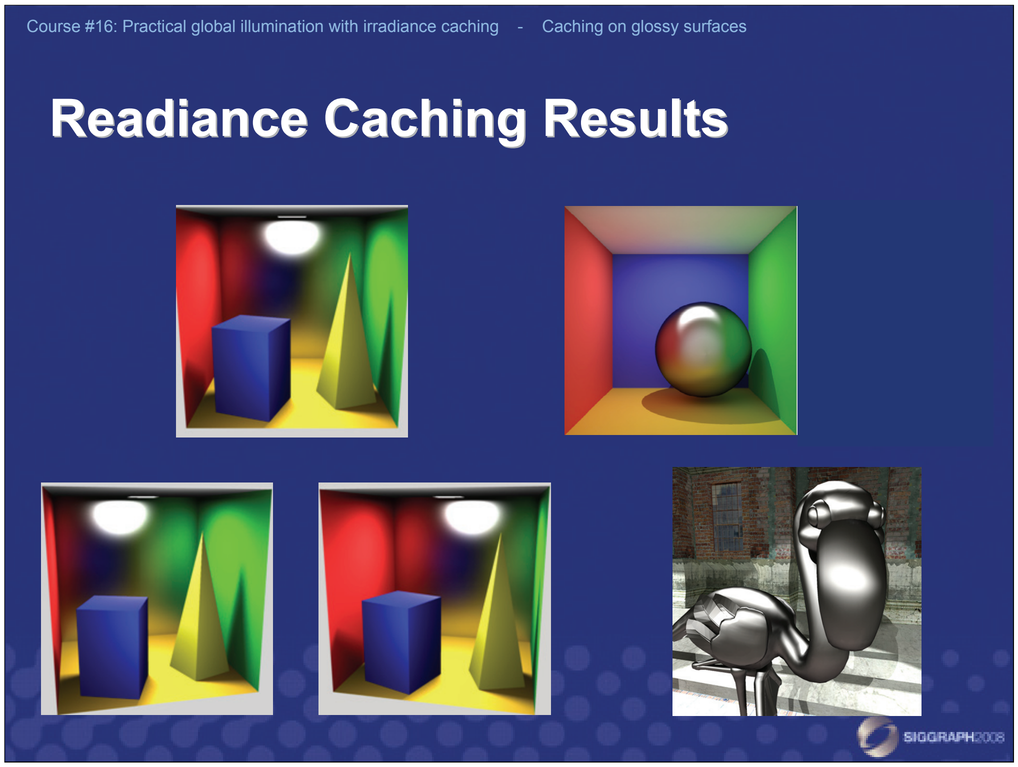
BRDF Representation

`BRDF(x,w)`

- [Kautz et al. 2002] parabolic parameterization
- BRDF coefficients pre-computed (≈1 min per BRDF)
- For a given ω_o, BRDF coefficient vector looked up from a texture



For a given, fixed, outgoing direction, the BRDF lobe is a hemispherical function. We represent this function by (hemi)spherical harmonics. We discretize the outgoing directions and for each discrete outgoing direction, we compute the coefficient vector representing the BRDF lobe. This is done in pre-process.

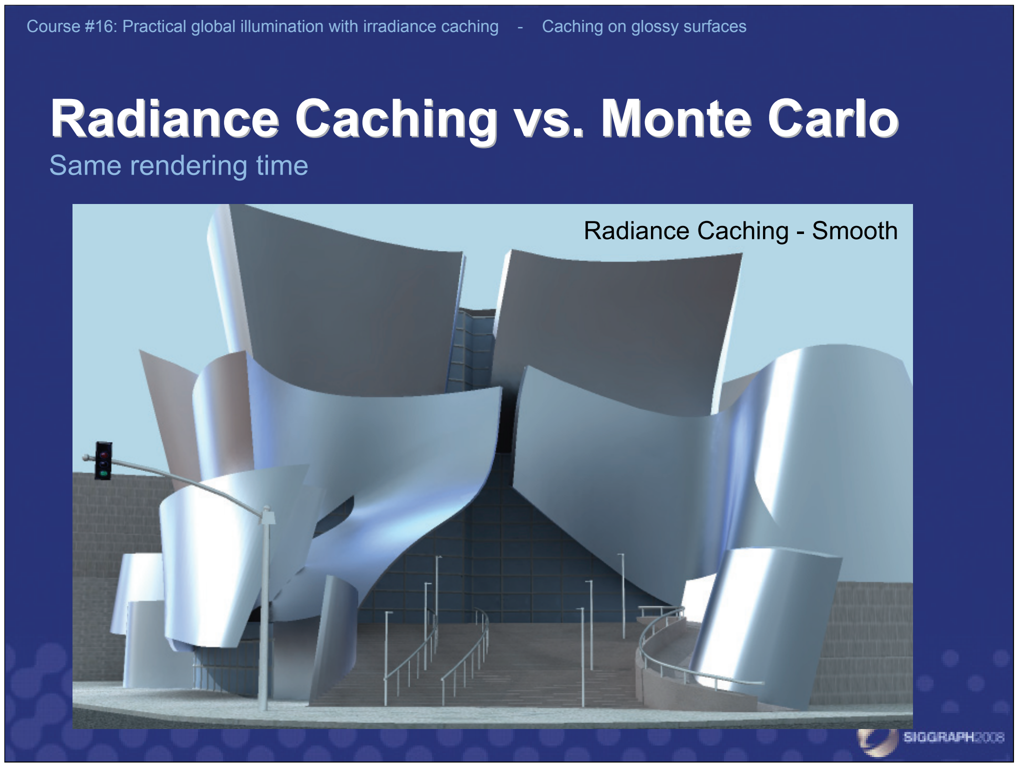


Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Readiance Caching Results

SIGGRAPH2008

This slide shows some images rendered with radiance caching. The images show that radiance caching can handle fairly sharp glossy reflections (boxes on the left), anisotropy (sphere) and curved geometry (flamingo). For sharper or more anisotropic reflections and for more complex geometry, radiance caching is usually outperformed by Monte Carlo importance sampling. As such, radiance caching complements importance sampling.

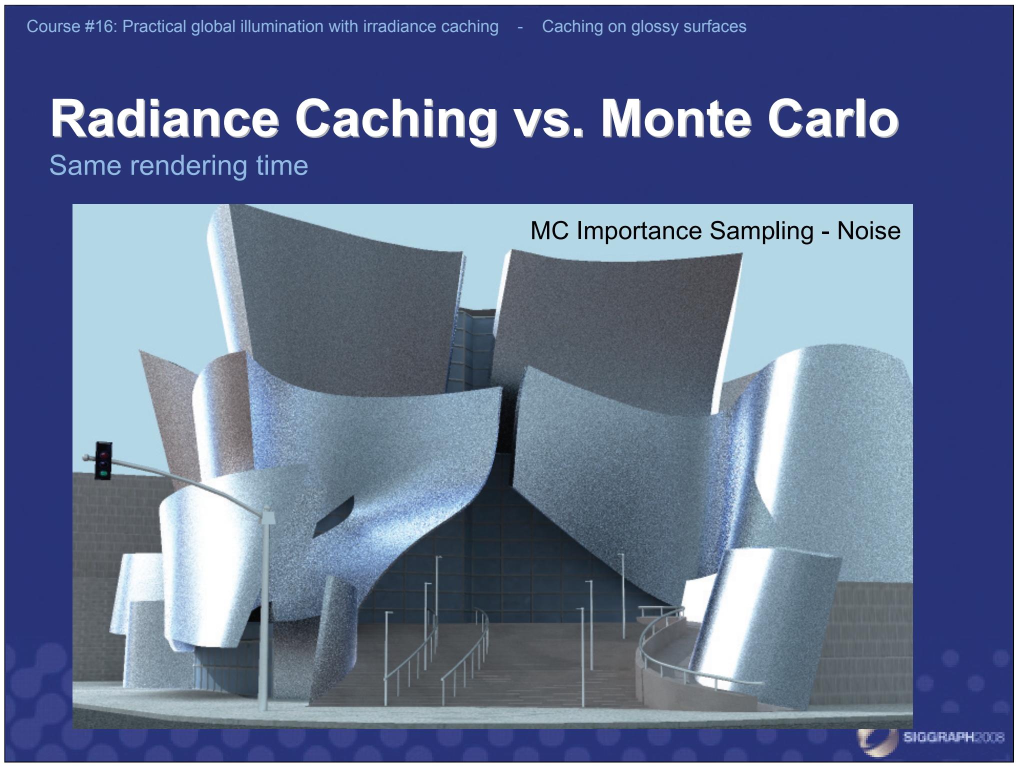


Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Radiance Caching vs. Monte Carlo

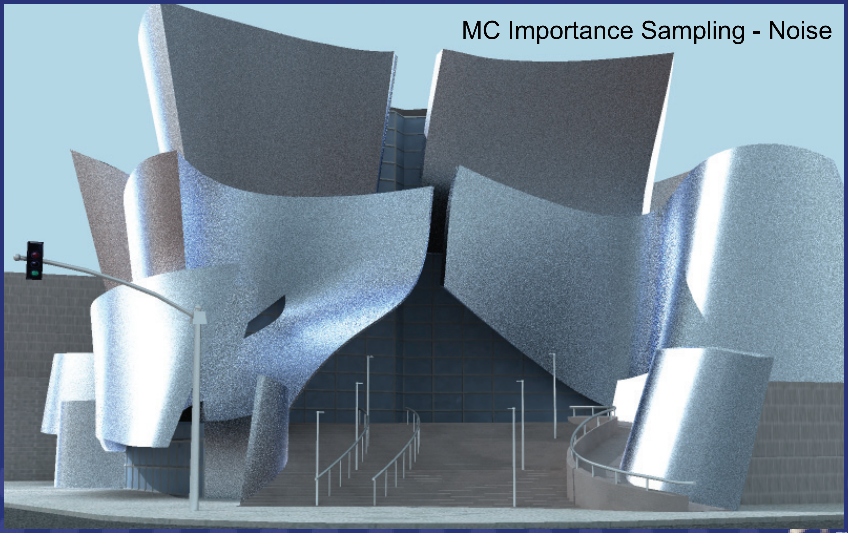
Same rendering time

Given the same rendering time, radiance caching, produces smooth rendering.

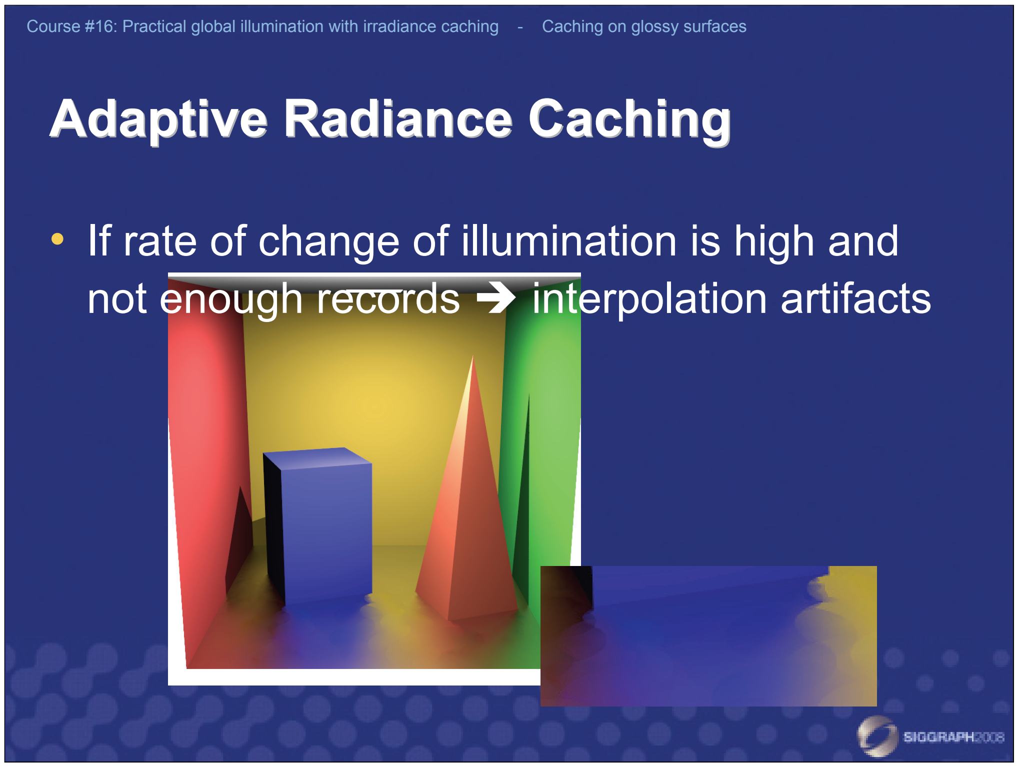


Radiance Caching vs. Monte Carlo

Same rendering time



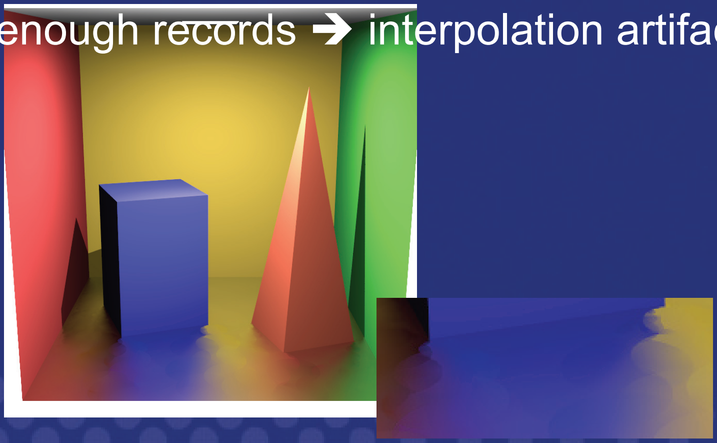
This and the following slide compare image quality delivered by MC importance sampling and radiance caching given the same rendering time. Here we see that Monte Carlo produces noisy image.



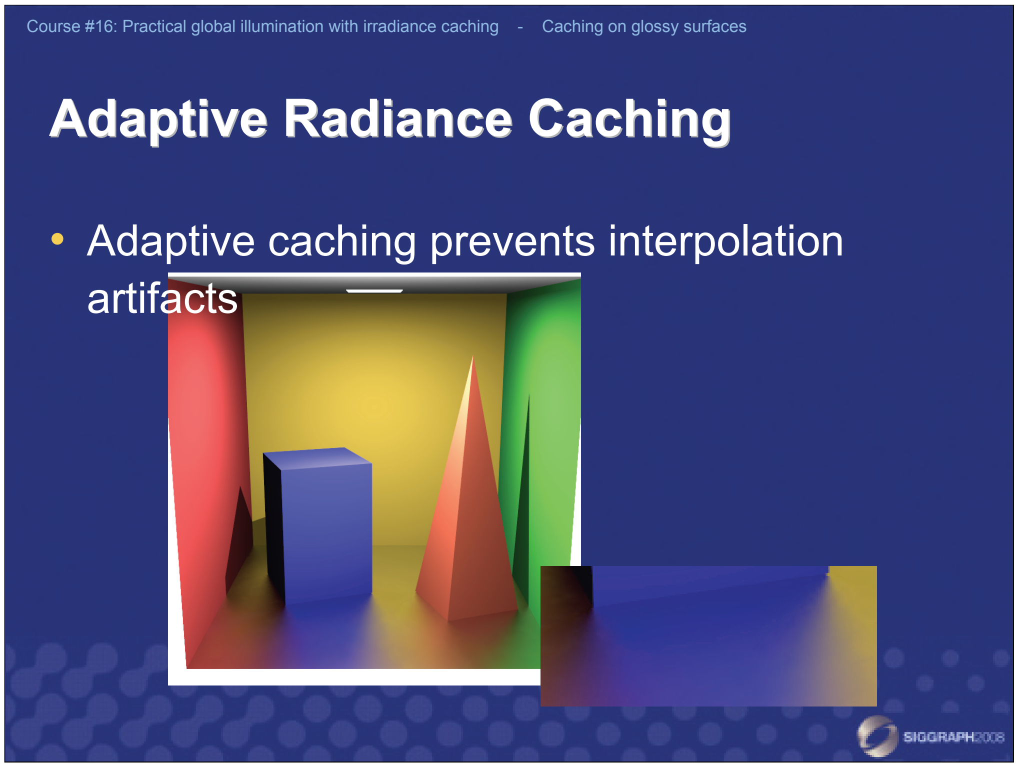
Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Adaptive Radiance Caching

- If rate of change of illumination is high and not enough records ➔ interpolation artifacts



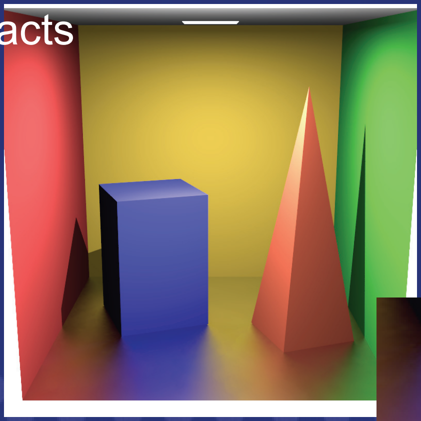
The basic radiance caching algorithm as described so far works fine in many cases, but it fails more often than irradiance caching.



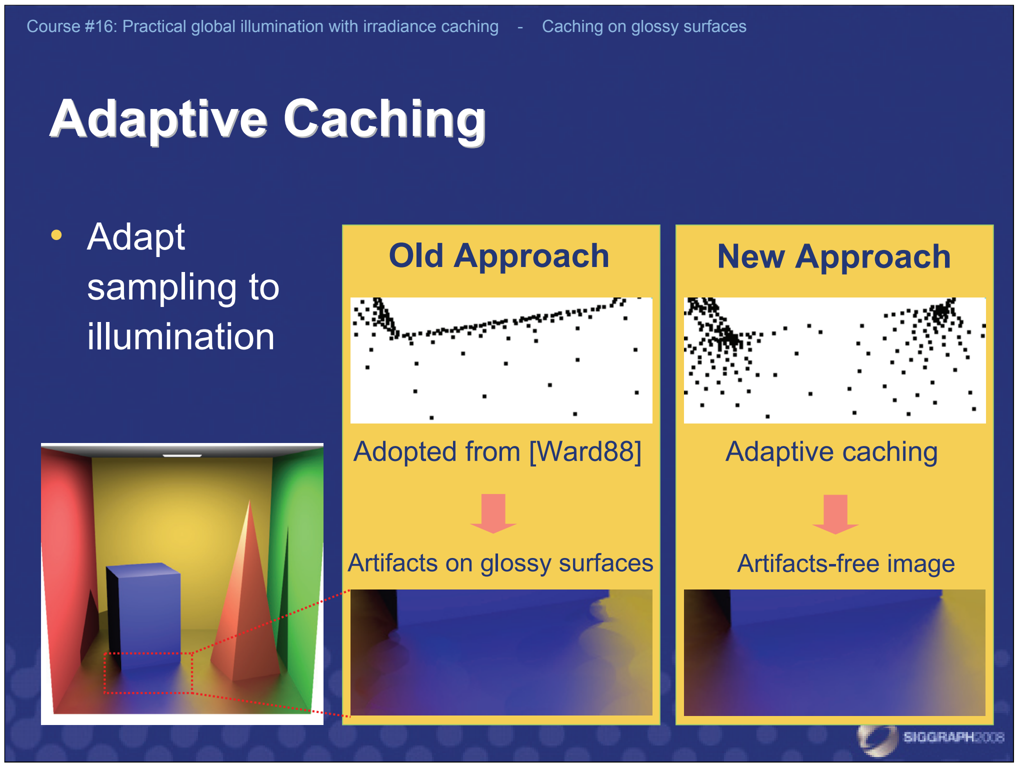
Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Adaptive Radiance Caching

- Adaptive caching prevents interpolation artifacts



We use adaptive radiance caching to handle the failure cases.



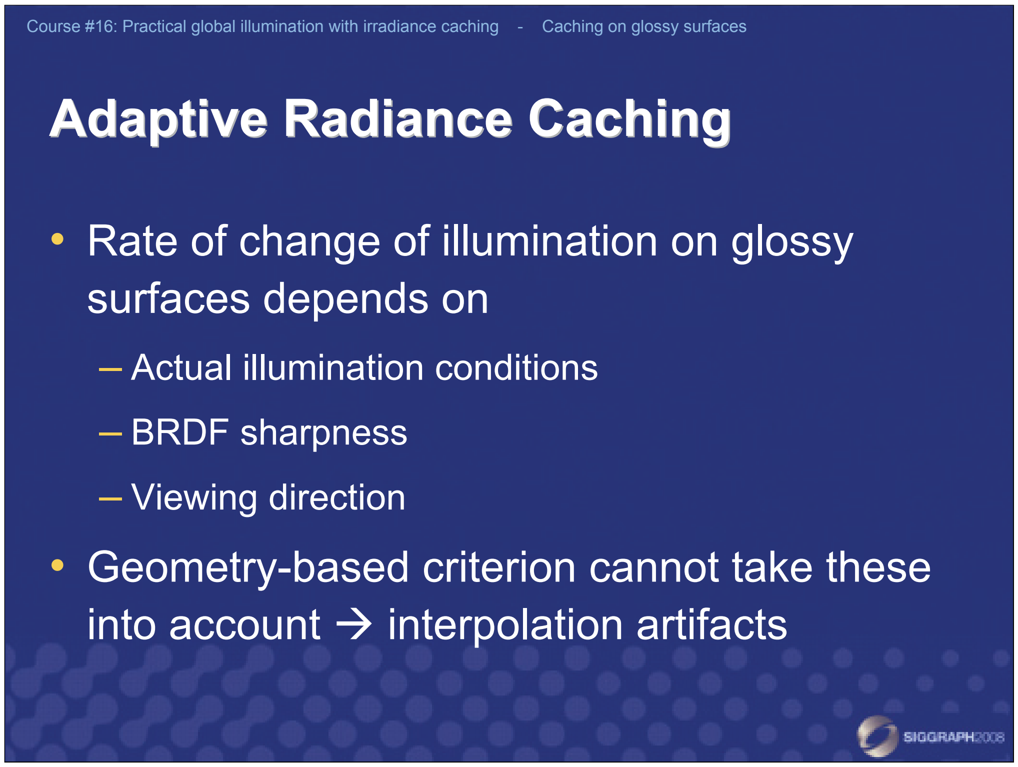
Adaptive Caching

- Adapt sampling to illumination

Old Approach	New Approach
Adopted from [Ward88]	Adaptive caching
Artifacts on glossy surfaces	Artifacts-free image

SIGGRAPH2008

In particular, we adapt the density of spatial sampling of the indirect illumination to the actual local illumination conditions. This is in contrast to irradiance caching, where the sampling density is adapted based solely on the scene geometry.

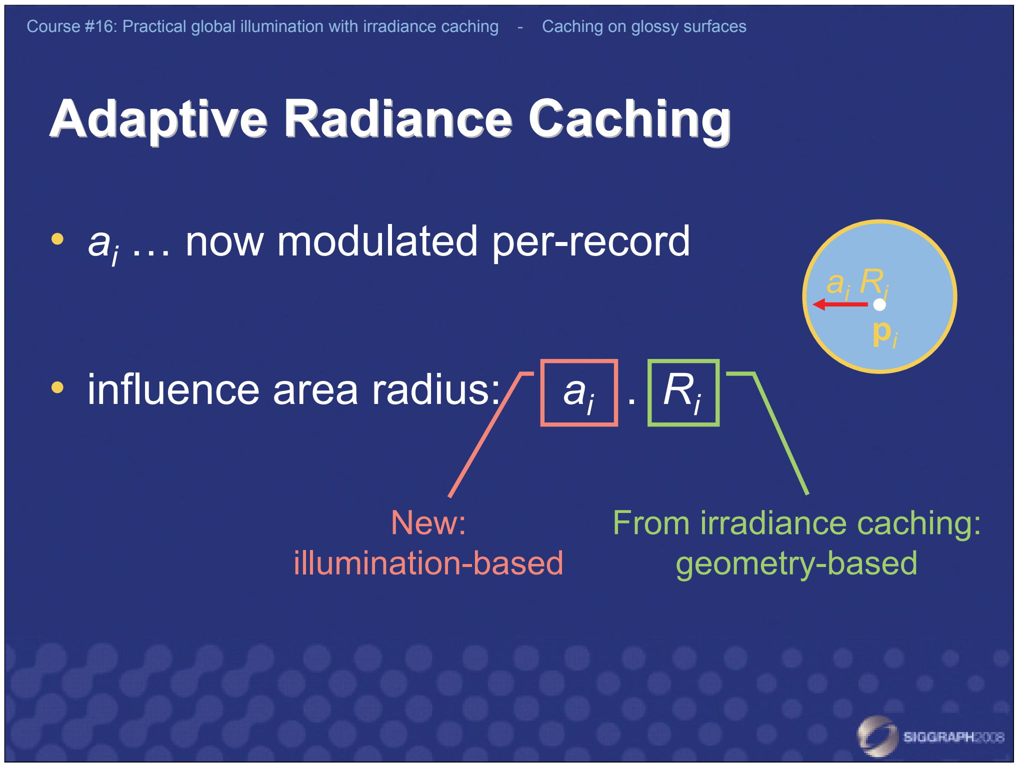


Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Adaptive Radiance Caching

- Rate of change of illumination on glossy surfaces depends on
  - Actual illumination conditions
  - BRDF sharpness
  - Viewing direction
- Geometry-based criterion cannot take these into account → interpolation artifacts

When we take the geometry-based criterion from irradiance caching and use it on glossy surfaces, we may fail to predict important illumination changes. This is because indirect glossy illumination (as opposed to indirect diffuse) very much depends on the actual illumination conditions in the scene. In addition, it depends on the BRDF sharpness and the viewing direction.

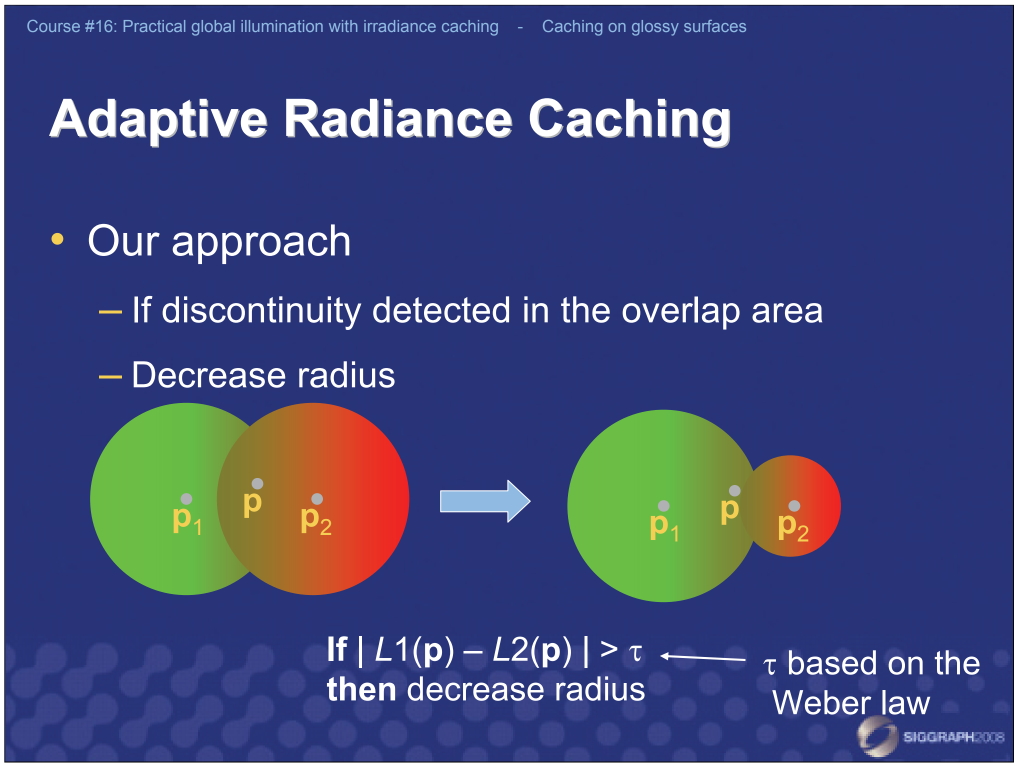


Adaptive Radiance Caching

- a_i … now modulated per-record
- influence area radius: $\boxed{a_i} \cdot \boxed{R_i}$

$a_i\,R_i$
$\mathbf{p}_i$
New:
illumination-based
From irradiance caching:
geometry-based
SIGGRAPH2008

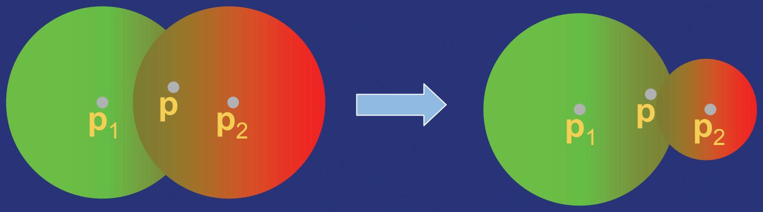
Remember that in irradiance caching and in the basic radiance caching, the record spacing is determined by the mean distance to the surrounding geometry, denoted R_i, multiplied by a user supplied constant a. In adaptive radiance caching, we extend this by also modulating the allowed error a automatically on a per-record basis in order to produce higher sampling in areas of high lighting complexity.



Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Adaptive Radiance Caching

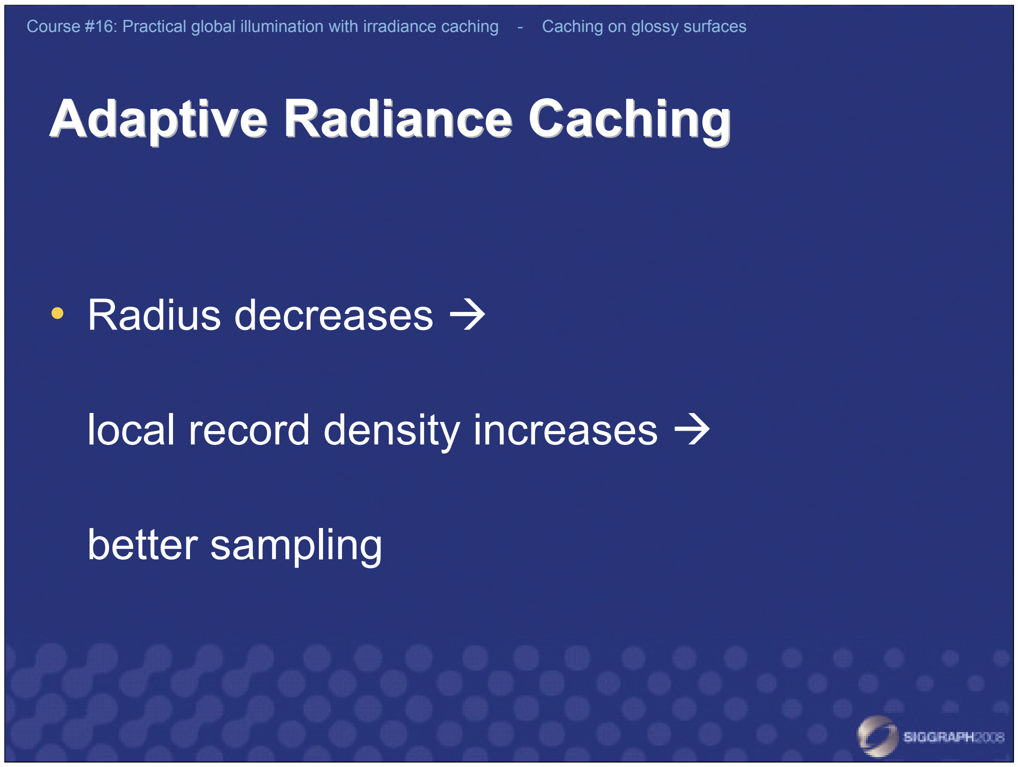
- Our approach
  - If discontinuity detected in the overlap area
  - Decrease radius



If | $L1(\mathbf{p}) - L2(\mathbf{p})$ | > τ ← τ based on the Weber law
then decrease radius

SIGGRAPH2008

To adapt the a_i-value to the local illumination conditions, we proceed as follows. During the radiance interpolation, we perform an additional check on the difference of radiance values of the contributing records. If this difference is perceptually important, we conclude that a visible discontinuity would be produced by interpolation. To determine the perceptual importance, we use the simplest possible metric – the Weber law. It states that the minimum perceivable difference of luminance is given by a fixed fraction of the luminance value. We use the threshold of 2 to 3 %. If a potentially visible discontinuity is detected, we decrease the a_i value associated with one of the records in order to exclude it from interpolation. This ‘shrinks’ the area over which that record can be reused for interpolation.



Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Adaptive Radiance Caching

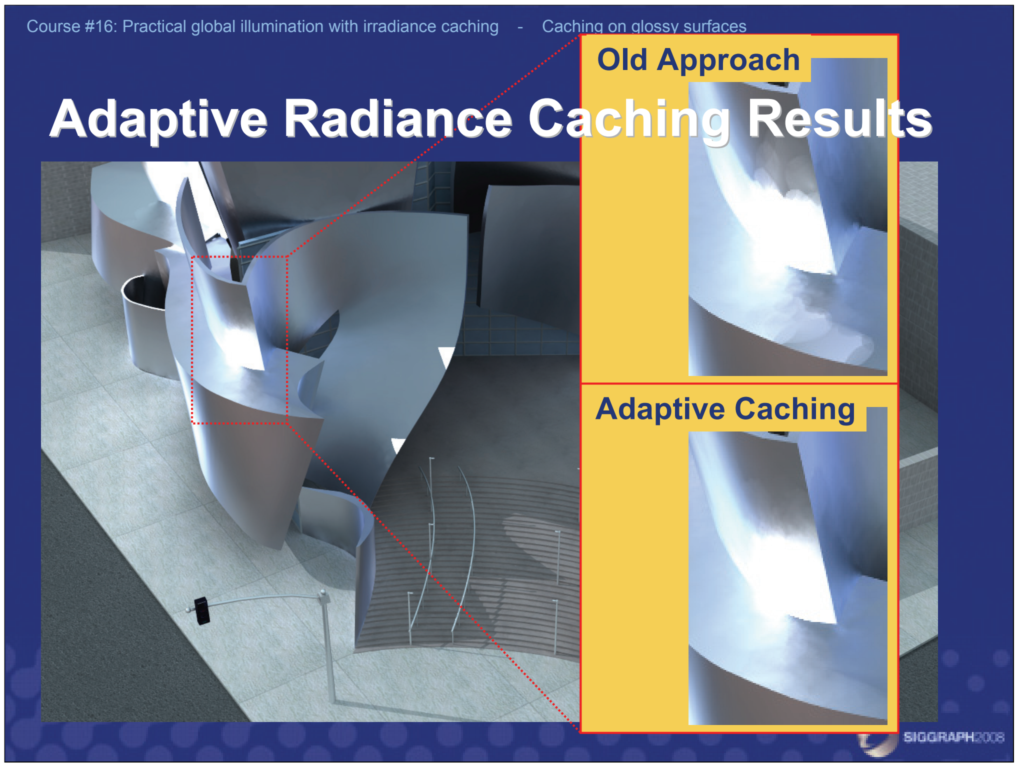
- Radius decreases →

  local record density increases →

  better sampling

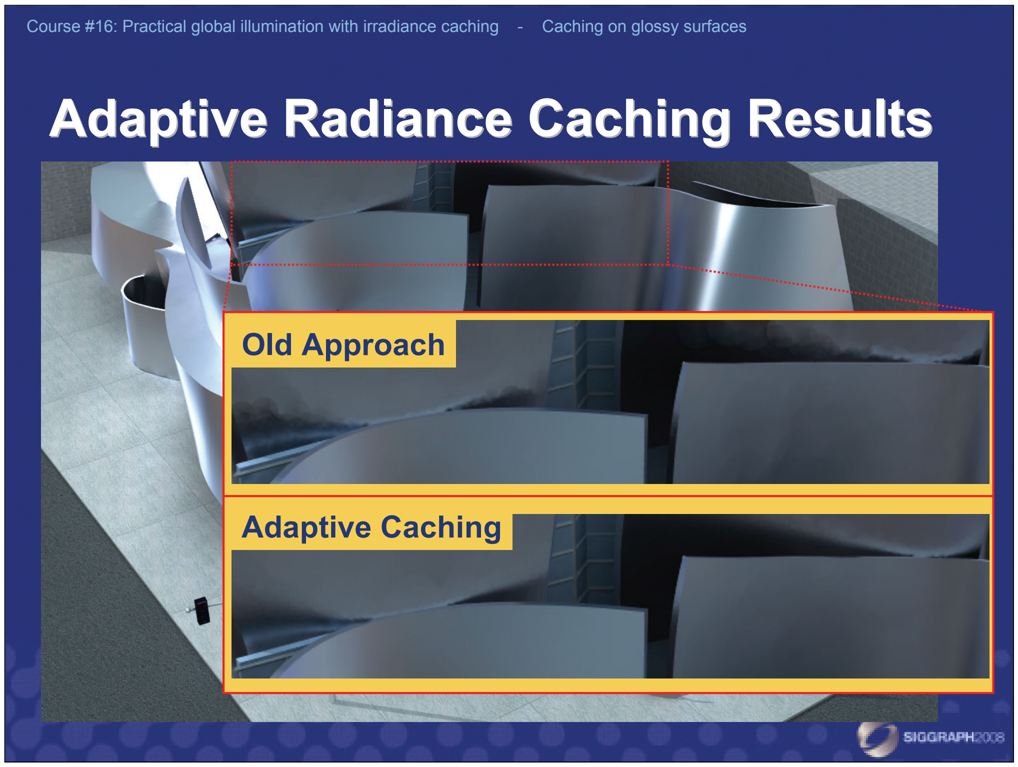
SIGGRAPH2008

Shrinking the influence area of a records has for consequence increased local density of radiance records and thus better local sampling of indirect illumination.

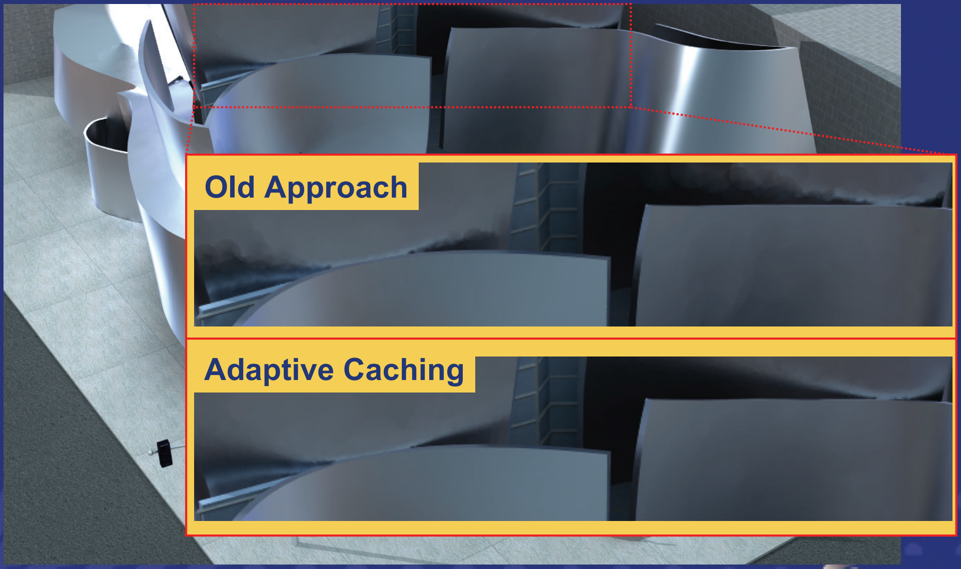

Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

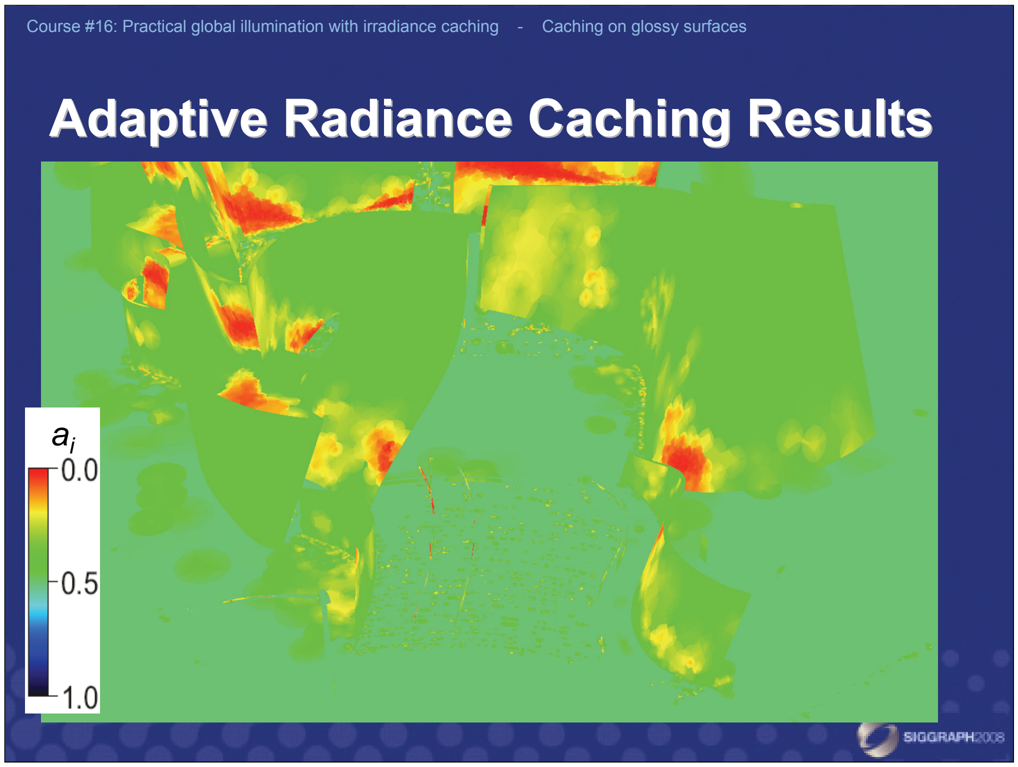
Adaptive Radiance Caching Results

This and the following slide show the effect of adaptive radiance caching on the rendered images. The two rendering were created in the same rendering time. Without adaptive radiance caching, we see some discontinuities because of the interpolation, which are successfully suppressed by adaptive caching.



Adaptive Radiance Caching Results

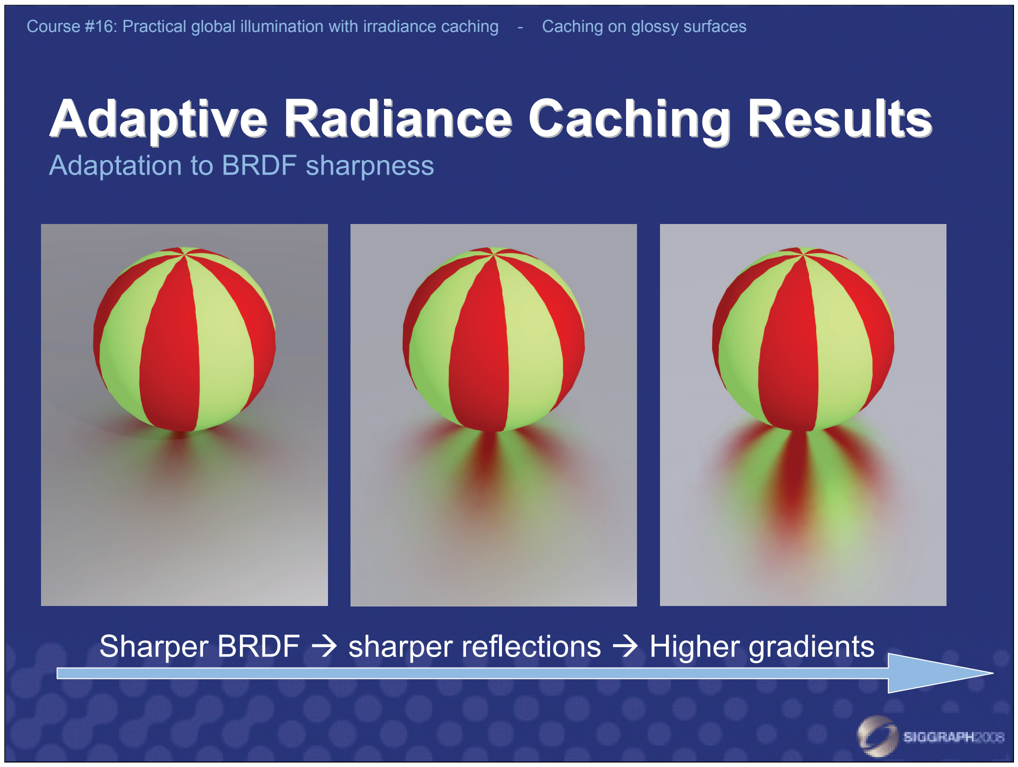




Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Adaptive Radiance Caching Results

This image shows the locally adapted *ai* value for the rendering of the Walt Disney Hall.



Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

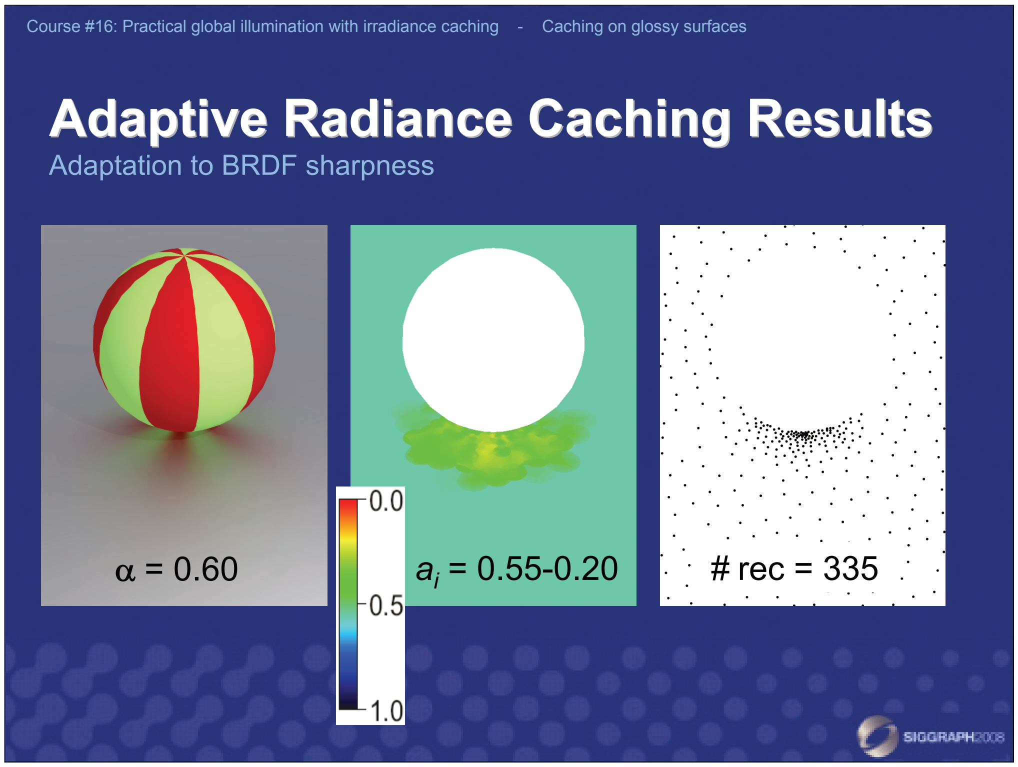
Adaptive Radiance Caching Results

Adaptation to BRDF sharpness

Sharper BRDF → sharper reflections → Higher gradients

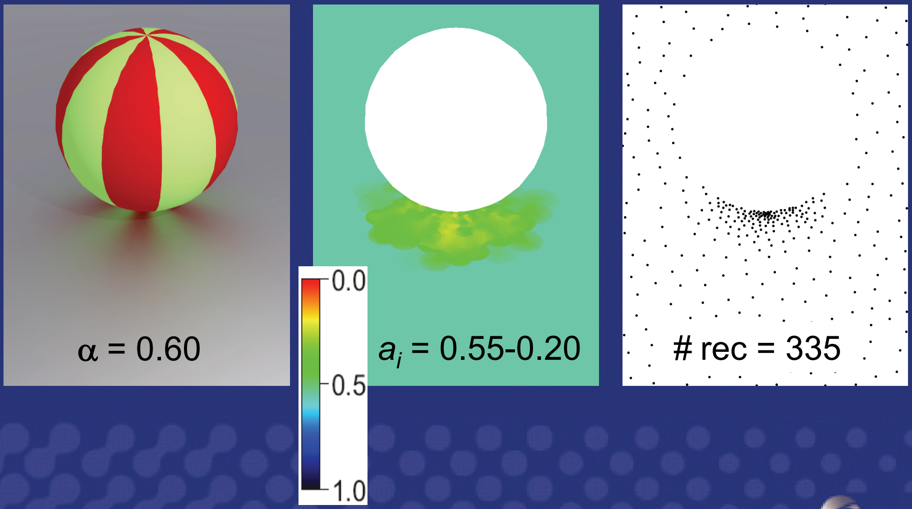
SIGGRAPH2008

Adaptive caching also automatically adapts the record density to the BRDF sharpness. On these three images, the BRDF sharpness of the glossy floor gradually increases from left to right, giving sharper and sharper reflections. Sharper reflections result in higher illumination gradients.

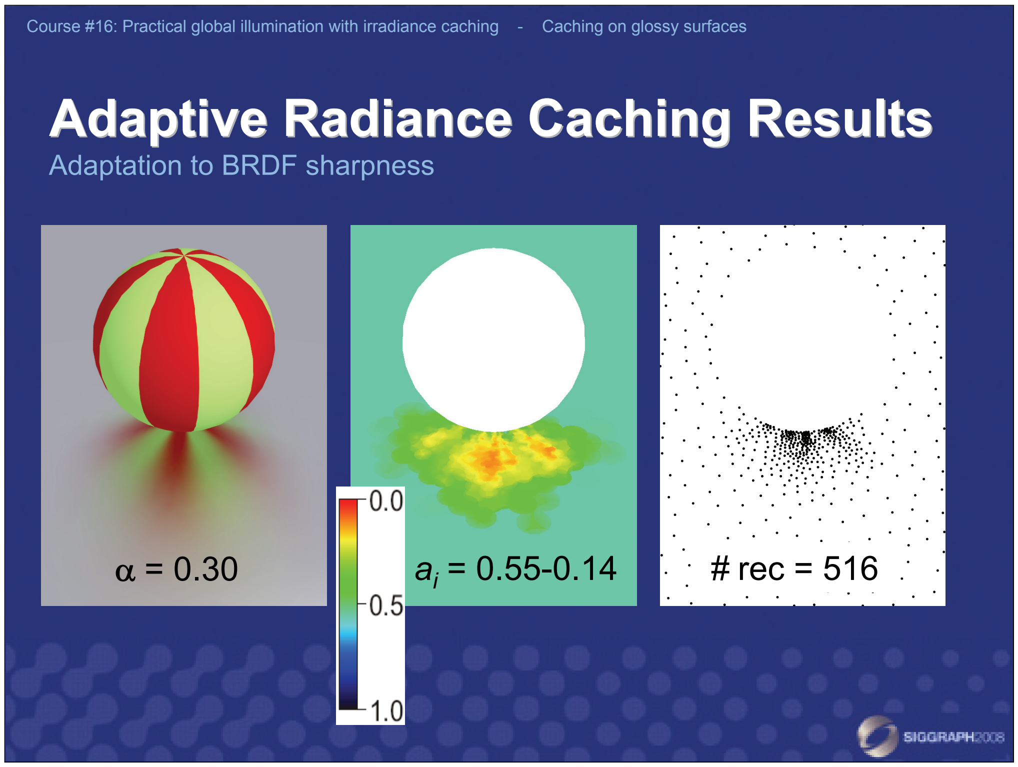


Adaptive Radiance Caching Results

Adaptation to BRDF sharpness



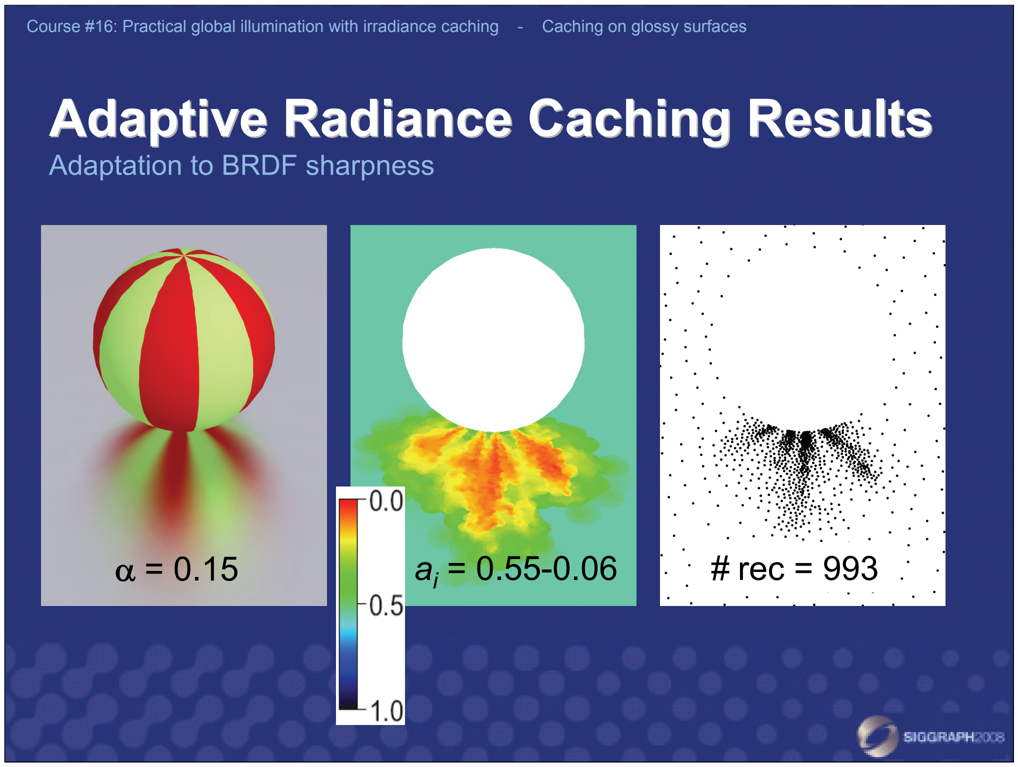
When the BRDF sharpness is small, we can get away with very sparse sampling.



Adaptive Radiance Caching Results

Adaptation to BRDF sharpness

As the BRDF sharpness increases, the adaptive caching increases the record density accordingly, in order to avoid interpolation artifacts.

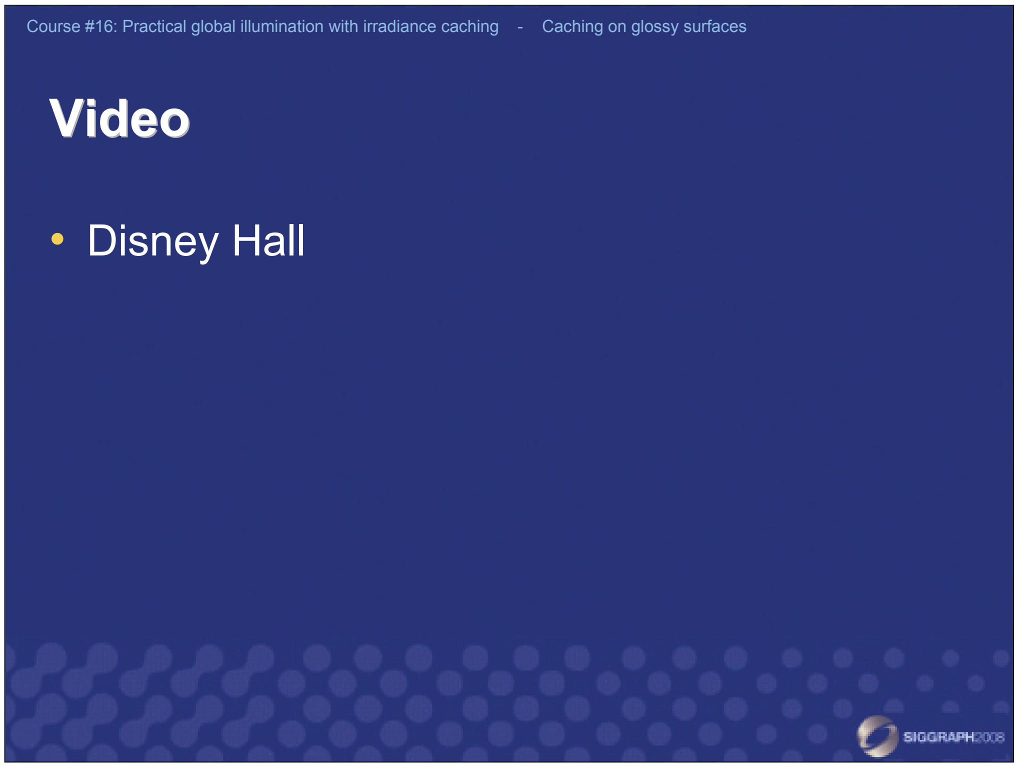


Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Adaptive Radiance Caching Results

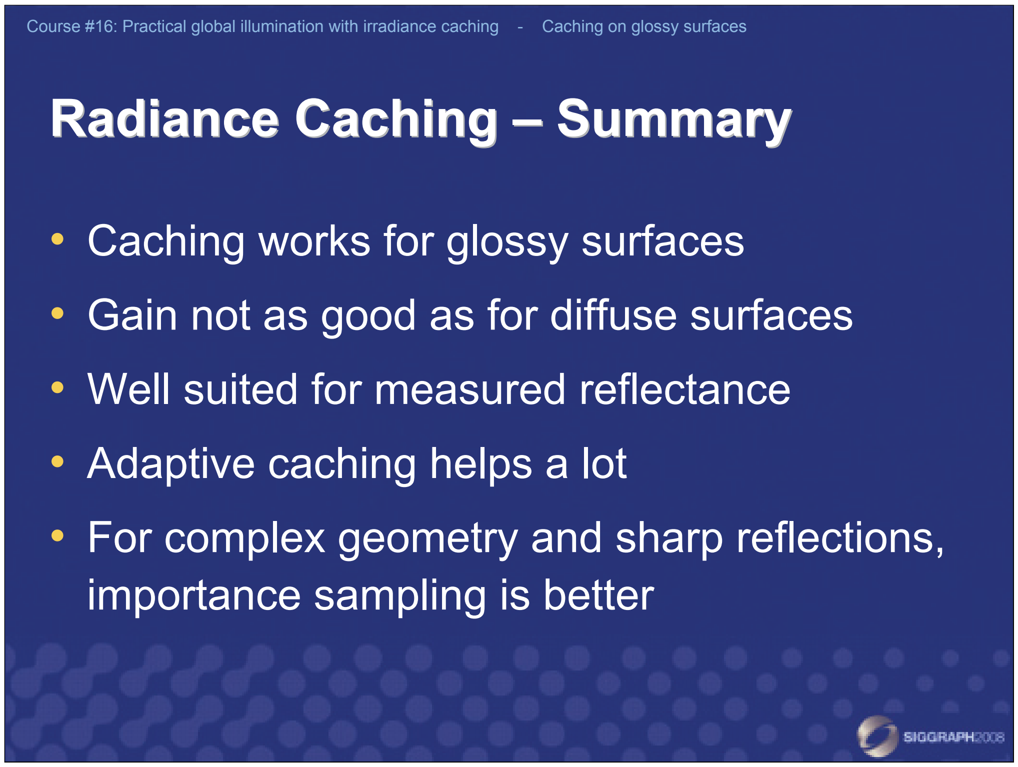
Adaptation to BRDF sharpness

Increasing the BRDF sharpness further, still more records are automatically added by adaptive caching to preserve smoothness of indirect illumination. Notice that if we based the interpolation criterion a priori on the BRDF sharpness, we would have to increase the sampling density all over the glossy floor, which would result in a overly dense sampling and a significant performance drop.



Video

- Disney Hall



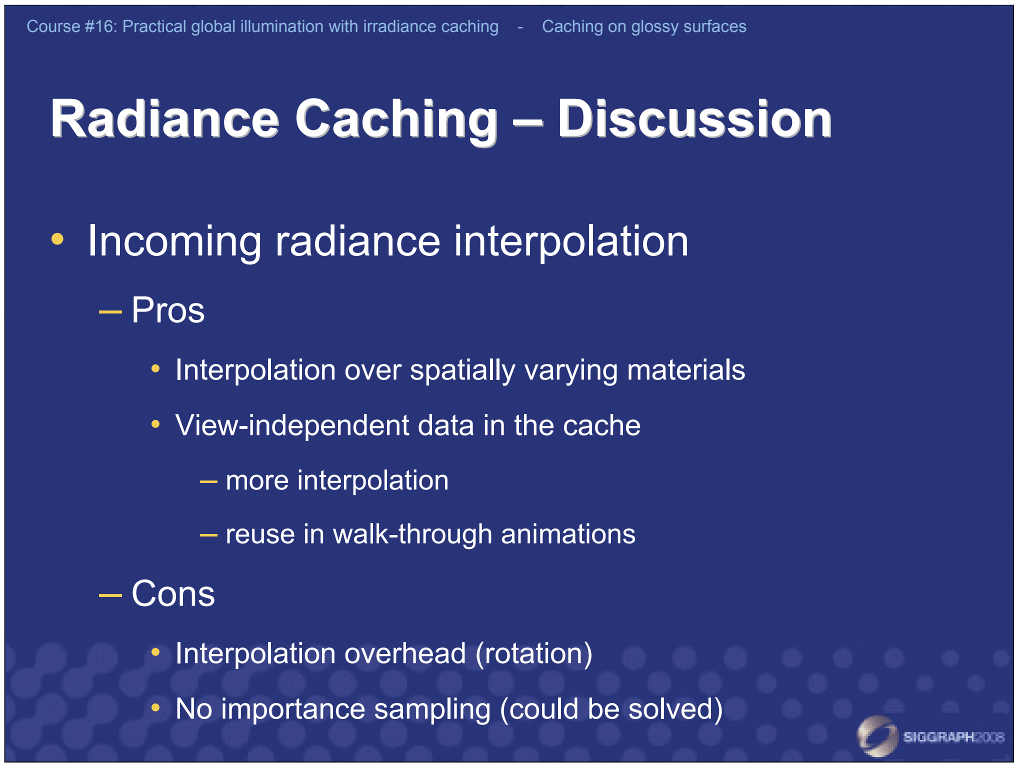
Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Radiance Caching – Summary

- Caching works for glossy surfaces
- Gain not as good as for diffuse surfaces
- Well suited for measured reflectance
- Adaptive caching helps a lot
- For complex geometry and sharp reflections, importance sampling is better

SIGGRAPH2008

We have shown that lazy illumination caching can be used not only for diffuse surfaces, but even for glossy ones. The performance gain is not as substantial as for irradiance caching, though. The main overhead in radiance caching is due to uniform hemisphere sampling in radiance caching – this requires many sampling rays and, for a certain threshold of BRDF sharpness, importance sampling out-performs radiance caching. In addition, the interpolation performance is crippled by the spherical harmonic rotation. Nevertheless, radiance caching brings significant performance gains over Monte Carlo importance sampling when rendering fairly smooth geometry with low-frequency BRDF. In particular, it effective for rendering with measured reflectance data, for which importance sampling is difficult.



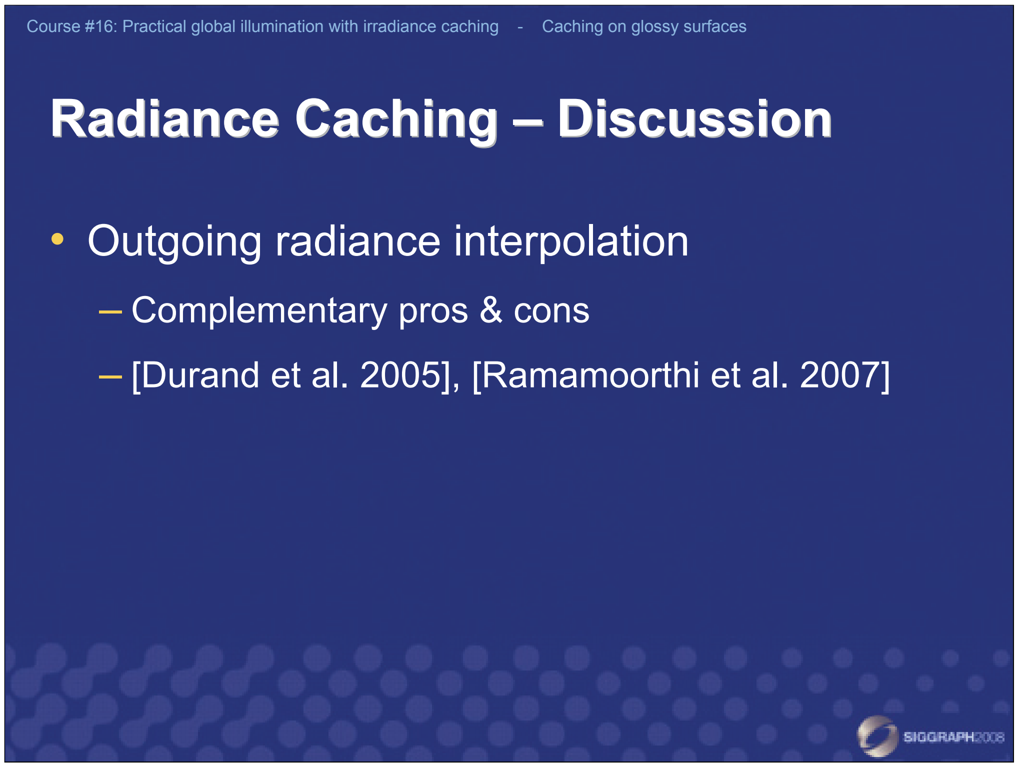
Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Radiance Caching – Discussion

- Incoming radiance interpolation
  - Pros
    - Interpolation over spatially varying materials
    - View-independent data in the cache
      - more interpolation
      - reuse in walk-through animations
  - Cons
    - Interpolation overhead (rotation)
    - No importance sampling (could be solved)

The essential design choice in radiance caching is to interpolate the directional distribution of incoming radiance (as opposed to interpolation outgoing radiance for the viewing direction). Since the cached quantity is view-independent, we can reuse cached values over larger areas and we can re-use the them in animations. In addition, we can also interpolate over spatially varying materials.

On the other hand, the overhead related to caching and interpolating a directional function may be quite large as discussed on the previous slide.

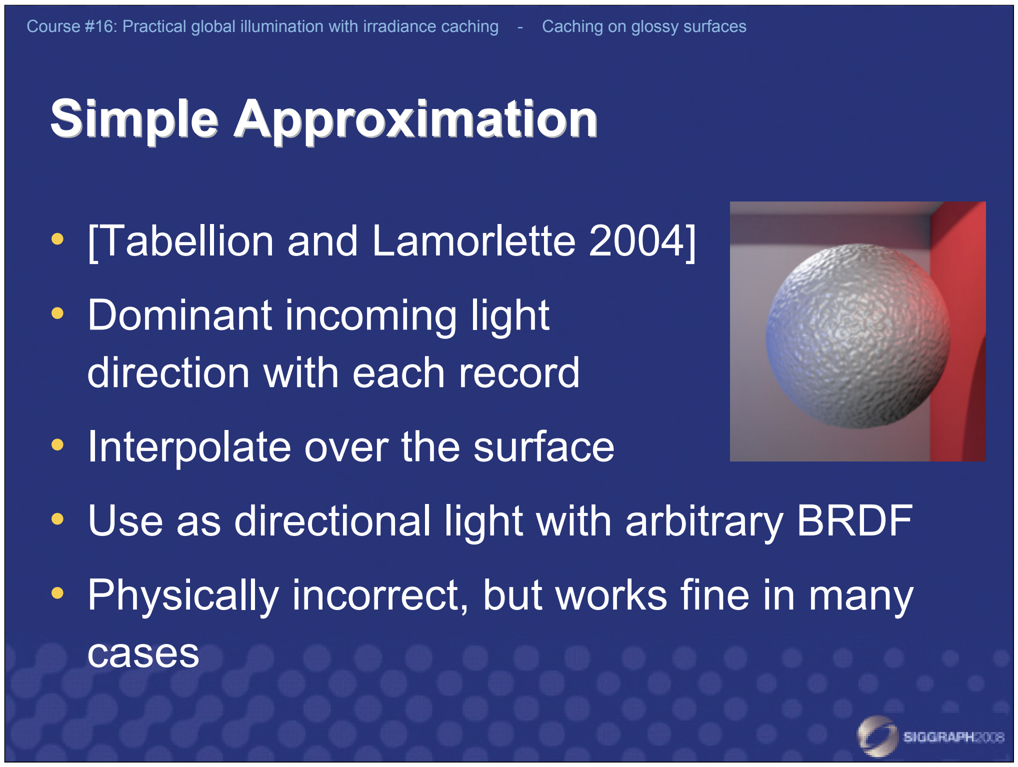


Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Radiance Caching – Discussion

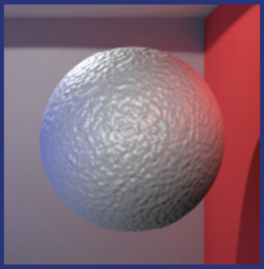
- Outgoing radiance interpolation
  - Complementary pros & cons
  - [Durand et al. 2005], [Ramamoorthi et al. 2007]

An alternative to caching incoming radiance is to interpolate outgoing radiance for the viewing direction.



Simple Approximation

- [Tabellion and Lamorlette 2004]
- Dominant incoming light direction with each record
- Interpolate over the surface
- Use as directional light with arbitrary BRDF
- Physically incorrect, but works fine in many cases



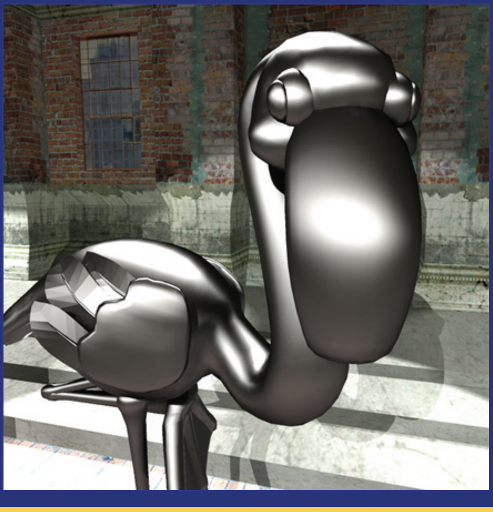
An alternative method for illumination interpolation on glossy surfaces was proposed by Tablellion and Lamorlette [2004].

It is a simple extension of irradiance caching. When sampling the hemisphere, the dominant incoming light direction is computed weighted average of sample directions weighted by the radiance coming from that direction. This dominant direction is stored along with irradiance in the cache for possible interpolation. When shading a surface with arbitrary, non-diffuse BRDF, the interpolated irradiance is interpreted as coming all from the dominant direction, i.e. as if it was a directional light source. A standard surface shader is then used to compute color.

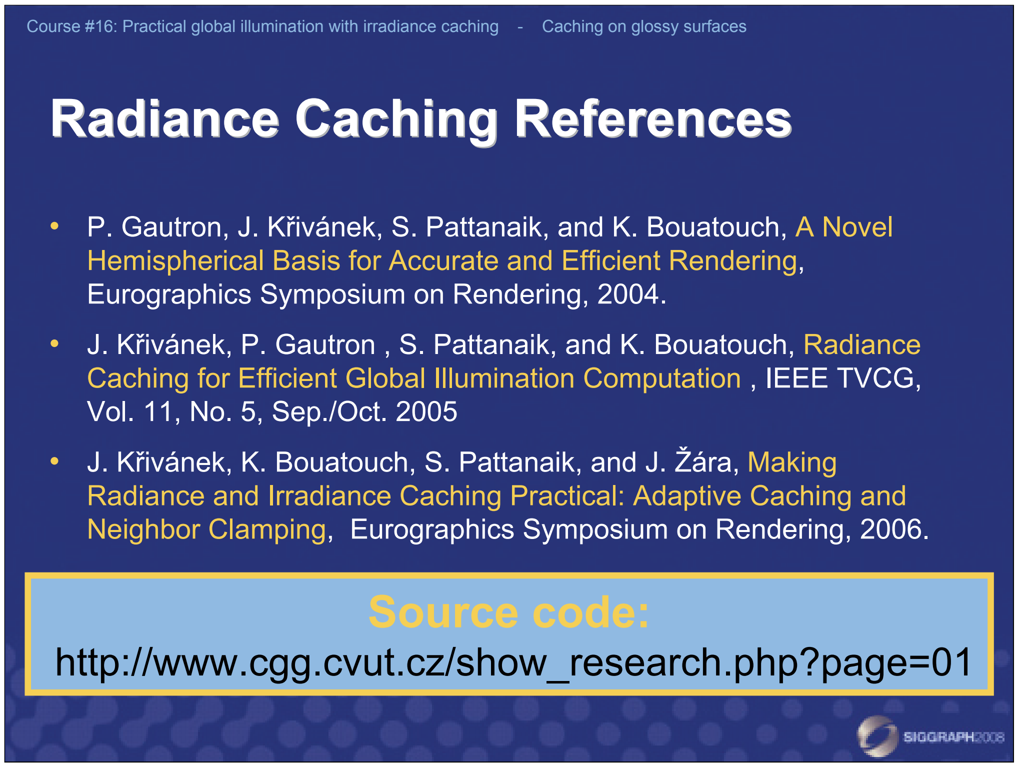
Their techniques is much lighter-weighted than radiance caching. It is an arbitrary approximation, but works fine in many cases.

Thank you

?



Source code:
http://www.cgg.cvut.cz/show_research.php?page=01



Course #16: Practical global illumination with irradiance caching - Caching on glossy surfaces

Radiance Caching References

- P. Gautron, J. Křivánek, S. Pattanaik, and K. Bouatouch, A Novel Hemispherical Basis for Accurate and Efficient Rendering, Eurographics Symposium on Rendering, 2004.
- J. Křivánek, P. Gautron , S. Pattanaik, and K. Bouatouch, Radiance Caching for Efficient Global Illumination Computation , IEEE TVCG, Vol. 11, No. 5, Sep./Oct. 2005
- J. Křivánek, K. Bouatouch, S. Pattanaik, and J. Žára, Making Radiance and Irradiance Caching Practical: Adaptive Caching and Neighbor Clamping, Eurographics Symposium on Rendering, 2006.

Source code:
http://www.cgg.cvut.cz/show_research.php?page=01

SIGGRAPH2008

Hardware Implementation

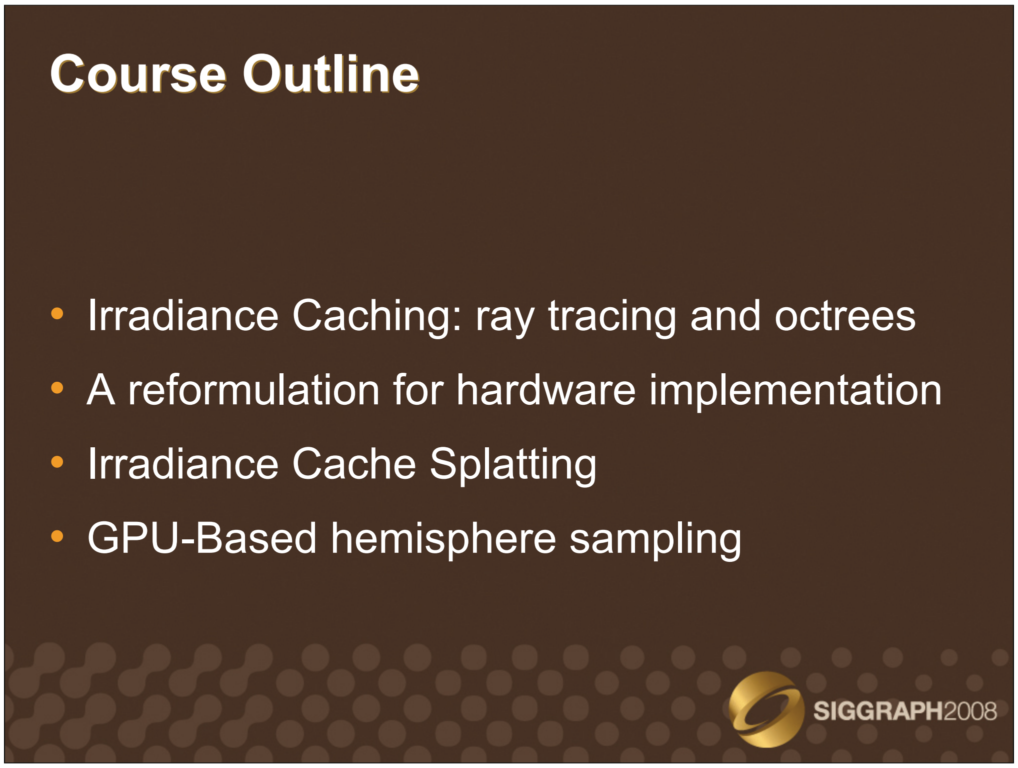
SIGGRAPH2008

Pascal Gautron

R&D Engineer

Thomson Corporate Research

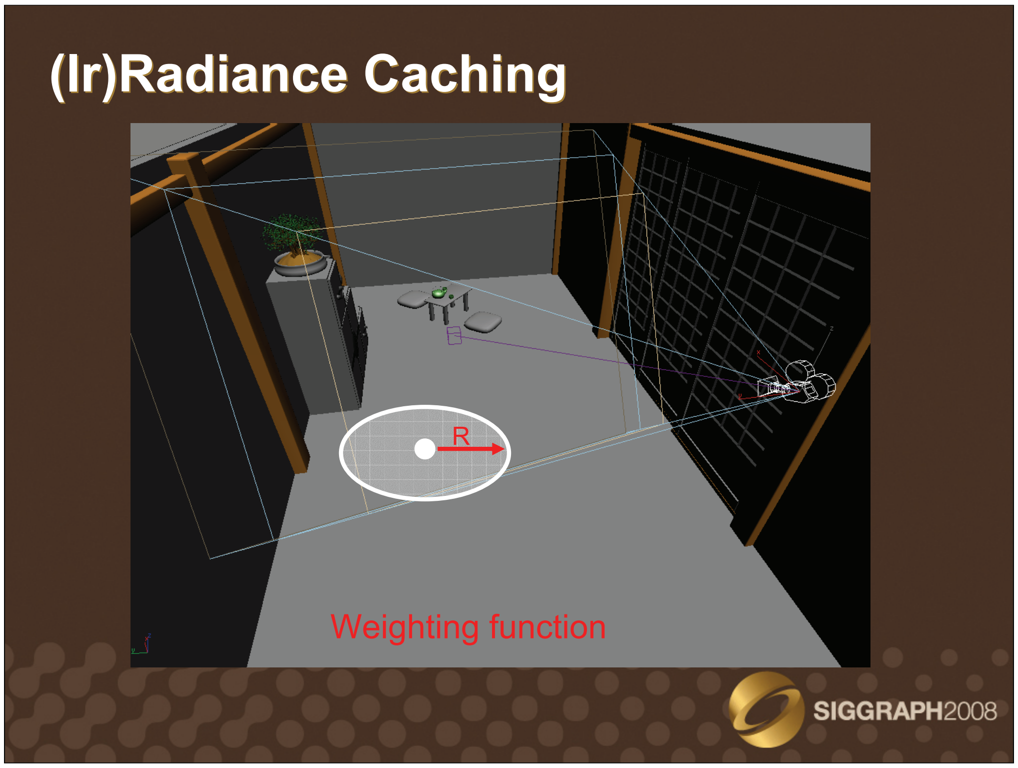
France



Course Outline

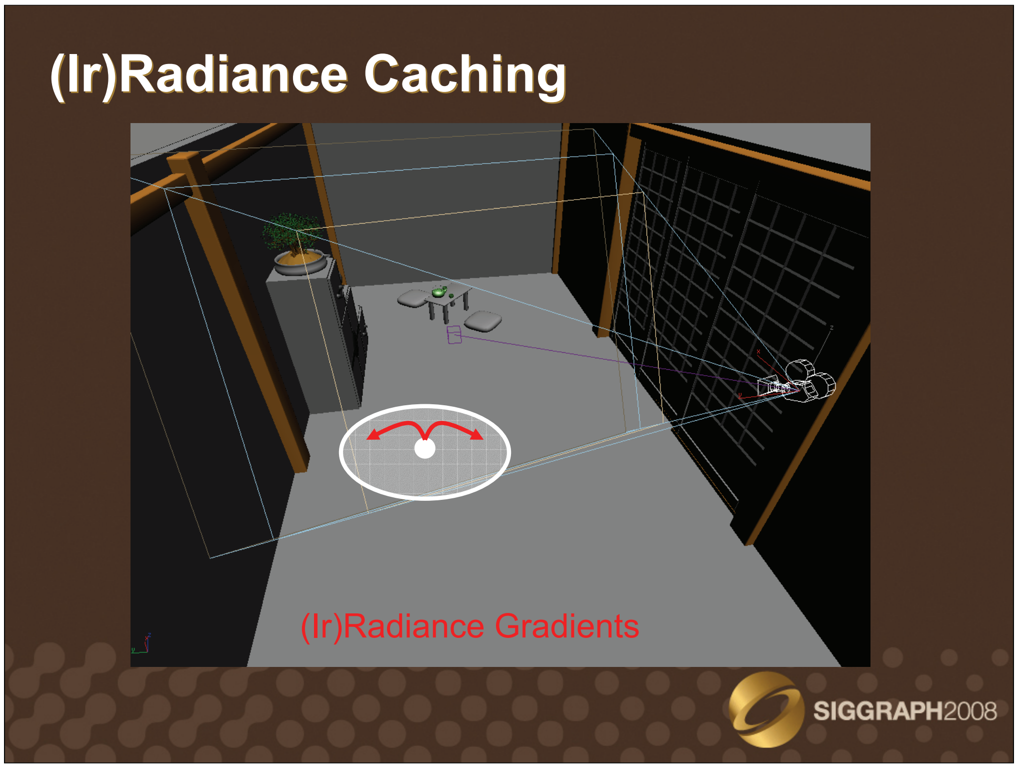
- Irradiance Caching: ray tracing and octrees
- A reformulation for hardware implementation
- Irradiance Cache Splatting
- GPU-Based hemisphere sampling

SIGGRAPH2008



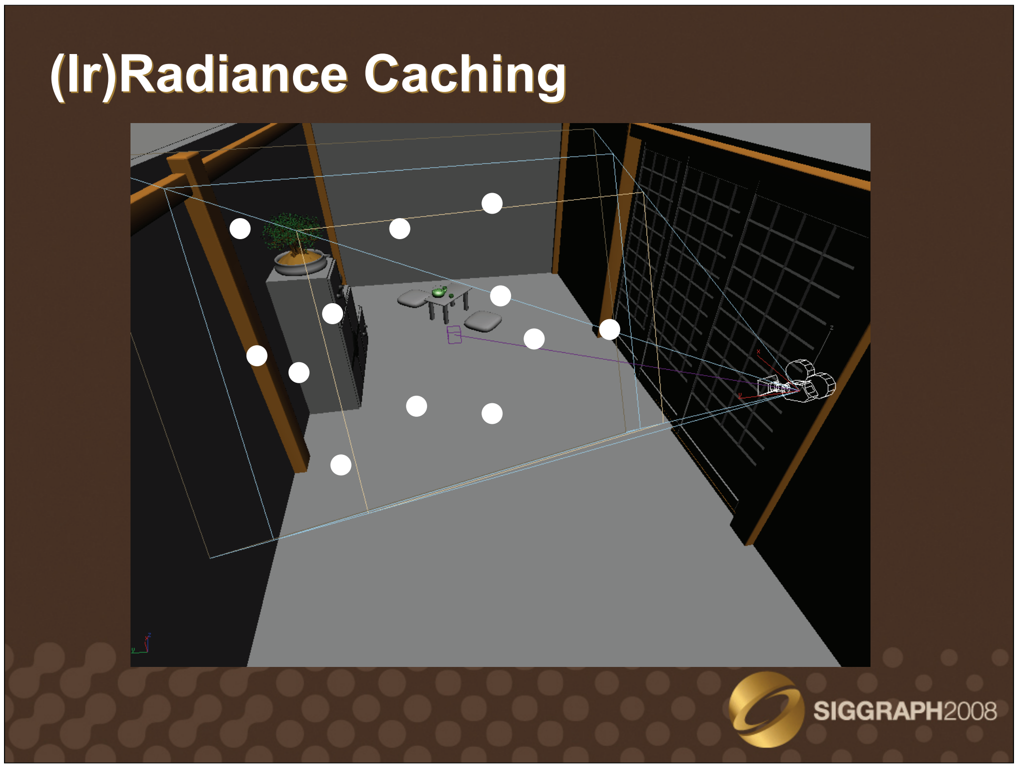
(Ir)Radiance Caching

The irradiance caching algorithm is based on sparse sampling and interpolation of indirect diffuse lighting at visible points. Each irradiance record contributes to the indirect lighting of points within its zone of influence. The size of this zone is adapted according to the mean distance R to the surrounding objects using the irradiance weighting function [Ward88].



(Ir)Radiance Caching

The irradiance value at points within the zone of influence of a record can be extrapolated using irradiance gradients [Ward92, Krivanek05a, Krivanek05b].



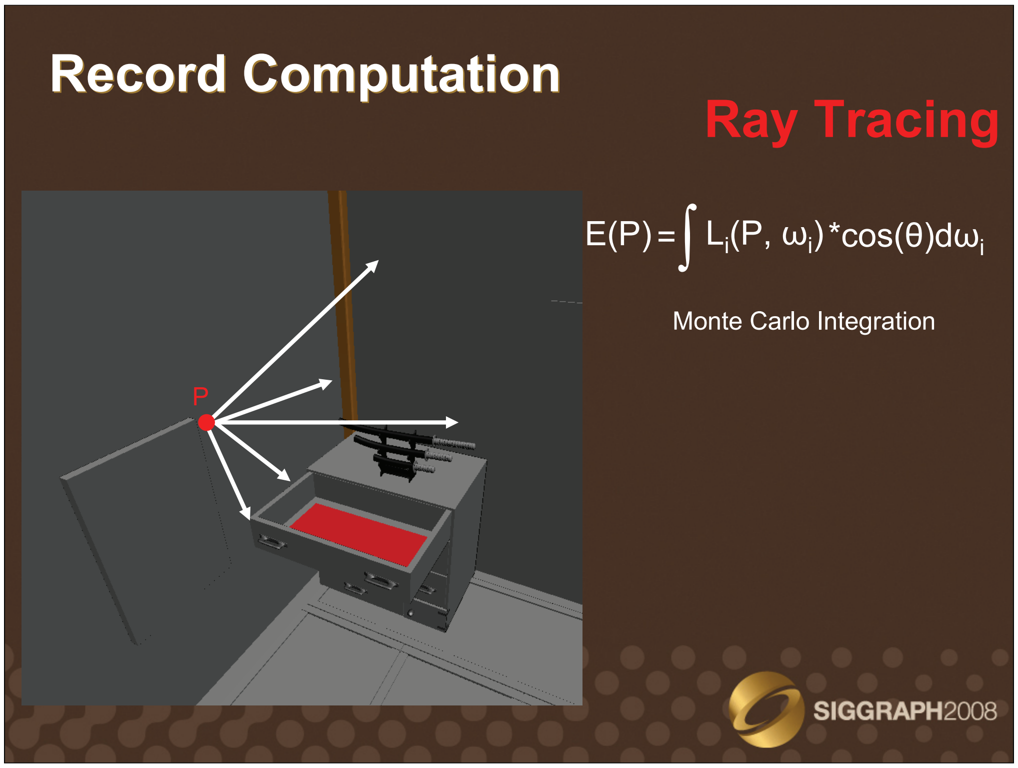
(Ir)Radiance Caching

When the zones of influence of the records cover the entire zone visible from the viewpoint, the image representing the indirect lighting can be rendered. The irradiance caching algorithm is then divided into three steps:

-The computation of the records

-The records storage

-The estimation of the indirect lighting using nearby records

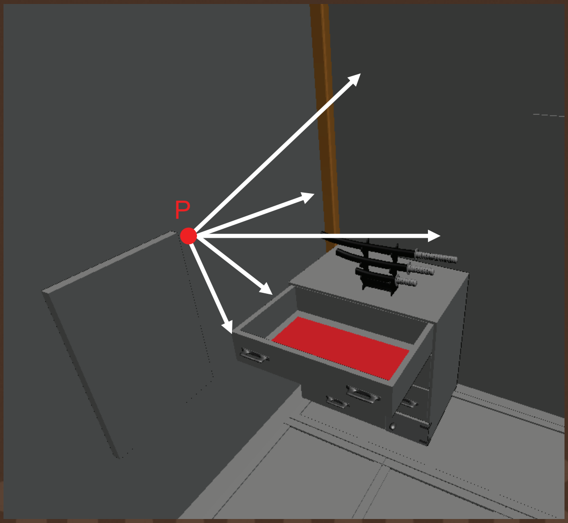


Record Computation

Ray Tracing

$$E(P) = \int L_i(P, \omega_i) * \cos(\theta) d\omega_i$$

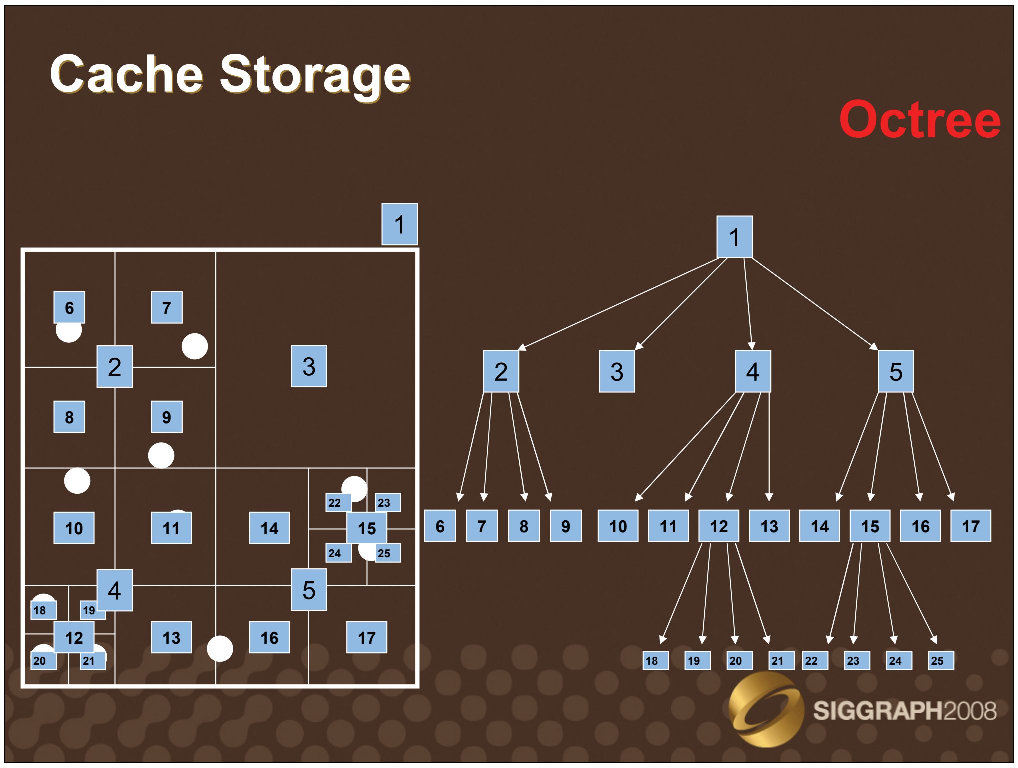
Monte Carlo Integration



SIGGRAPH2008

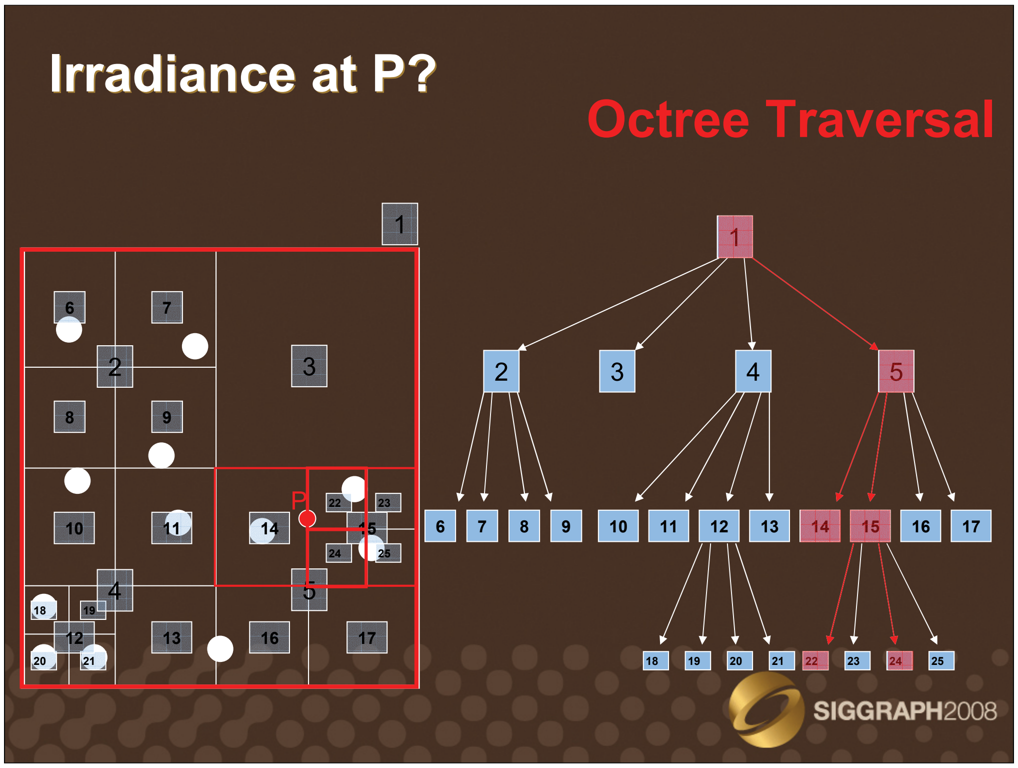
The computation of the irradiance value at a given point P requires the evaluation of the integral of the lighting over the surrounding hemisphere. This integral is typically estimated by Monte Carlo integration. Since the irradiance caching algorithm reuses the value of irradiance records over many pixels, the irradiance value of the record is computed with high precision, typically by tracing several hundreds to thousands rays.

Once the irradiance value is computed, a record is created. This record contains the corresponding position, normal, irradiance, gradients, and mean distance to the surrouding objects.



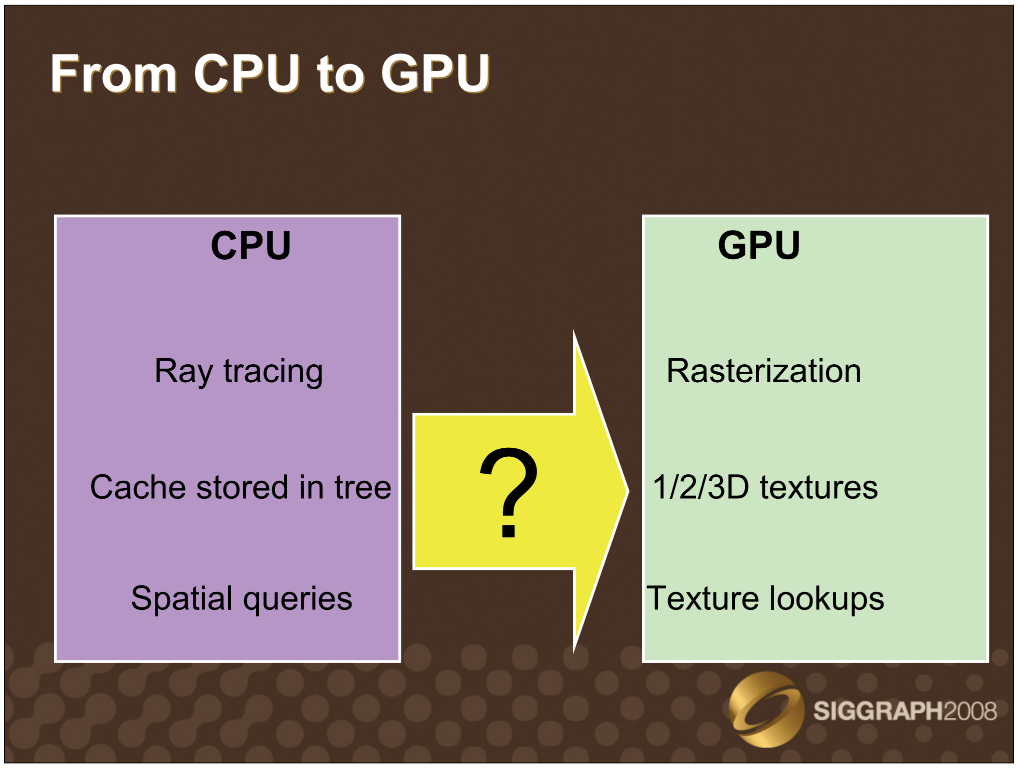
Cache Storage

The records are then stored in an octree, which allows for fast spatial queries. Note that the octree is a recursive data structure, which construction involves many conditional statements.


Irradiance at P?

Octree Traversal

The irradiance at a point P can then be estimated efficiently by querying the octree for nearby records. However, this involves a traversal of the structure, which also involves many conditional statements.



From CPU to GPU

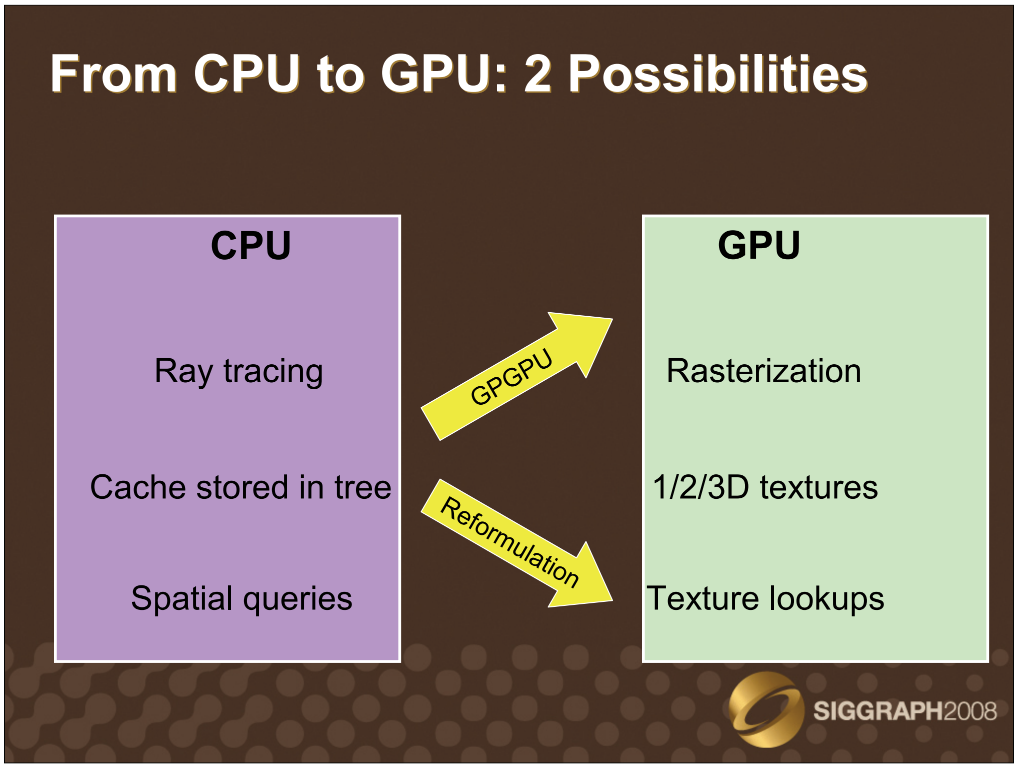
To summarize, the classical irradiance caching algorithm is implemented on the CPU, and is based on:

-Ray tracing

-Octrees

-Spatial queries in the octree

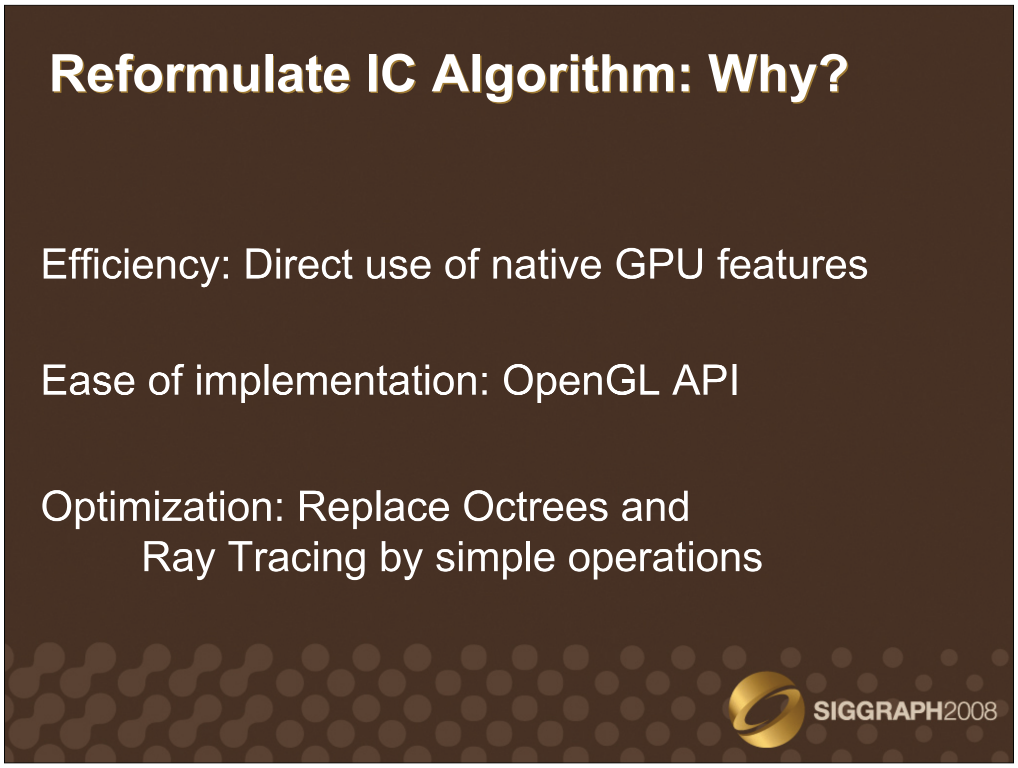
However, the GPU does not natively implement those operations: the visibility tests are performed using rasterization and Z-Buffering, and the only data structures available are 1, 2, and 3D textures. Particularly, the GPUs do not support pointers, which are the most common way of implementing recursive data structures.



From CPU to GPU: 2 Possibilities

The irradiance caching algorithm cannot be directly mapped to the GPU architecture. Two methods can be considered: first, the use of libraries for general purpose computations on the GPU (such as Brook for GPU, http://graphics.stanford.edu/projects/brookgpu/). An important amount of research work has been performed to achieve interactive ray tracing on GPUs (such as [Purcell02, Purcell03]). A kD-Tree implementation for GPUs has also been proposed [Foley05].

An other way of performing irradiance caching on graphics hardware is to reformulate the algorithm so that only the native features of the GPU are used. This would allow us to get the best performance out of the graphics processors.



Reformulate IC Algorithm: Why?

Efficiency: Direct use of native GPU features

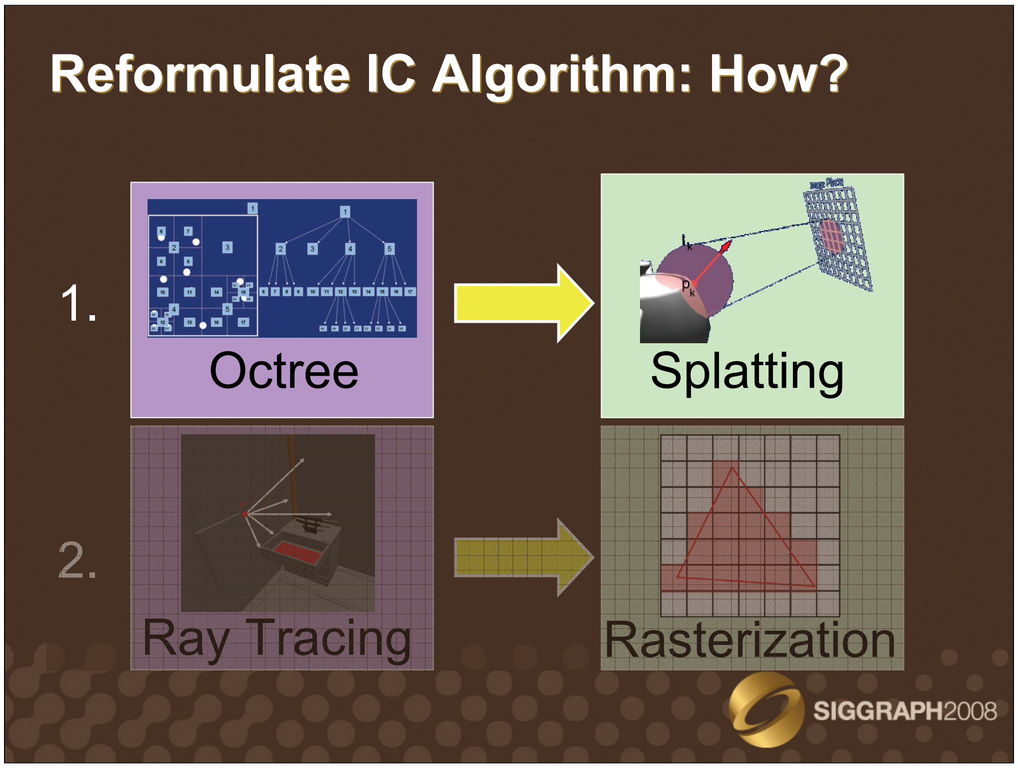
Ease of implementation: OpenGL API

Optimization: Replace Octrees and
Ray Tracing by simple operations

SIGGRAPH2008

More precisely, we chose to reformulate the irradiance caching algorithm for three reasons. First, the direct use of native GPU features allow us to use each part of the GPU at its best, improving the performance. Second, the reformulated algorithm can be implemented directly using at 3D graphics API such as OpenGL or DirectX.

Third, this reformulation gives us the occasion of attacking two costly aspects of the irradiance caching algorithm: the hierarchical data structure, and the irradiance computation using ray tracing. Replacing those by more simple operations on the GPU increases the performance, yielding interactive frame rates in simple scenes.

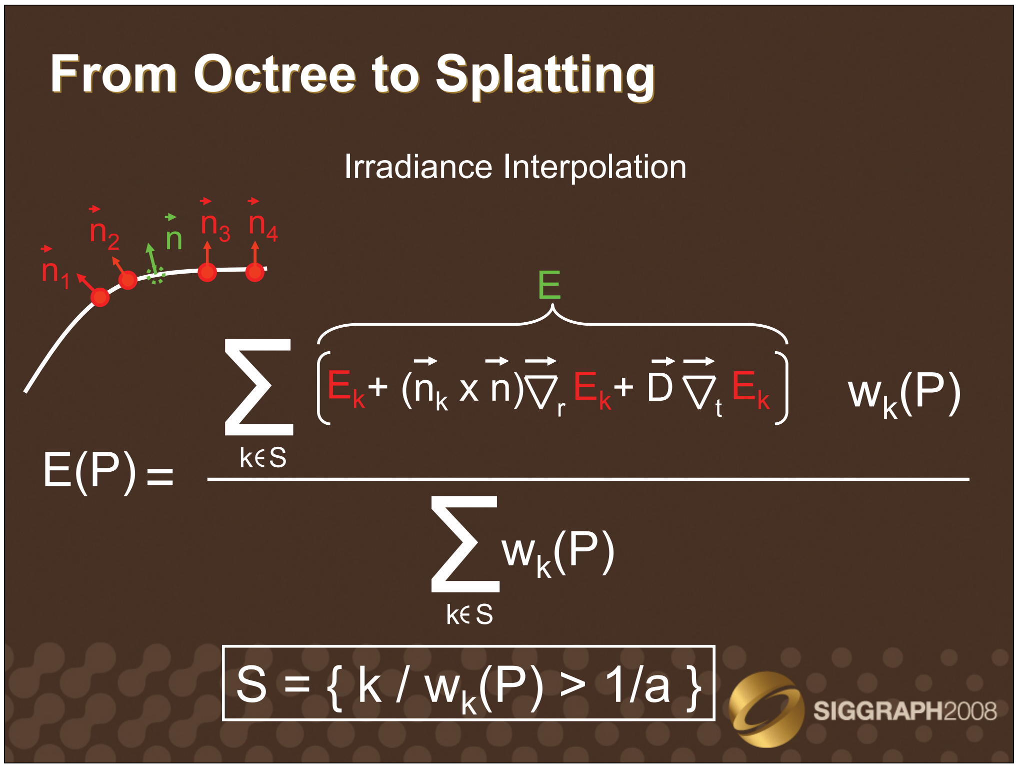


Reformulate IC Algorithm: How?

1. Octree → Splatting
2. Ray Tracing → Rasterization

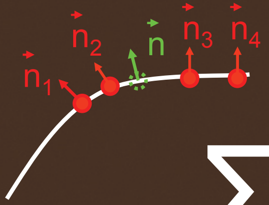
SIGGRAPH2008

The reformulation is divided into two tasks: the replacement of the octree by a splatting operation, and the use of rasterization in place of ray tracing.



From Octree to Splatting

Irradiance Interpolation

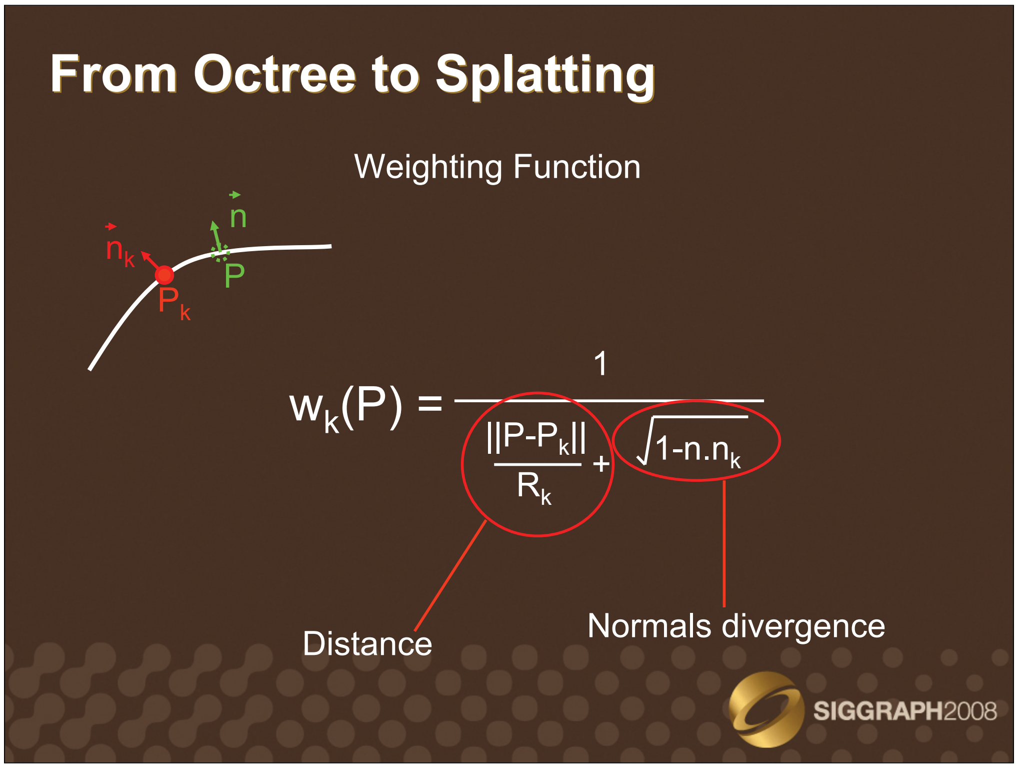


$$E(P) = \frac{\sum_{k \in S} \overbrace{\left(E_k + (\vec{n}_k \times \vec{n}) \vec{\nabla}_r E_k + \vec{D} \vec{\nabla}_t E_k \right)}^{E} \quad w_k(P)}{\sum_{k \in S} w_k(P)}$$

$$\boxed{S = \{ k \,/\, w_k(P) > 1/a \}}$$

SIGGRAPH2008

The reformulation is based on the irradiance interpolation equation [Ward88, Ward92] presented before: the irradiance estimate at a point P is the weighted average of the contributions of the surrounding records. The contribution of each record is computed using irradiance gradients for translation and rotation. The set S of contributing records is defined as the set of records for which the weigthing function evaluated at P is above a user-defined threshold a.



From Octree to Splatting

Weighting Function

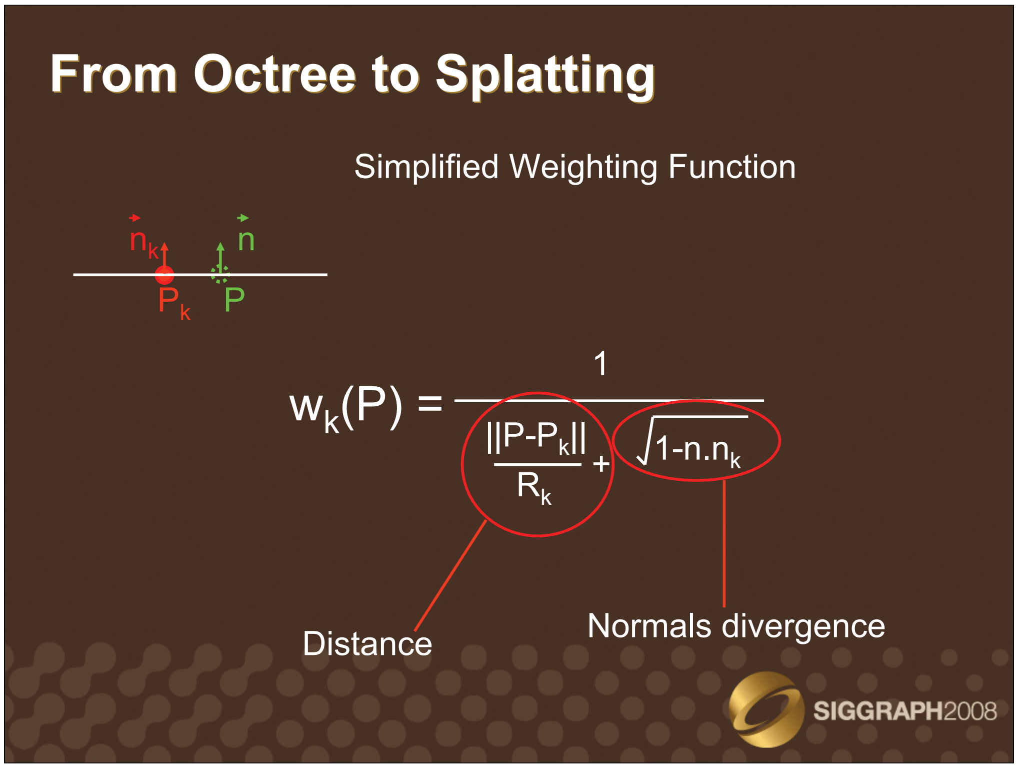
$$w_k(P) = \frac{1}{\frac{\|P-P_k\|}{R_k} + \sqrt{1-n.n_k}}$$

The weighting function is very simple, and depends on:

-The distance between the record location and the point P

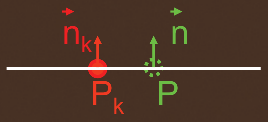
-The mean distance R to the surrounding objects

-The divergence of the surface normals between the record location and the point P



From Octree to Splatting

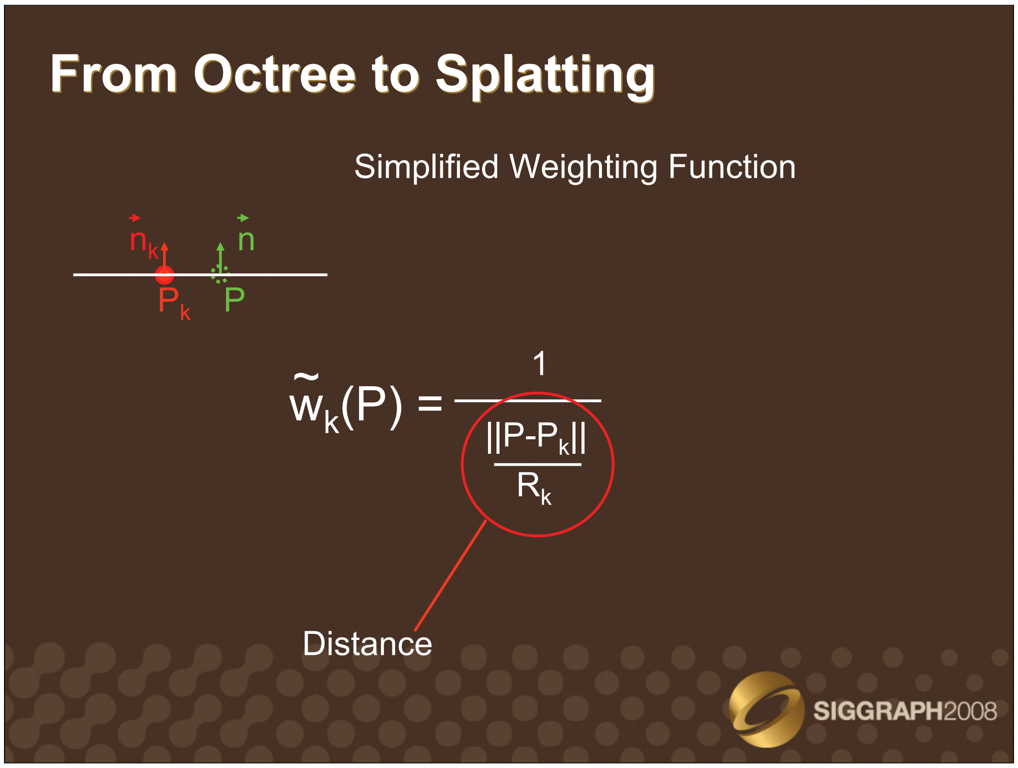
Simplified Weighting Function



$$w_k(P) = \frac{1}{\underbrace{\frac{\|P-P_k\|}{R_k}}_{\text{Distance}} + \underbrace{\sqrt{1-n.n_k}}_{\text{Normals divergence}}}$$

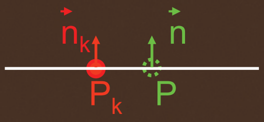
SIGGRAPH2008

If we assume that the record location and the point P **always have the same normal**, the weighting function can be simplified by removing the dependence to the surface normals.



From Octree to Splatting

Simplified Weighting Function



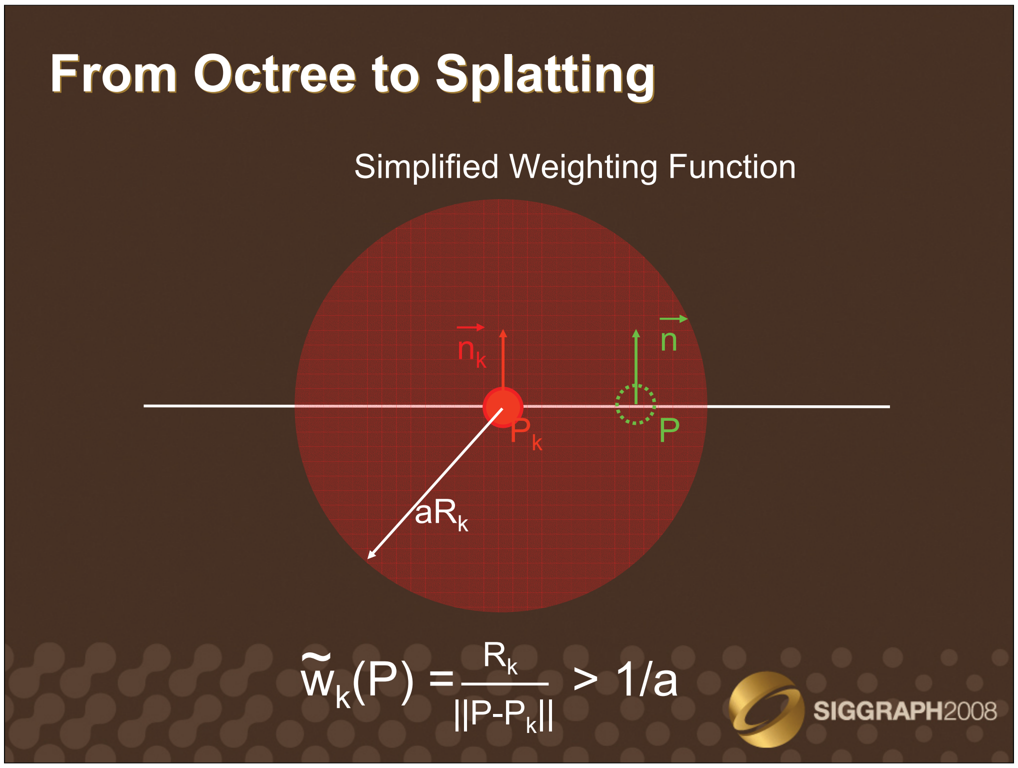
$$\tilde{w}_k(P) = \frac{1}{\boxed{\frac{\|P-P_k\|}{R_k}}}$$

Distance

This yields a simplified weighting function, which depends only on two factors:

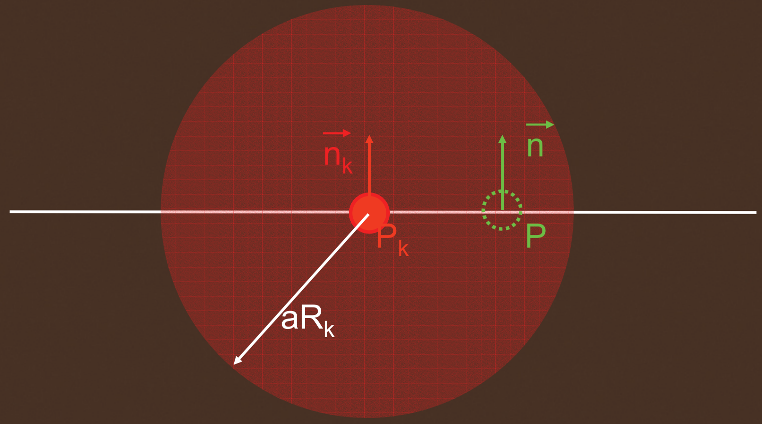
-The distance between the record location and the point P

-The mean distance to the surrounding objects



From Octree to Splatting

Simplified Weighting Function



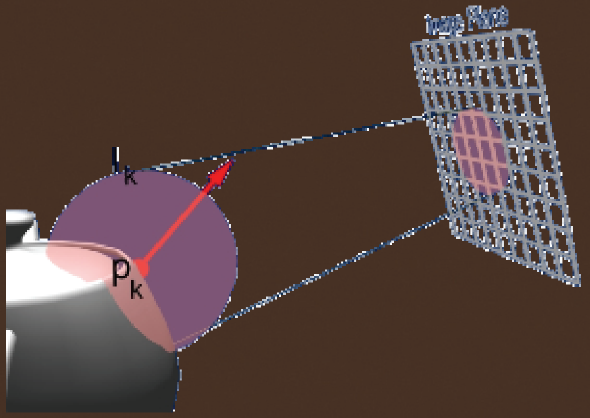
$$\tilde{w}_k(P) = \frac{R_k}{\|P-P_k\|} > 1/a$$

Using this simplified weighting function, a record k contributes to **all** points located within a sphere centered at the record location, with radius aRk. Hence this radius depends not only on the user-defined parameter a, but also on the mean distance to the surrounding objects.

Note that the simplified weighting function removes the constraint on the surface normals. Therefore, the set of points at which the record actually contributes (with respect to the full weighting function) is a subset of the points located within the sphere.

From Octree to Splatting

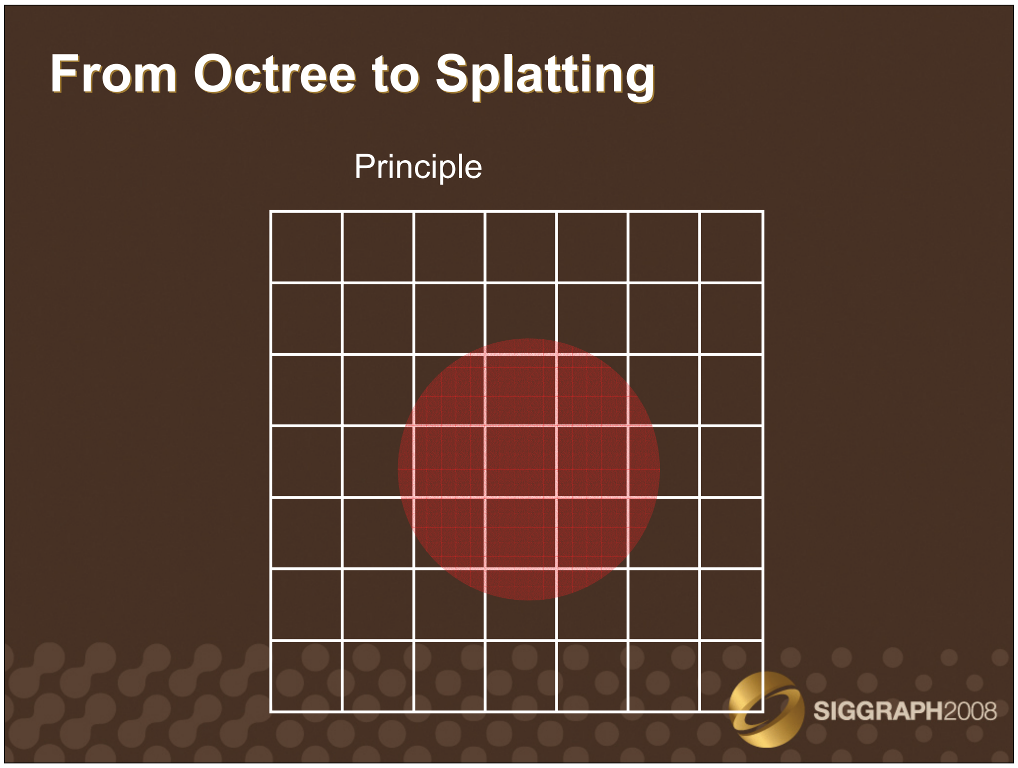
Principle



$$\tilde{w}_k(P) = \frac{R_k}{\|P-P_k\|} > 1/a$$

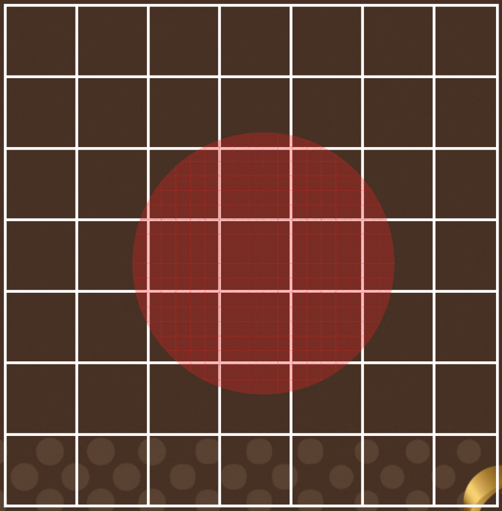
SIGGRAPH2008

The sphere can be splatted onto the image plane. Hence the covered pixels correspond to the visible points at which the record may contribute.

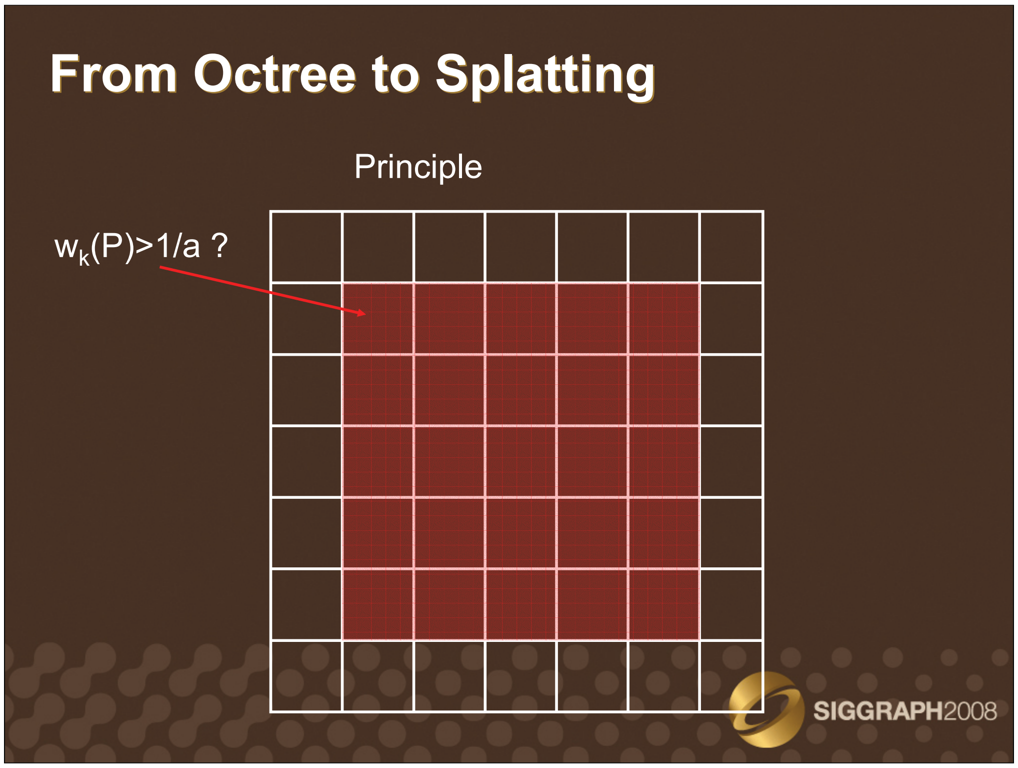


From Octree to Splatting

Principle

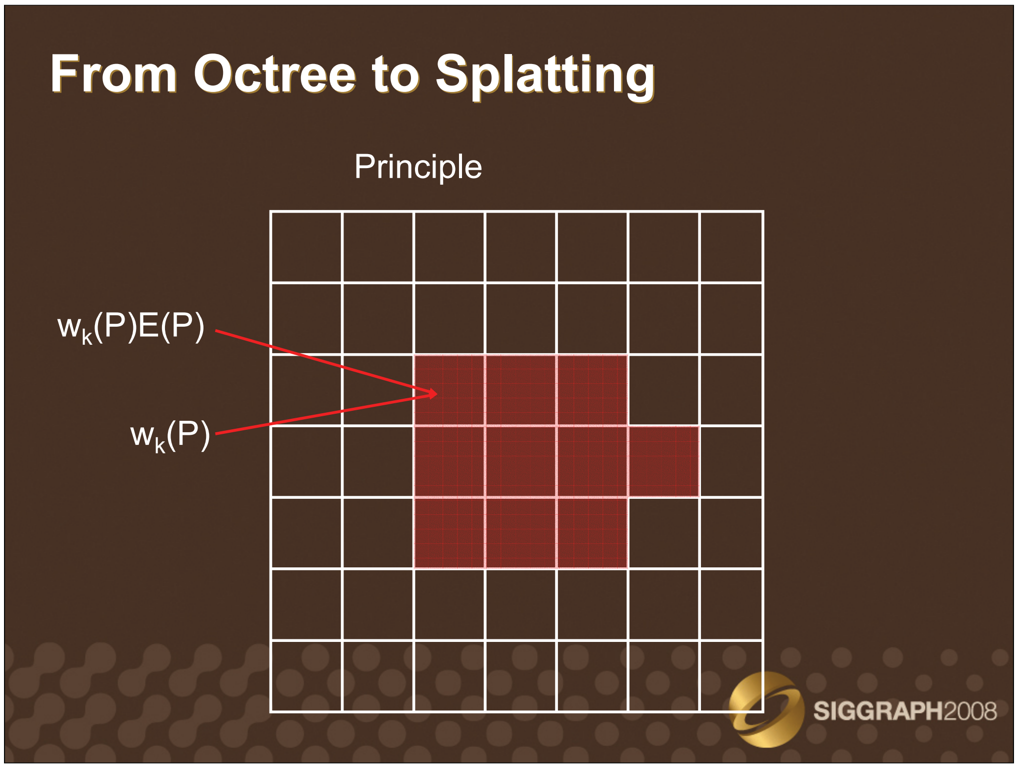


Let us consider the image plane, on which the sphere has been splatted. The splatted sphere encloses **all** the visible points at which the considered record may contribute (with respect to the full weighting function). Our goal is now to select the points for which the condition on the **full** weighting function is satisfied, that is $w_k(P) > 1/a$.



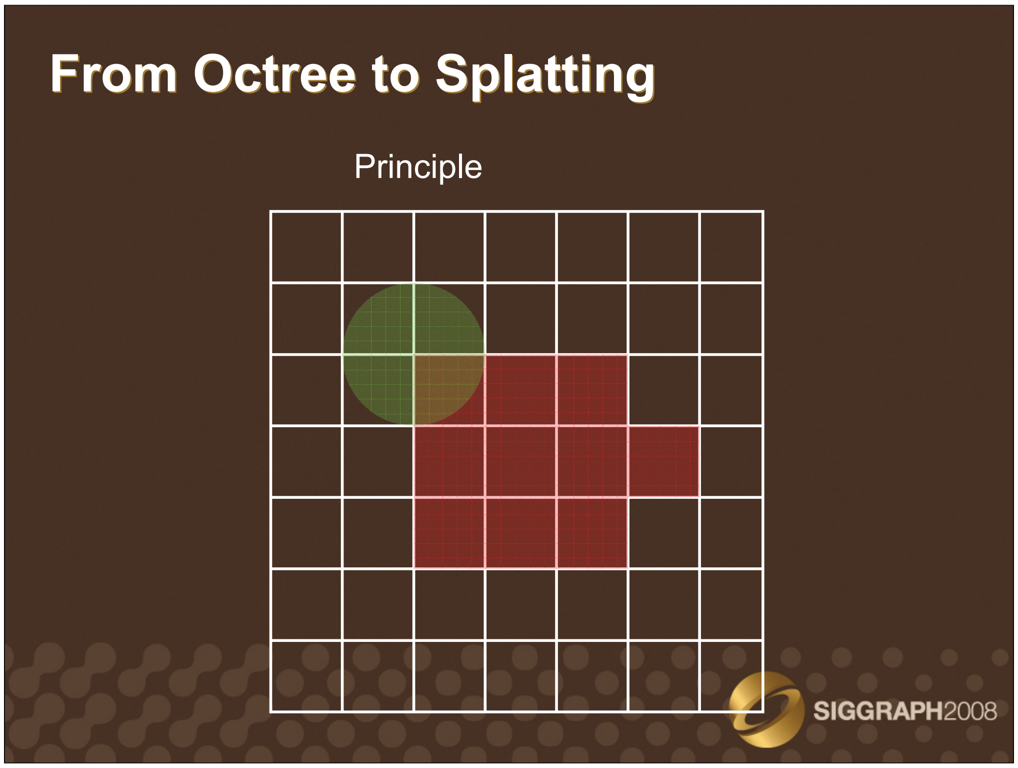
From Octree to Splatting

For the convenience of implementation, we use a quadrilateral tightly enclosing the splatted sphere. For each point visible through the pixels of the quadrilateral, the full weighting function is evaluated, and tested against the user-defined threshold. If the condition is not satisfied, the pixel is discarded.



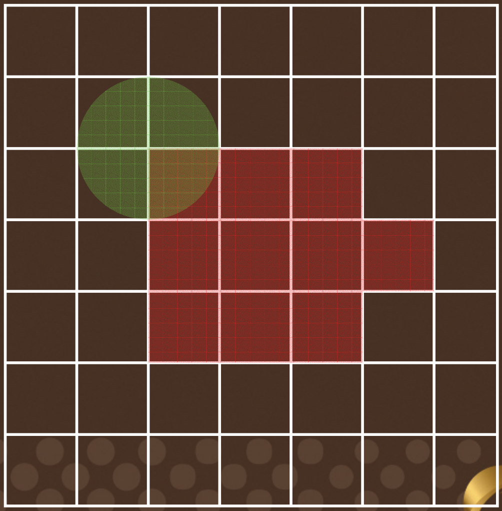
From Octree to Splatting

This yields a set of pixels, corresponding to the set of visible points at which the record actually contributes. At those points, we compute separately the weighted contribution of the record (with respect to the full weighting function and to the irradiance gradients) and the weight of the contribution. This information can be easily stored within floating point RGBA pixels, using RGB for the weighted contribution, and the alpha channel for the weight value.

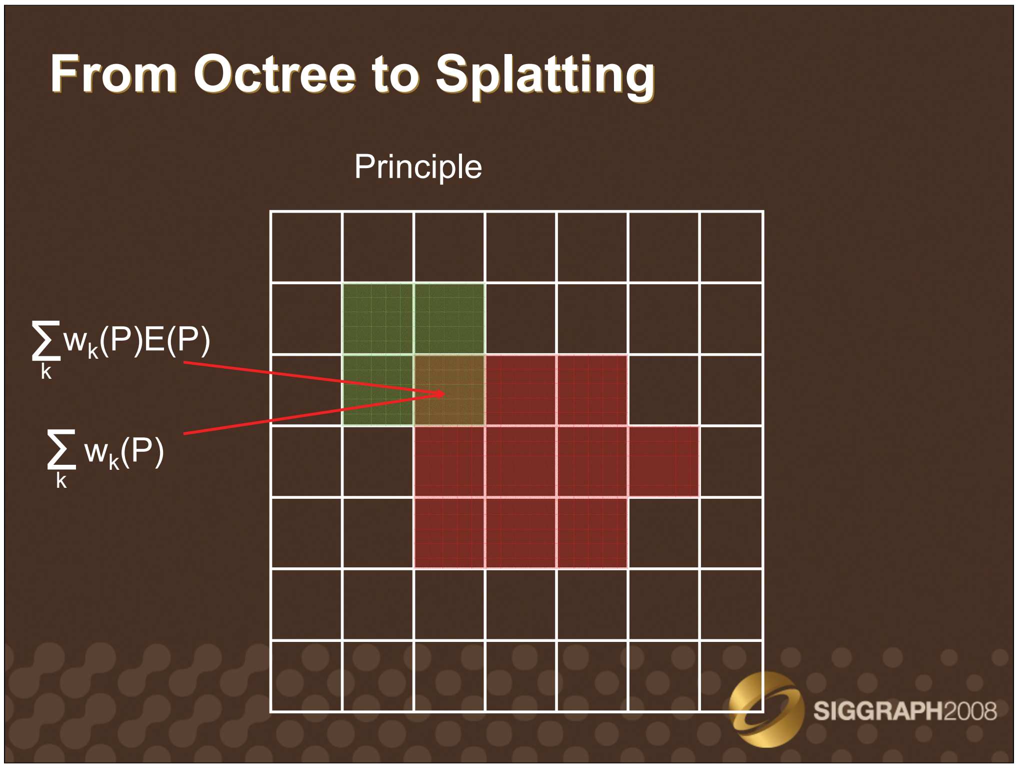


From Octree to Splatting

Principle

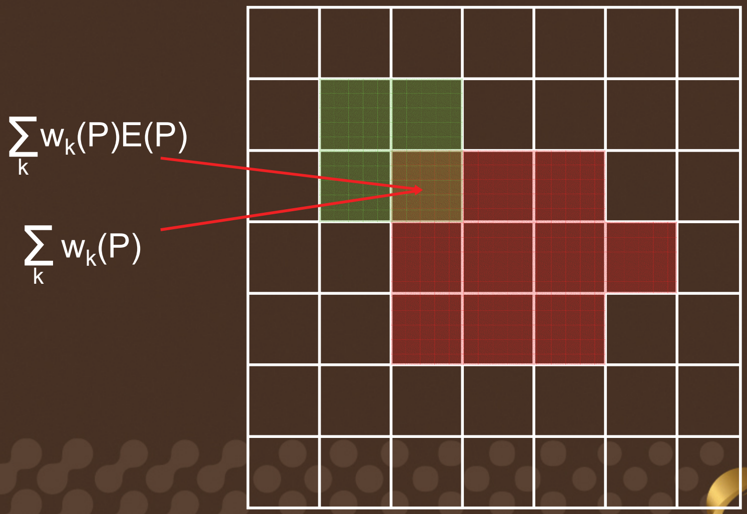


When splatting an other record, the same operations are to be done. However, the zones of influence of the two records overlap.



From Octree to Splatting

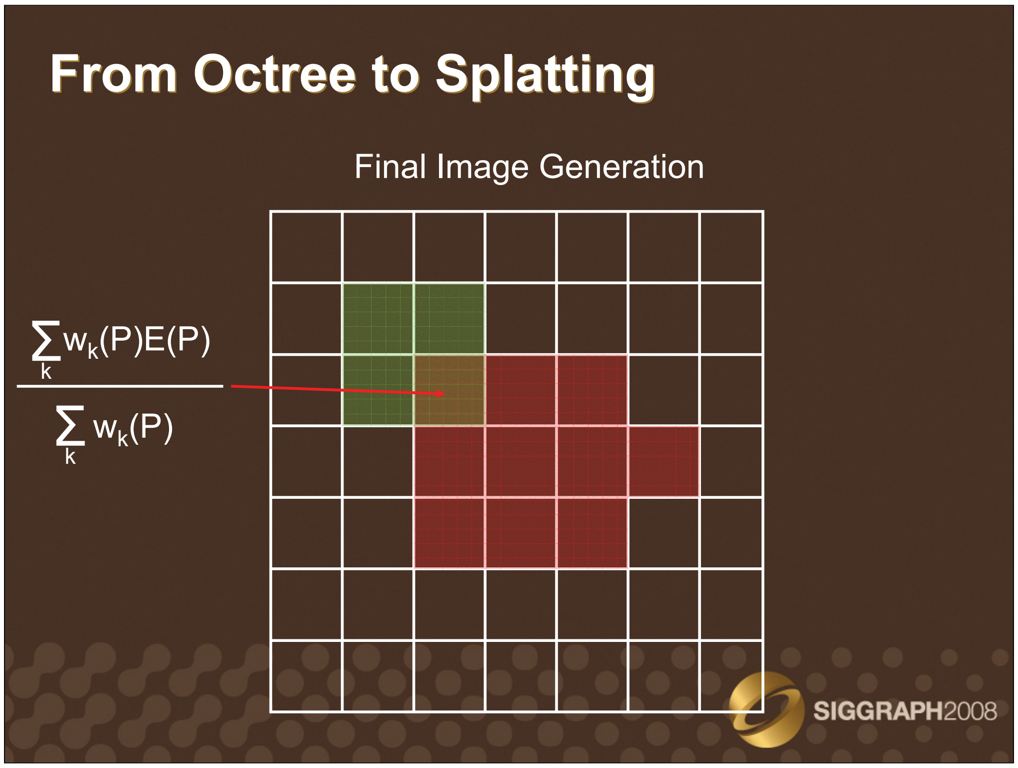
Principle



SIGGRAPH2008

In this case, we first compute the weighted contribution and weight of the second record as described before. Then, the built-in alpha blending of graphics processors adds the contributions and weights together in the overlapping area.

Then, each pixel contains both the weighted sum of the contributions, and the sum of the contribution weights.

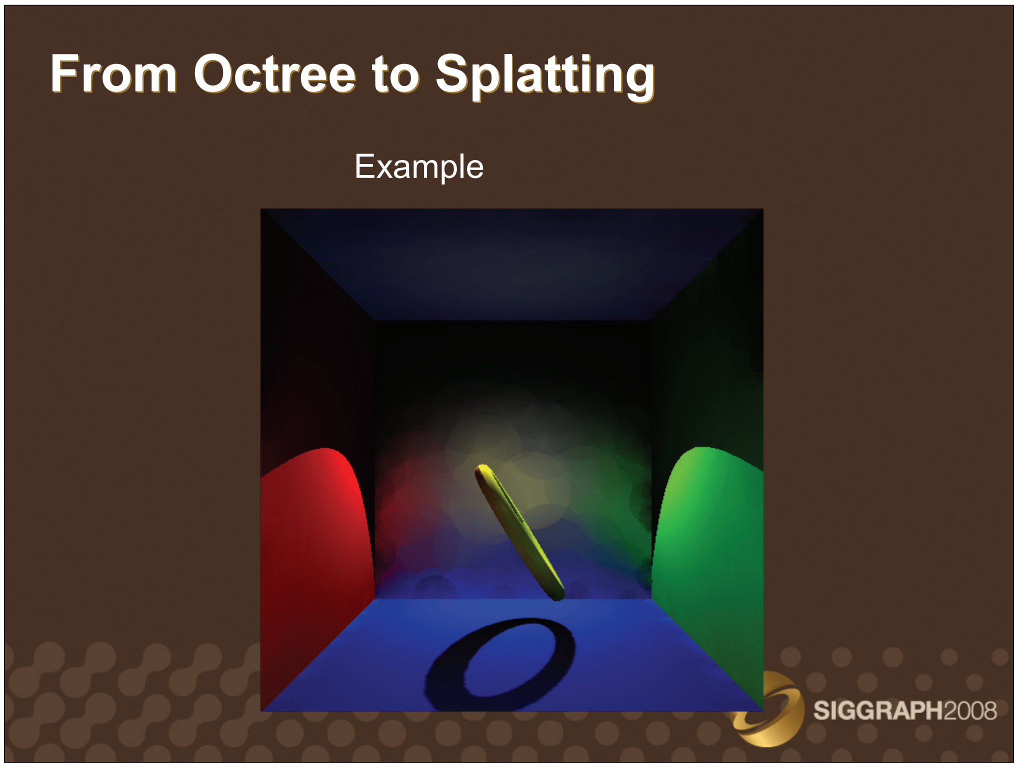


From Octree to Splatting

Final Image Generation

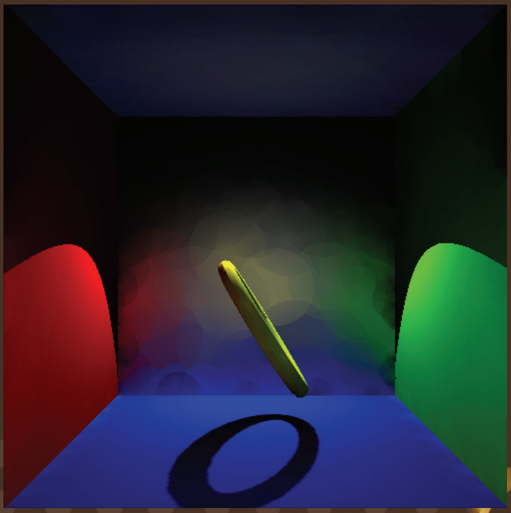
$$\frac{\sum_k w_k(P)E(P)}{\sum_k w_k(P)}$$

Once all the necessary records have been splatted, the result of the irradiance interpolation equation can then be obtained by dividing the weighted sum of contributions by the sum of the weights. This operation can be easily performed within a fragment shader, by dividing the RGB components (that is, the weighted sum of contributions) by the alpha channel (the sum of weights).



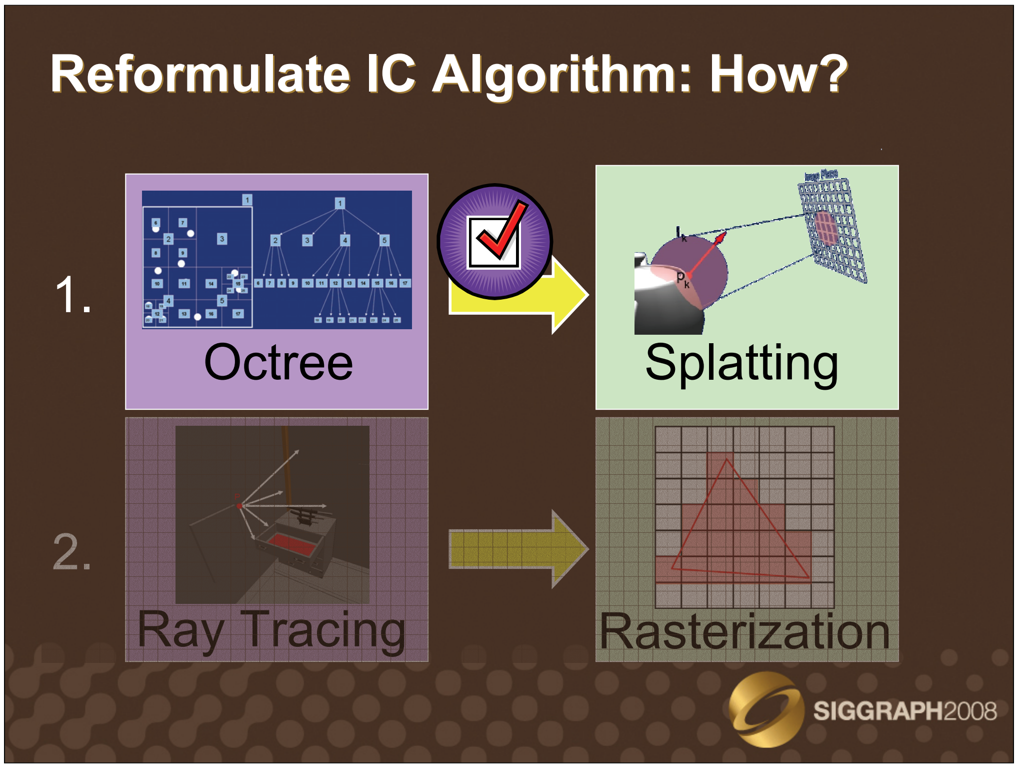
From Octree to Splatting

Example

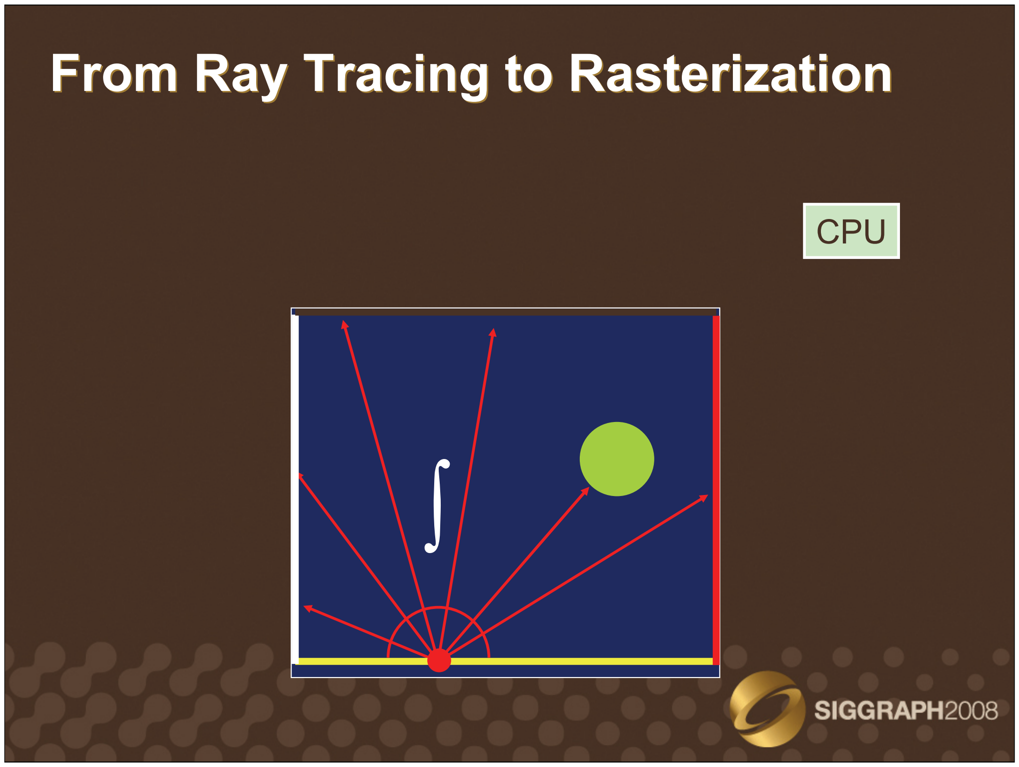


To summarize, let us consider an example. At the beginning of the algorithm, the cache contains no records. Hence the generated image only features direct illumination. Then, we add a record on the back wall. This record contributes to the points within its neighborhood. When adding a second record, the zones of influence overlap. For the pixels within the overlapping area, the estimated irradiance is calculated using the weighted average of the contributions. Other records are successively added to the cache, until the entire visible area of the scene is covered by the zones of influence of the records. The resulting image features both direct and indirect lighting for every visible point.

Note that for explanation purposes, the reconstruction of the indirect lighting is very coarse to highlight the zones of influence of each record. Also, the gradients are not used. As with the classical irradiance caching algorithm, high quality can be obtained by using irradiance gradients and setting an appropriate value for the user-defined parameter a.

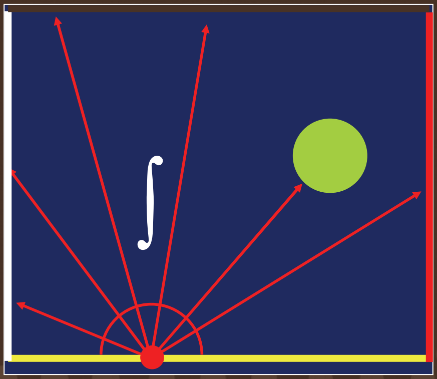


Reformulate IC Algorithm: How?



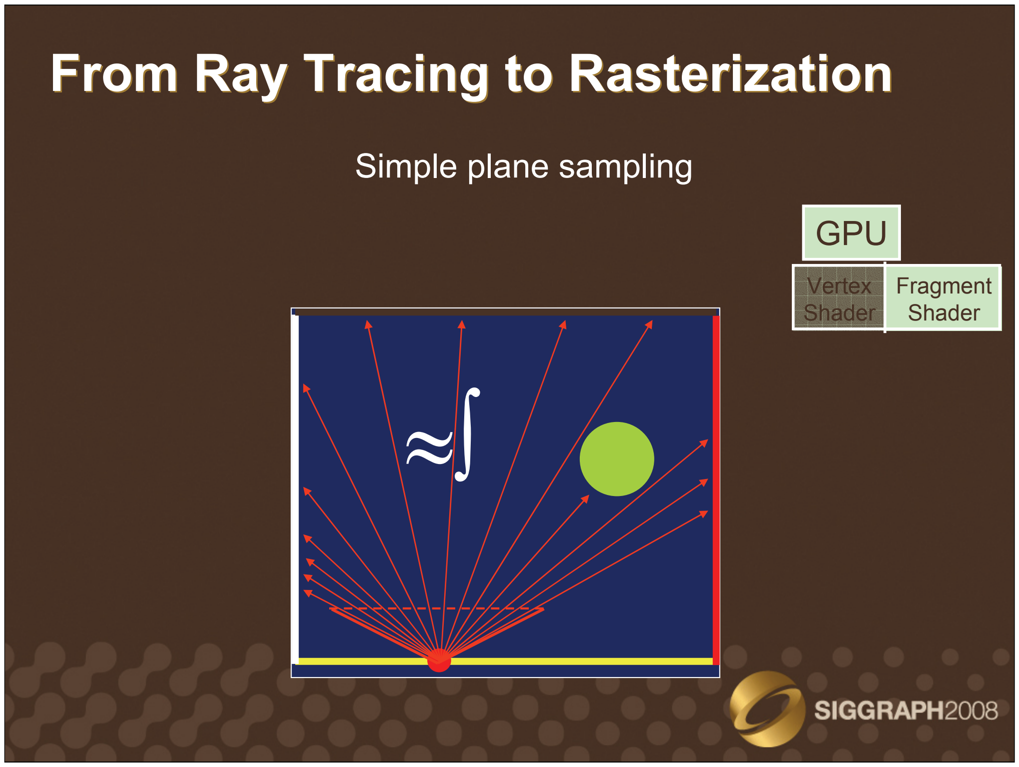
From Ray Tracing to Rasterization

CPU



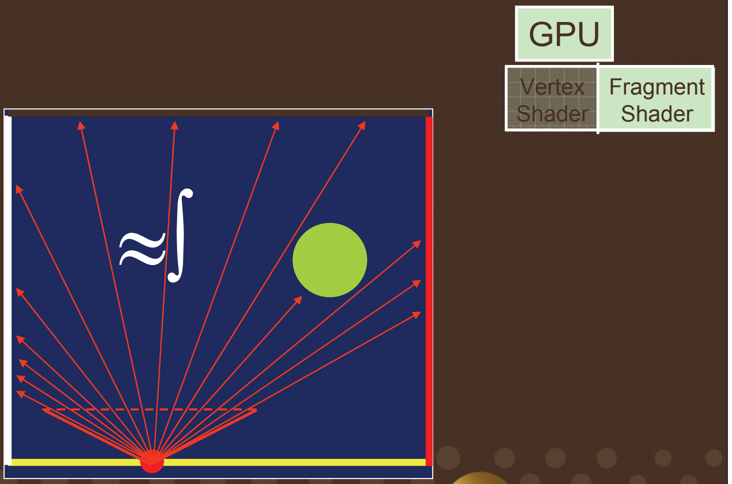
SIGGRAPH2008

In classical irradiance caching, the irradiance at a point is estimated by Monte Carlo ray tracing: random rays are traced through the scene, gathering the lighting incoming from the surrounding environment. This estimate is usually computed on the CPU.



From Ray Tracing to Rasterization

Simple plane sampling



SIGGRAPH2008

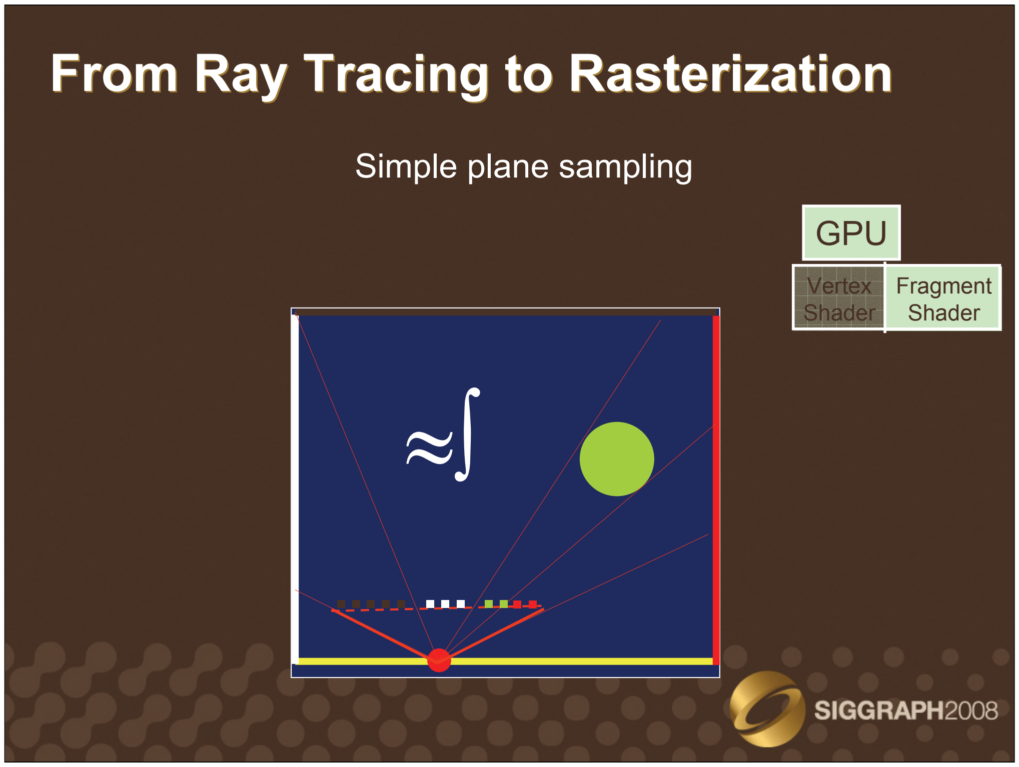
A well-known approximate method for hemisphere sampling on the GPU is the simple plane sampling: a virtual camera with a large aperture is placed at the point of interest. The scene is then rendered on graphics hardware, yielding an image of the surrounding objects. On recent graphics hardware, this data can be computed in high dynamic range (HDR) using floating-point render target and programmable shaders. The shadowing effects can be efficiently accounted for using fast shadowing techniques such as shadow mapping [Wil78].

The solid angle subtended by a pixel p is defined as Ω_p=A*cos(θ)/(d*d) where:

•A is the surface of a pixel

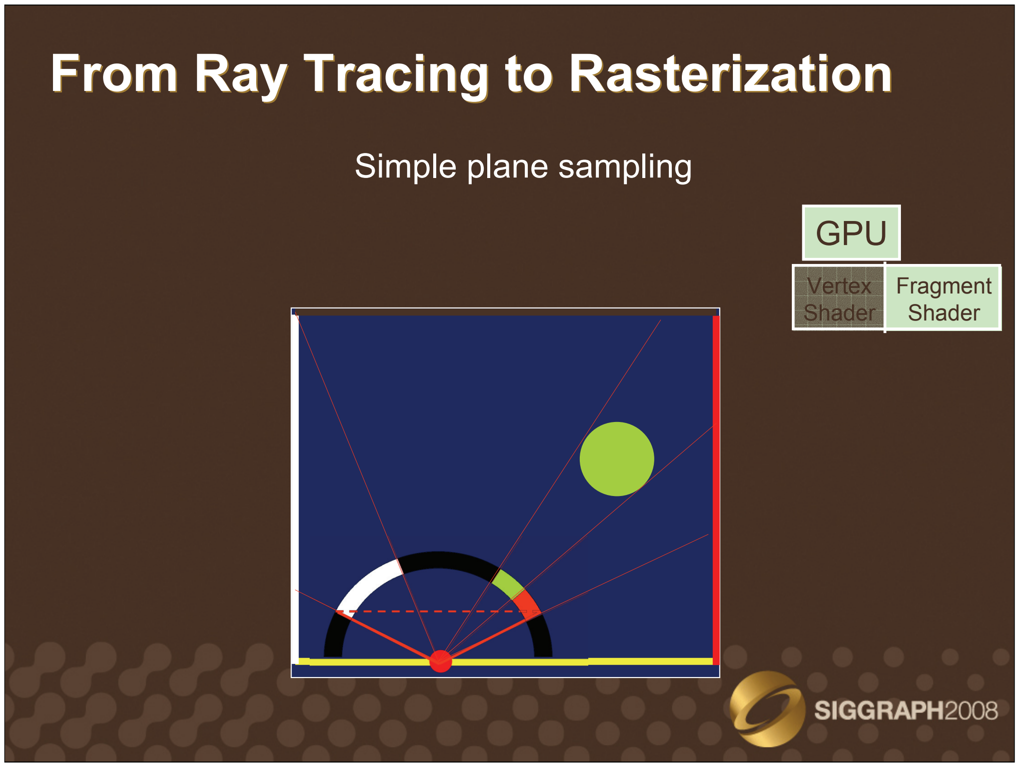
•θ is the angle between the direction passing through the pixel and the surface normal

•d is the distance between the point of interest and the image plane



From Ray Tracing to Rasterization

Simple plane sampling

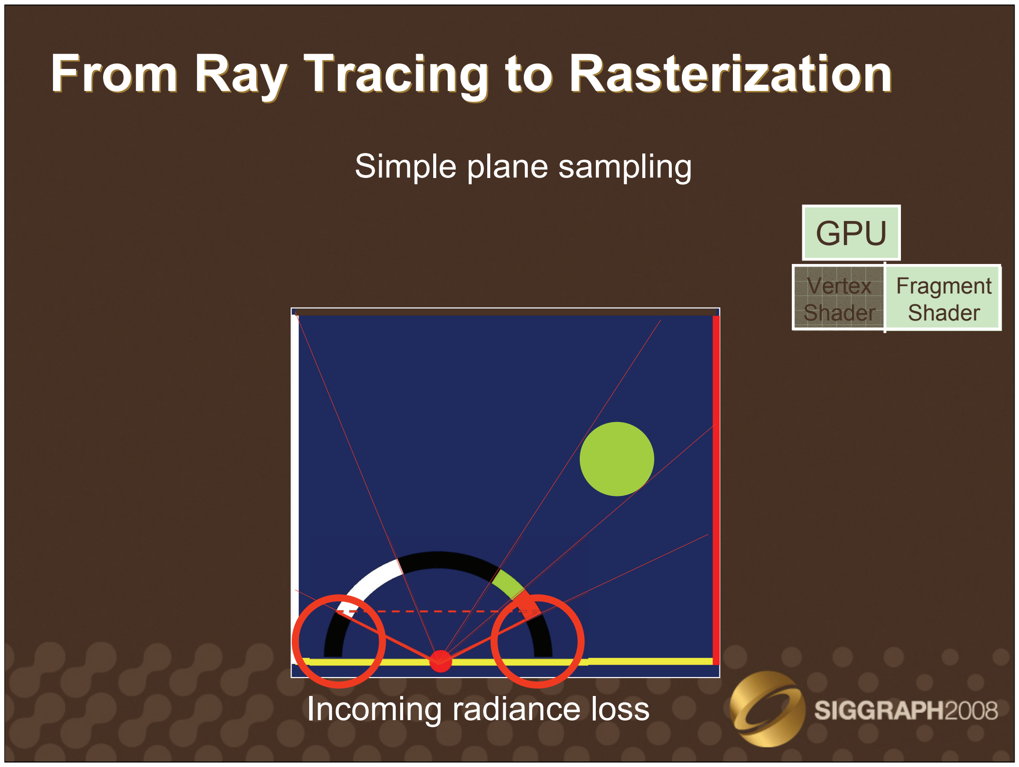


From Ray Tracing to Rasterization

Simple plane sampling

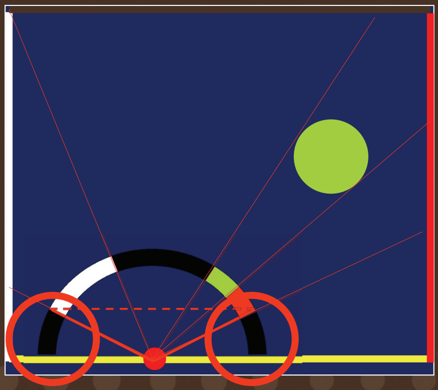
However, the aperture of the camera cannot allow us to sample the entire hemisphere: a perspective camera is represented by a perspective projection matrix which is applied to the visible contents of the scene. In OpenGL, such a camera is modeled using the gluPerspective function, whose values are calculated using the field of view (FOV) of the camera. More precisely, a key value in this matrix is f = cot(FOV/2), which is undefined for FOV=180°. Furthermore, using a very large aperture such as 179.999° leads to important perspective deformation and sampling problems.

In [LC04], Larsen et al. show that an aperture of 126.87° is sufficient for capturing 86% of the incoming directions. However, the remaining 14% are unknown and must be compensated to avoid a systematic underestimation of the incoming radiance. Furthermore, this compensation must respect the directional information of the incoming radiance to allow for a later implementation of radiance caching for global illumination computation on glossy surfaces.

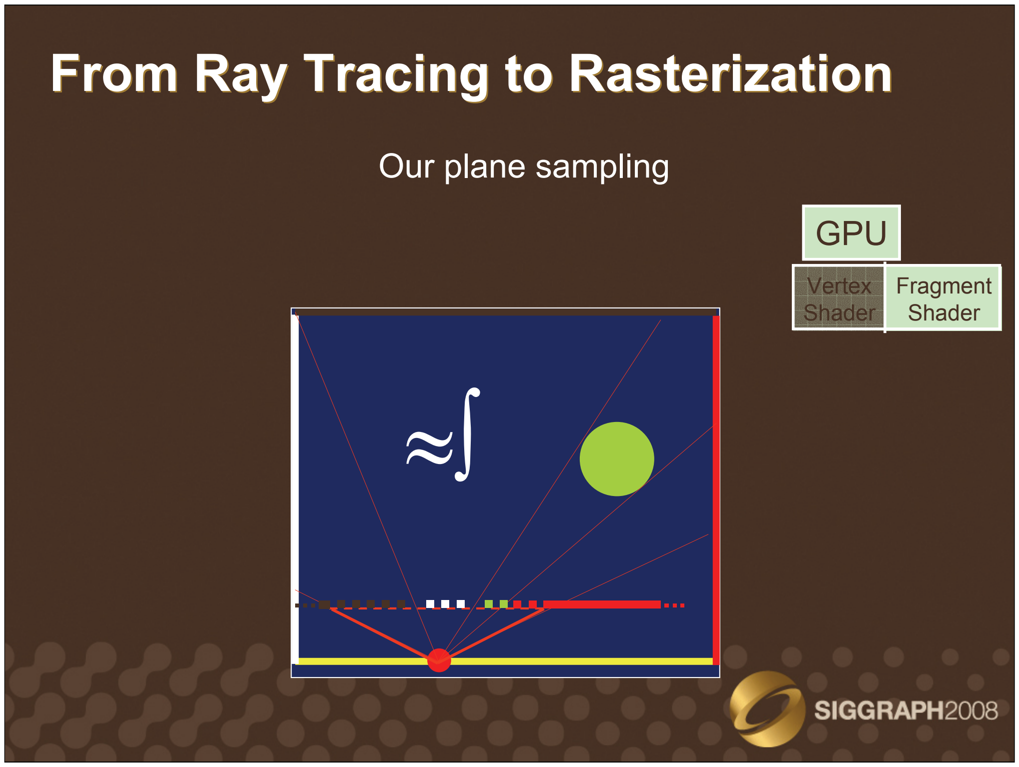


From Ray Tracing to Rasterization

Simple plane sampling



Incoming radiance loss



From Ray Tracing to Rasterization

Our plane sampling

We propose a very simple compensation method, in which the border pixels are virtually « extended » to fill the parts of the hemisphere which have not been actually sampled. To this end, border pixels are considered as covering a solid angle of:

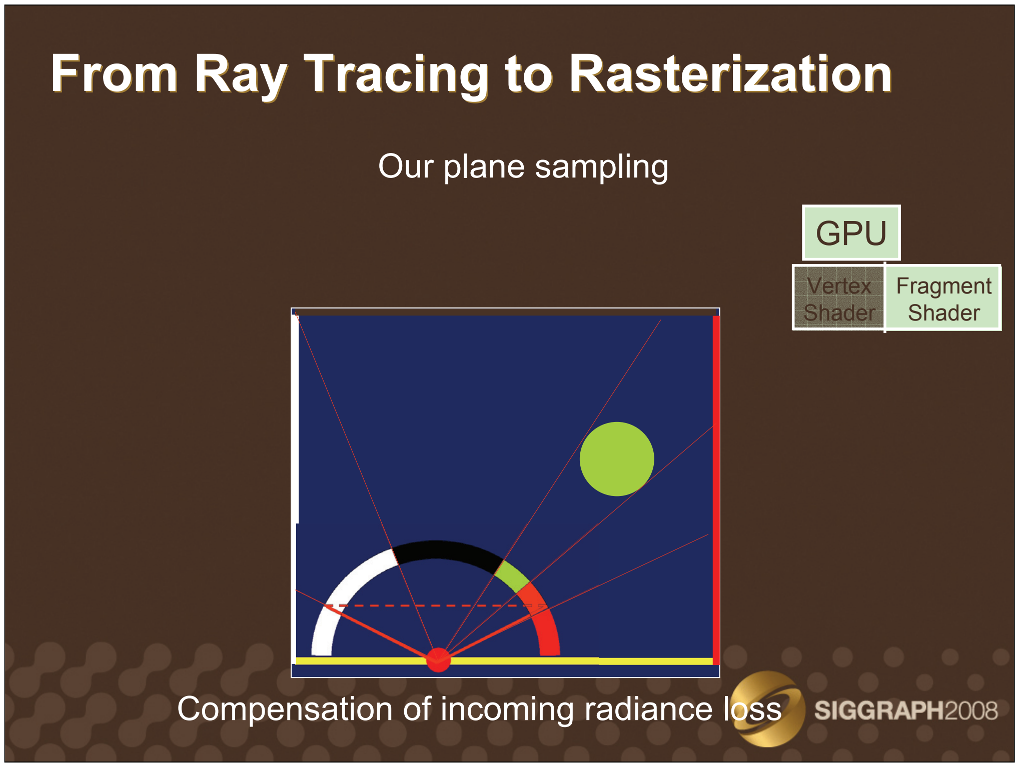
$\Omega_{border} = \Omega_p + \cos(\theta_{border}) * \delta_\Phi$

Where:

•Ω_p is the solid angle subtended by the pixel

•θ_{border} is the largest θ covered by the aperture of the camera

•δ_Φ is the interval of Φ spanned by the pixel



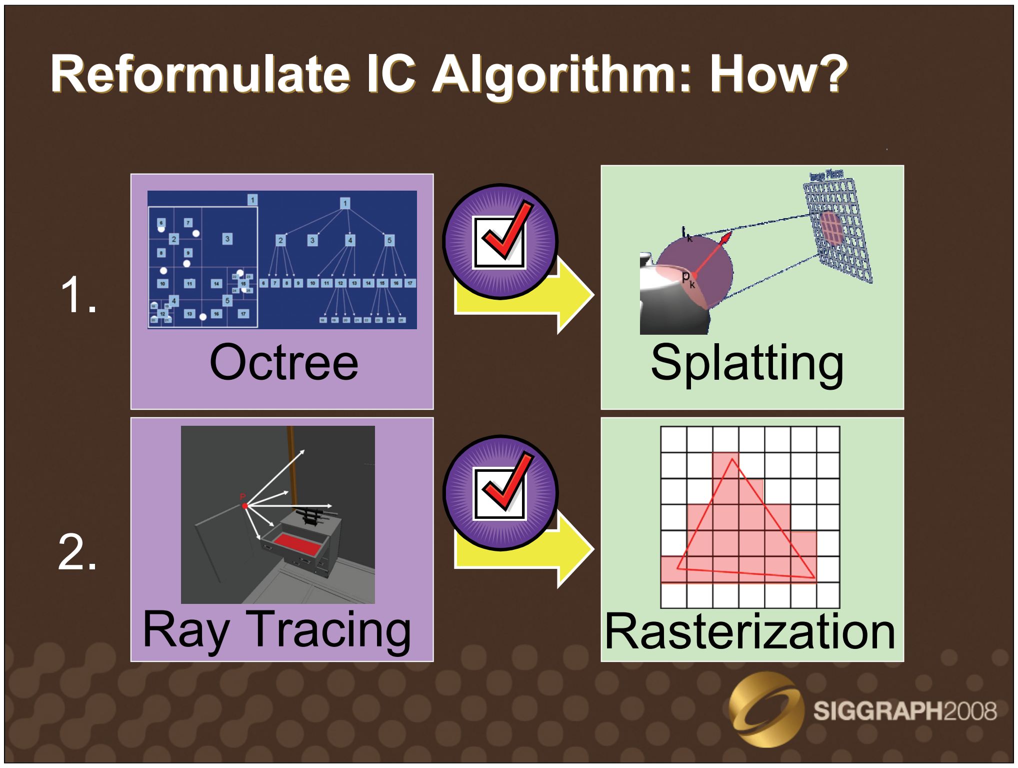
From Ray Tracing to Rasterization

Our plane sampling

Compensation of incoming radiance loss

This allows us to compensate the missing information by extrapolating the radiance values at the extremities of the sampling plane. This extrapolation provides a plausible estimate of the missing radiances, hence making it suitable for radiance caching also.

It must be noted that other techniques can be used, such as the full hemicube sampling (which requires several passes), or a hemispherical parametrization of the hemisphere in vertex shaders.



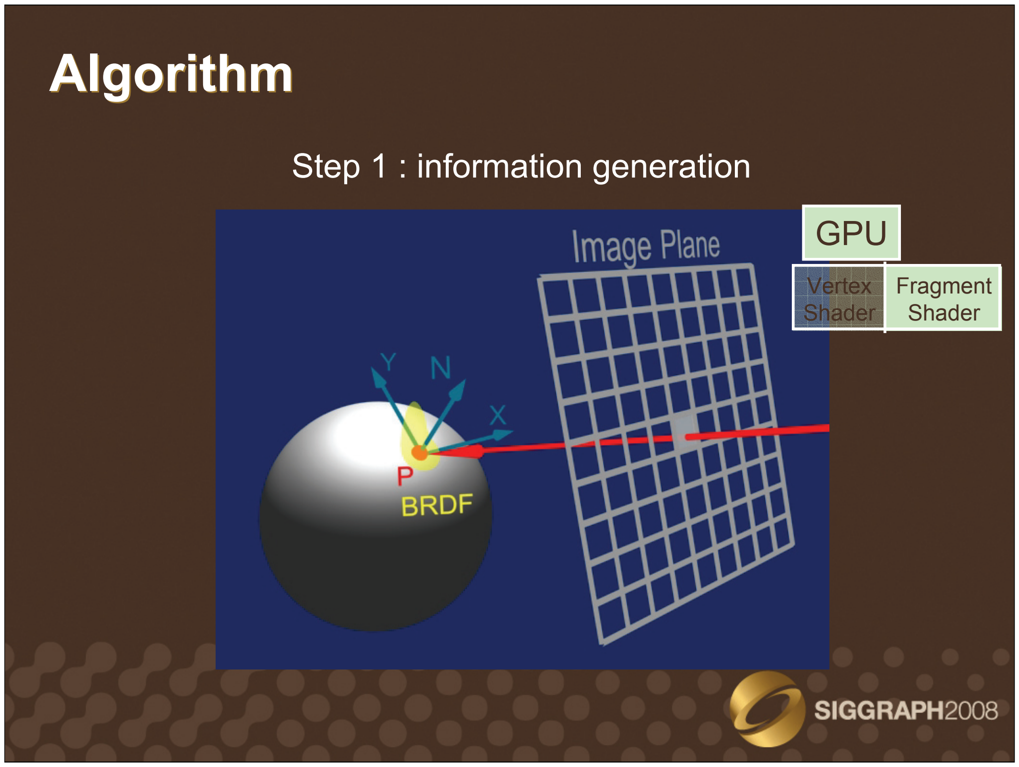
Reformulate IC Algorithm: How?

1. Octree → Splatting

2. Ray Tracing → Rasterization

SIGGRAPH2008

Now the algorithm has been reformulated for implementation on graphics hardware. Next section will present the entire algorithm for global illumination computation using irradiance splatting.

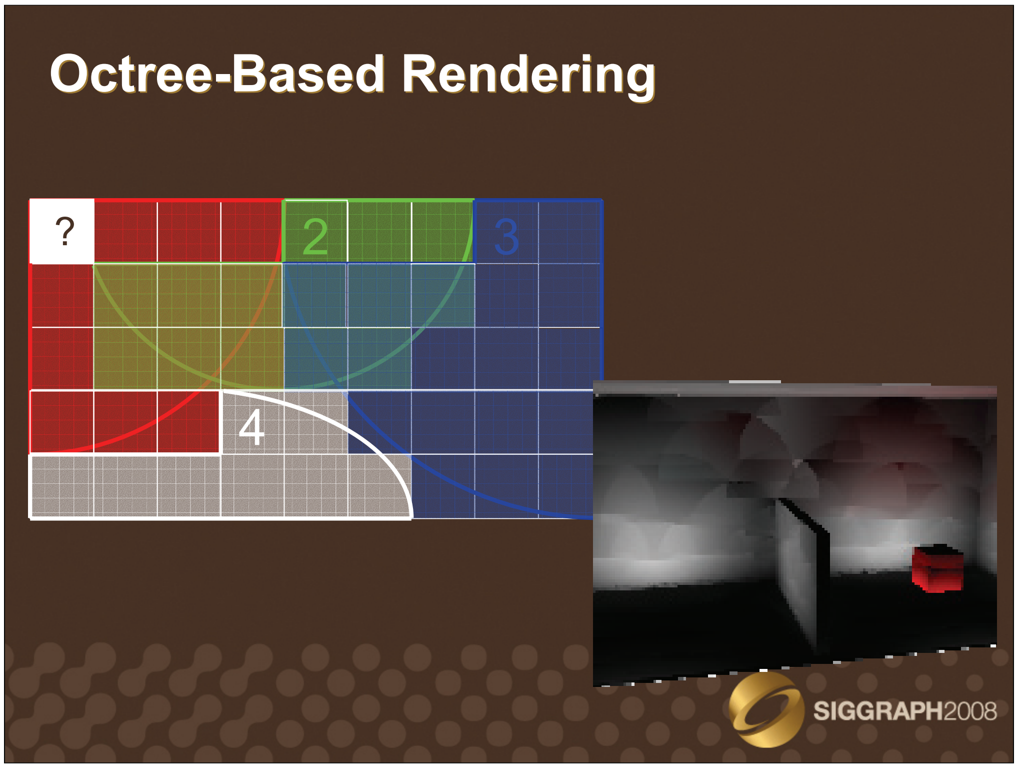


Algorithm

Step 1 : information generation

The first step of the algorithm consists in obtaining basic information about the points visible from the user point of view. In particular, this information includes the position and normal of the visible points. This data can be easily generated in one pass on the GPU using floating-point multiple render targets and programmable shaders.

In the next step, the algorithm transfers this data into the main memory. The CPU is then used to traverse the image and detect where new records are required to render the image.

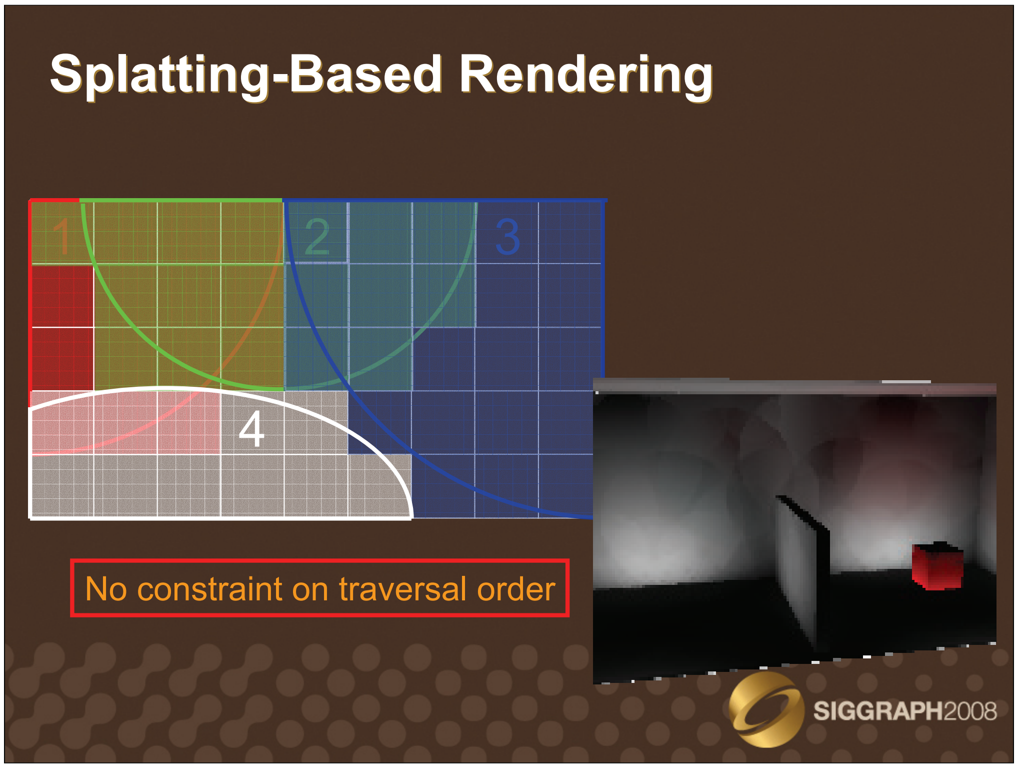


Octree-Based Rendering

Usually, the detection is performed by querying the irradiance cache (hence the octree) for each pixel as follows:

```
•For each visible point P with normal N corresponding to pixel p
    •W = GetSumOfContributions(P,N)
    •If (W < a)
        •R = CreateNewRecord(P,N)
        •IrradianceCache.StoreRecord(R)
        •p.radiance = R.irradiance*SurfaceReflectance(P)
    •Else
        •E = EstimateIrradiance(P,N)
        •P.radiance = E*SurfaceReflectance(P)
    •EndIf
•EndFor
```

While this algorithm ensures the presence of a sufficient number of records, such records are not used in an optimal way: when record 2 is created, the algorithm only propagates the contribution of 2 to the pixels which have not been checked yet. The previous pixels remain unchanged, even though record 2 may contribute to their radiance. If the image is traversed linearly, disturbing artifacts may appear: in this example the image is traversed from bottom to top. Therefore, the records contribute only to the points located above them in the image. This problem is usually compensated by using other traversal algorithms, typically based on a hierarchical subdivision of the image.



Splatting-Based Rendering

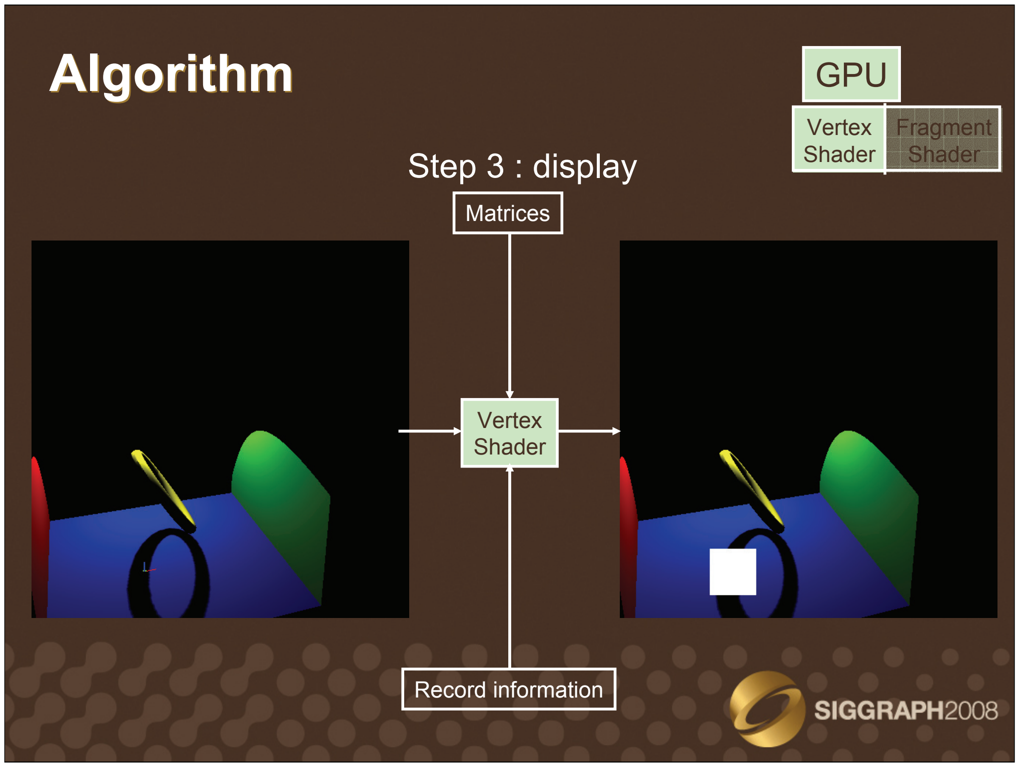
No constraint on traversal order

SIGGRAPH2008

When using irradiance splatting, the detection code becomes:

```
•For each visible point P with normal N corresponding to pixel p
    •W = GetSumOfContributions(P,N)
    •If (W < a)
        •R = CreateNewRecord(P,N)
        •IrradianceCache.StoreRecord(R)
        •SplatRecord(R)
        •p.radiance = R.irradiance*SurfaceReflectance(P)
    •Else
        •// Nothing to do: the radiance value depends of surrounding records and may be
        updated
    •EndIf
•EndFor
```

In our method, the records are splatted onto their entire zone of influence: even pixels which have been previously checked can be updated by the addition of a novel record. Hence this method removes the constraint on the image traversal order. In the example image, we used a simple linear traversal. The zone of influence of each record is completely accounted for by our splatting method.



Once all records have been computed, each record must be rendered on the GPU. Let us consider a record located at point P with normal N, and with an harmonic mean distance to surrounding objects H. The zone of influence of the record is contained within a screen-aligned square. Let us consider the vertices of a square such that:

```
 v0 = (1.0, 0.0, 0.0)
 v1 = (1.0, 1.0, 0.0)
 v2 = (0.0, 1.0, 0.0)
 v3 = (0.0, 0.0, 0.0)
```

Each vertex can be transformed into screen space using the following function:

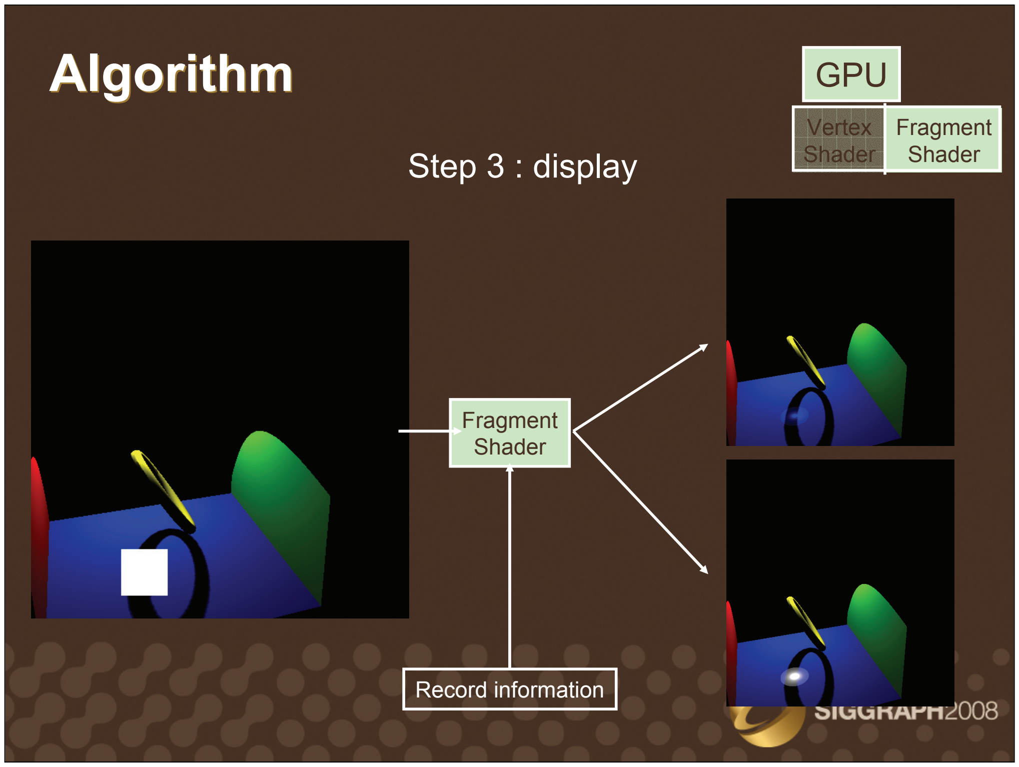
```
float4 TransformVertex(Vertex v)
{
                    float4 projPos = ModelViewMatrix*P; // Transform the point into camera space
                    float scale = projPos.w;
                    float4 projPos /= projPos.w;

                    float radius = a*H; // Radius of the zone of influence in camera space

                    float4 radiusV = float4(radius, radius, 0.0, 0.0);
                    // Project the corner of the bouding box into screen coordinates
                    float4 topRight = ProjectionMatrix*(projPos + radiusV*scale);
                    float4 bottomLeft = ProjectionMatrix *(projPos - radiusV*scale);
                    topRight /= topRight.w;
                    bottomLeft /= bottomLeft.w;

                    float2 tr = topRight.xy;
                    float2 bl = bottomLeft.xy;
                    float2 delta = tr - bl;

                    // return the screen coordinates of the vertex
  return float4(
    bl.x + vertex.x*delta.x,
    bl.y + vertex.y*delta.y,
    0.0, 1.0);
}
```

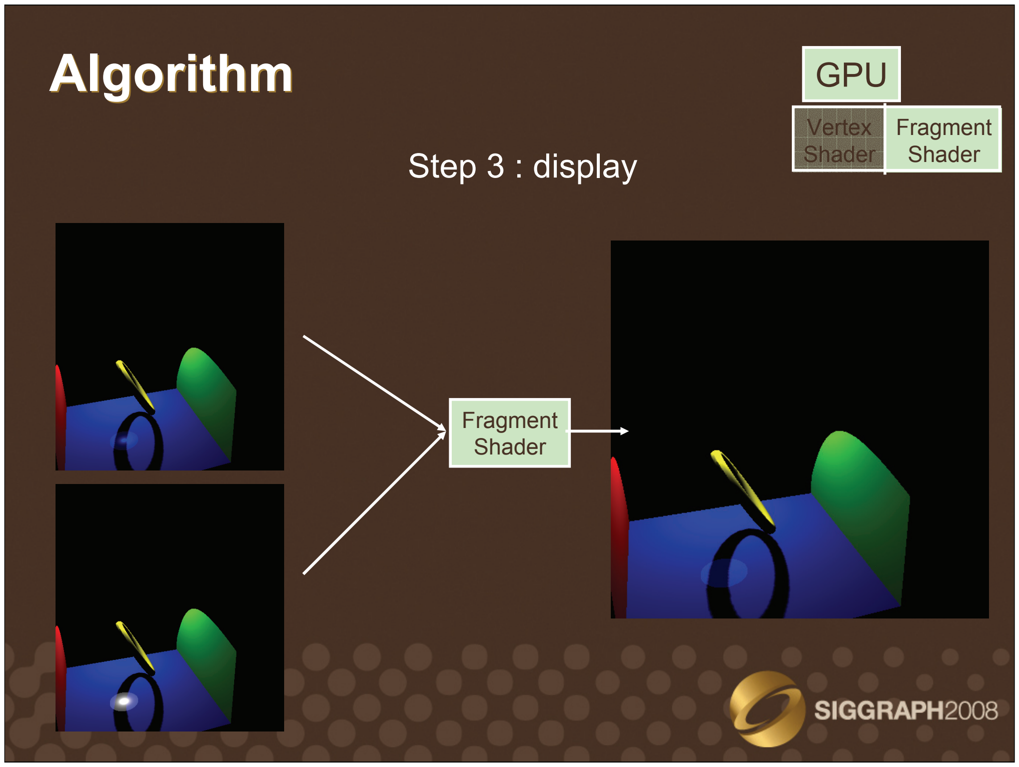


Algorithm

Step 3 : display

The fragment shader then computes the actual value of the weighting function for each point within the quadrilateral, using the formula defined in [Ward88]. If the record is allowed to contribute to the lighting of a visible point, its contribution is computed using irradiance gradients such as in [Ward92]. The output of the shader for the corresponding fragment is a HDR RGBA value, where RGB represents the weighted contribution of the record, and A represents the weight of the record's contribution.

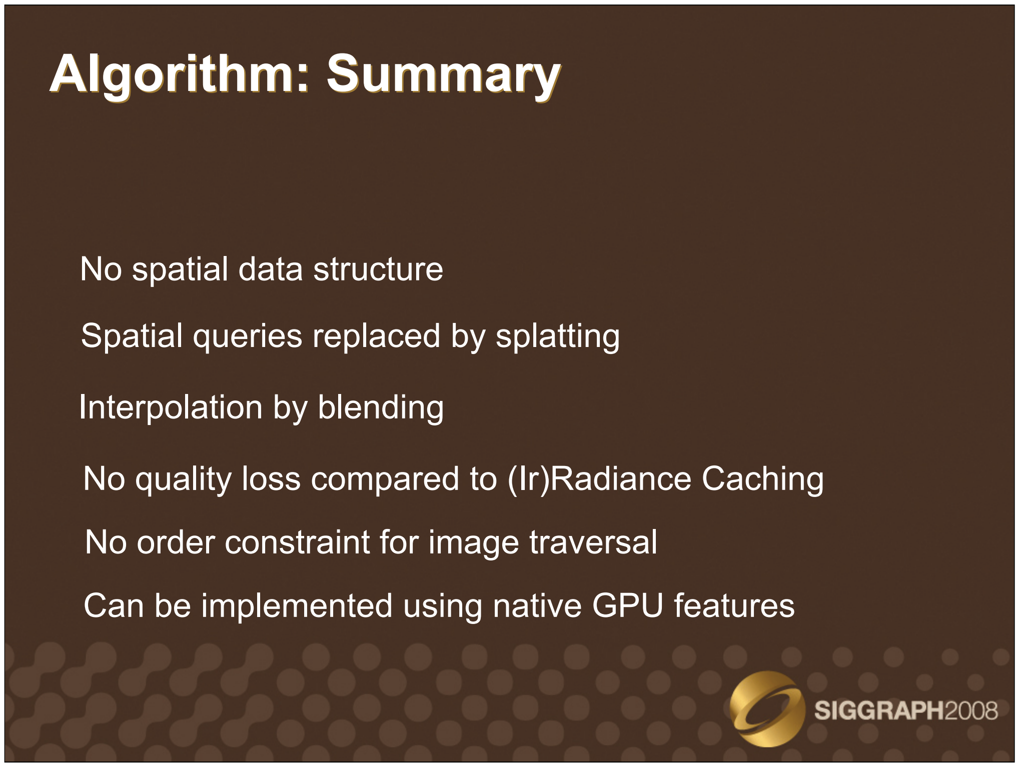
When several records are rendered, their RGBA values are simply added together using the floating-point alpha blending of graphics hardware (glBlendFunc(GL_ONE, GL_ONE)). This yields a sum of weighted contributions in the RGB components, and a sum of weights in the A component.



Algorithm

Step 3 : display

In a final step, a texture containing the RGBA values discussed above is used in a final render pass. This pass renders a single screen-sized quadrilateral. For a given pixel the fragment shader fetches the corresponding RGBA value, and outputs RGB/A to perform the irradiance estimation described in [Ward88].



Algorithm: Summary

No spatial data structure

Spatial queries replaced by splatting

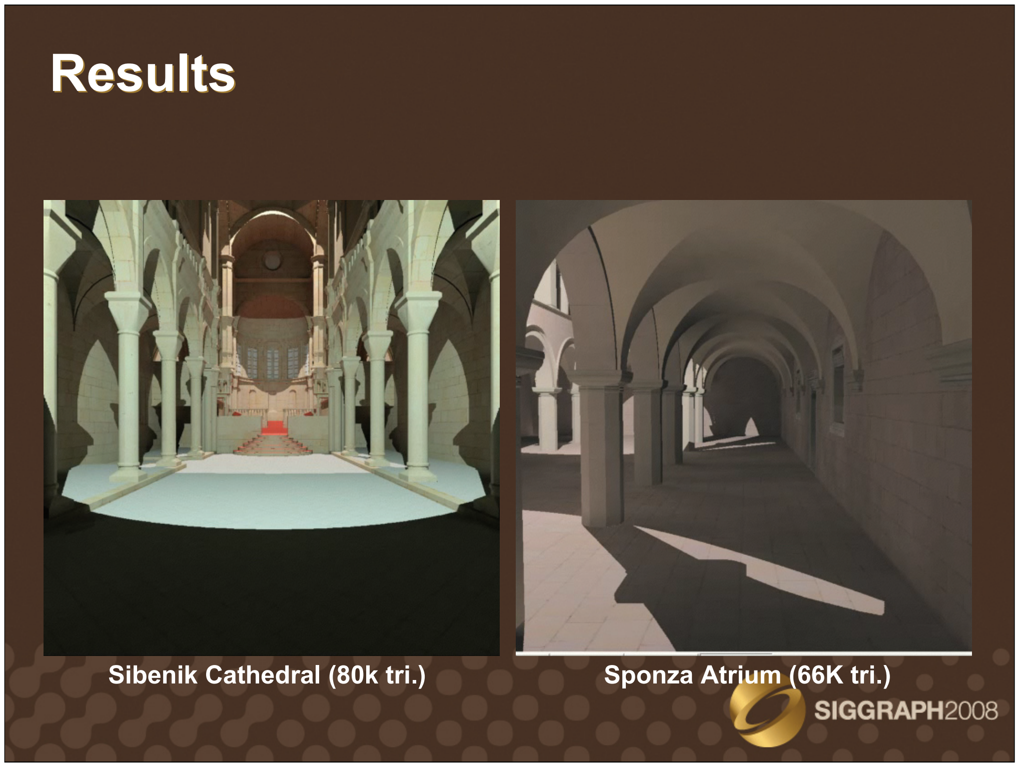
Interpolation by blending

No quality loss compared to (Ir)Radiance Caching

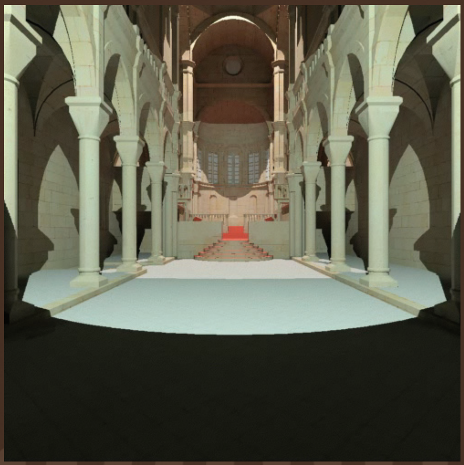
No order constraint for image traversal

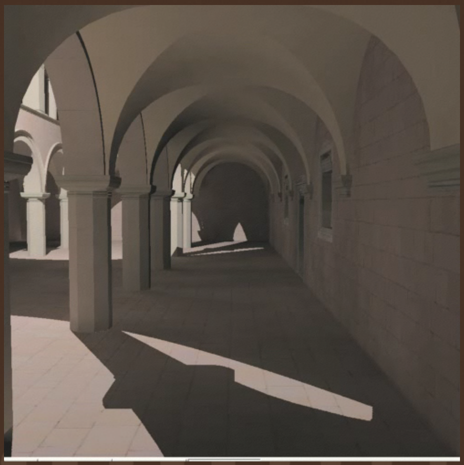
Can be implemented using native GPU features

SIGGRAPH2008

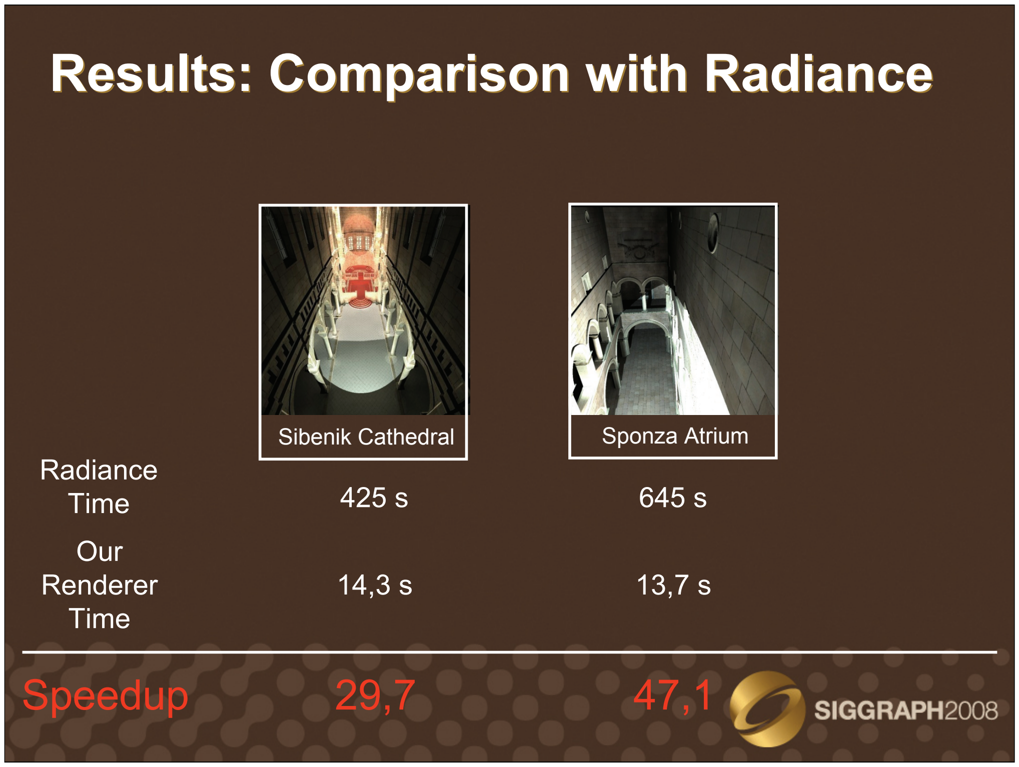


Results


Sibenik Cathedral (80k tri.)


Sponza Atrium (66K tri.)

Those videos were rendered on a GeForce 6800. The scene being static, the irradiance cache is reused across frames. Hence the first frame has been computed in approximately 20s, while the other frames took approximately 1s to render.



Results: Comparison with Radiance

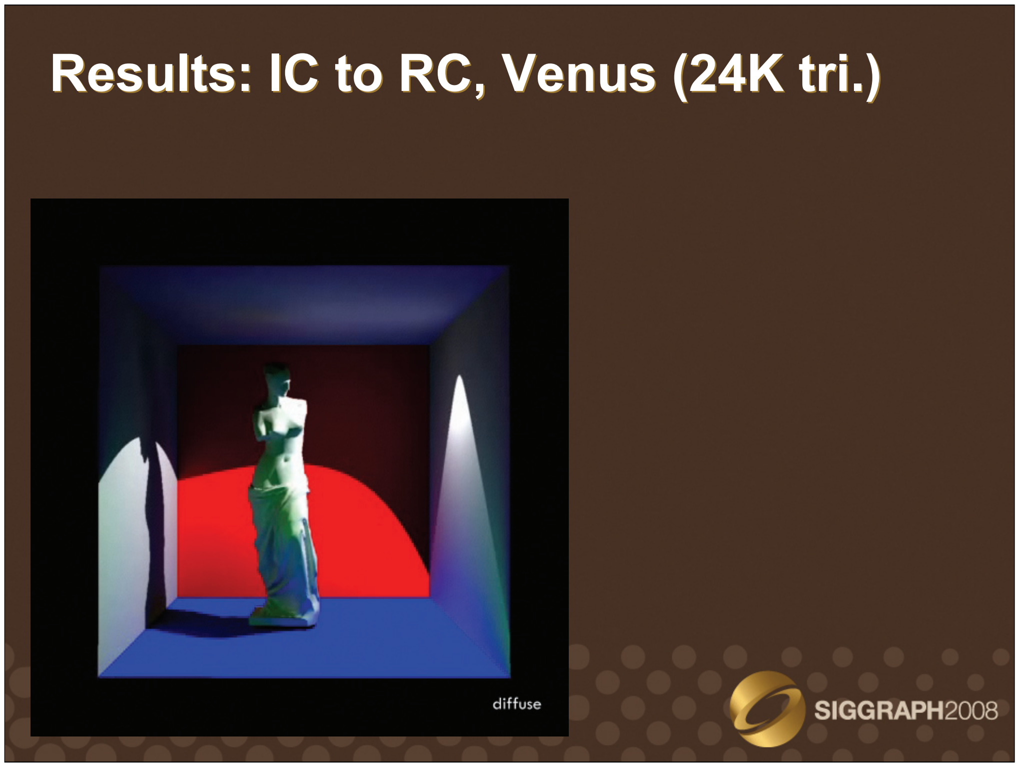
Sibenik Cathedral

Sponza Atrium

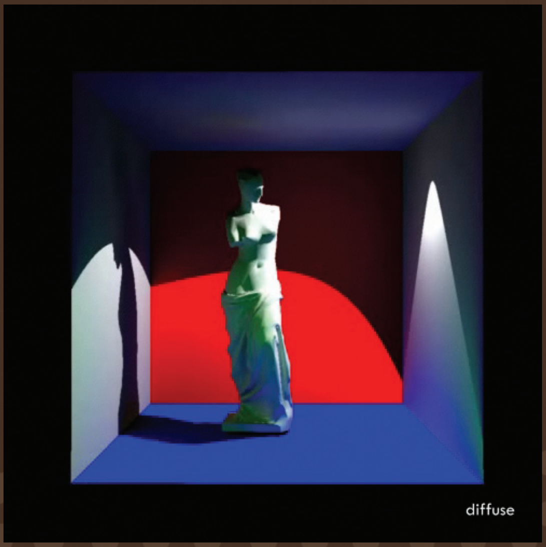
	Sibenik Cathedral	Sponza Atrium
Radiance Time	425 s	645 s
Our Renderer Time	14,3 s	13,7 s
Speedup	29,7	47,1

SIGGRAPH2008

A comparison between our GPU-based method and the Radiance Software.



Results: IC to RC, Venus (24K tri.)



SIGGRAPH2008

This method can be straightforwardly extended to glossy global illumination using radiance caching.

Chapter 8

Hardware Implementation

In the classical Irradiance Caching algorithm, rays are first traced from the viewpoint towards the scene. For each corresponding intersection point, the irradiance cache is queried to determine whether a new irradiance record has to be created. When a new record is required, the irradiance value is computed using distribution ray tracing. The newly created record is then stored in an octree, called the irradiance cache. For nearby intersection points the record is retrieved by querying the octree.

In this chapter, we consider the use of graphics hardware for high speed computation and rendering of the irradiance cache. However, besides the high computational power of graphics processors, efficiency is highly dependent on the adequacy between the processor model and the implemented algorithm. More precisely, stream processors such as GPUs are massively parallel, and hence are not very efficient for dealing with the repeated branching inherent to recursive structures. Also, as pointers are not natively supported, the implementation of an unbalanced octree is not straightforward. To achieve a better efficiency, we reformulate the algorithm to fit to the computational model of graphics hardware: parallel computing, simple data structures, and limited conditional statements.

To this end, we first consider an alternate algorithm for irradiance interpolation within the zones covered by irradiance records: the irradiance splatting. As GPUs are not natively dedicated to ray tracing and the traversal of acceleration structures, we also propose a GPU-based hemisphere sampling for the computation of 1-bounce global illumination.

8.1 Irradiance Splatting

The Irradiance Splatting algorithm takes the irradiance interpolation problem backwards compared to the octree-based method: instead of starting from a point an look for the nearby records, this method starts from a record and accumulates its contribution to the lighting at each visible point of the scene by splatting the record onto a *splat buffer*, which has the same size as the frame buffer. Each pixel $SPLATBUFF(x, y)$ of the splat buffer is a pair (L_o, w), where L_o is the sum of the weighted contributions of the records, and w is the cumulated weight.

The irradiance splatting (Algorithm 1) is designed for computing the contribution of an irradiance record to the indirect lighting of visible points. This approach starts on the irradiance estimate equation used in the irradiance caching interpolation scheme:

$$E(\mathbf{p}) = \frac{\sum_{k \in S(\mathbf{p})} E_k(\mathbf{p}) w_k(\mathbf{p})}{\sum_{k \in S(\mathbf{p})} w_k(\mathbf{p})} \tag{8.1}$$

The indirect irradiance $E(\mathbf{p})$ at point $\mathbf{p}$ is estimated as the weighted average of the contributions of nearby irradiance records evaluated at point $\mathbf{p}$. The weight allocated to a record k at point $\mathbf{p}$

P. Gautron

with normal $\mathbf{n}$ is defined in [WRC88]:

$$w_k(\mathbf{p}) = \frac{1}{\frac{\|\mathbf{p}-\mathbf{p}_k\|}{R_k} + \sqrt{1 - \mathbf{n}\cdot\mathbf{n}_k}} \tag{8.2}$$

where $\mathbf{p}_k$, $\mathbf{n}_k$ and R_k are respectively the location of record k, its normal and the harmonic mean distance to the objects visible from $\mathbf{p}_k$. The user-defined value a represents the accuracy of the computation. This value is used to threshold the contribution area of the records: a record k contributes to the estimate of the outgoing radiance at point $\mathbf{p}$ if and only if

$$w_k(\mathbf{p}) \geq \frac{1}{a} \tag{8.3}$$

Substituting Equation 8.2 into Equation 8.3 and assuming $\mathbf{n} = \mathbf{n}_k$, one can see that record k can contribute to the estimate of the outgoing radiance at point $\mathbf{p}$ only if:

$$\|\mathbf{p} - \mathbf{p}_k\| \leq aR_k \tag{8.4}$$

Therefore, Equation 8.4 guarantees that a record k cannot contribute to the outgoing radiance of a point outside a sphere I_k centered at $\mathbf{p}_k$, with radius $r_k = aR_k$.

Algorithm 1 Irradiance splatting

```
Let $k = \{\mathbf{p}_k, \mathbf{n}_k, E_k, R_k\}$ be the considered record
Let $I_k$ be the sphere centered at $\mathbf{p}_k$ with radius $aR_k$
Determine the bounding box of $I_k$ on the image plane
for all pixel $P(x,y) = \{\mathbf{p}, \mathbf{n}, \rho_d\}$ in the bounding box do
  // Evaluate weighting function at p
  $w = \frac{1}{\frac{\|\mathbf{p}-\mathbf{p}_k\|}{R_k} + \sqrt{(1-\mathbf{n}\cdot\mathbf{n}_k)}}$
  if $w \geq \frac{1}{a}$ then
    //Compute the contribution of record k at point p
    $E'_k = E_k(1 + \mathbf{n}_k \times \mathbf{n} \cdot \nabla_r + (\mathbf{p} - \mathbf{p}_k) \cdot \ \nabla_t)$
    // Compute the outgoing radiance
    $L_o = \rho_d E'_k$
    // Accumulate into the irradiance splat buffer
    $SPLATBUFF(x,y).L_o += wL_o$
    $SPLATBUFF(x,y).w += w$
  end if
end for
```

Given a camera, the *irradiance splatting* splats the sphere I_k onto the image plane (Figure 8.1).

In practice, this splatting is implemented within a vertex shader, where the record properties are stored in the attributes of each vertex of a quadrilateral $(-1,-1)$, $(1,-1)$, $(1,1)$, $(-1,1)$. Note that, while splatting a single point on the screen is also feasible, graphics hardware have a limited maximum point size in pixels, which may complicate the splatting of records with a large zone of influence.

The vertex attributes contain all the fields contained in the record: the position, the surface normal, the harmonic mean distance to the surrounding environment, the irradiance value and the translation/rotation gradients. For the sake of efficiency, the inverse squared harmonic mean is also passed in a vertex attribute.

The weighting function (Equation 8.2) is evaluated for each point visible through pixels covered by I_k. Then, the weight is checked against the accuracy value (Equation 8.3). For each

Algorithm 2 Pseudo-code for irradiance splatting: Vertex Shader

```
uniform float a;

attribute vec4 recordPosition;
attribute vec4 recordNormal;
attribute vec3 irradiance;
attribute vec3 rotGradR;
attribute vec3 rotGradG;
attribute vec3 rotGradB;
attribute vec3 posGradR;
attribute vec3 posGradG;
attribute vec3 posGradB;
attribute float harmonicMean;
attribute float sqInvHarmonicMean;

// Compute the record position in camera space
vec4 camSpacePosition = ModelViewMatrix * recordPosition;
// Compute the splatted sphere of influence in camera space
vec4 splatSphereBox(a*harmonicMean, a*harmonicMean, 0.0, 0.0);
// Compute the actual corners of the box in screen space
vec4 topRight = ProjectionMatrix*(
    camSpacePosition + splatSphereBox);
topRight /= topRight.w;
vec4 bottomLeft = ProjectionMatrix*(
    camSpacePosition - splatSphereBox);
bottomLeft /= bottomLeft.w;
// Deduce the projected vertex position
vec2 delta = topRight.xy - bottomLeft.xy;
outputVertex = vec4(
    bottomLeft.x + inputVertex.x*delta.x,
    bottomLeft.y + inputVertex.y*delta.y,
    0.0
    1.0);
Directly pass the vertex attributes to the fragment shader
```

Algorithm 3 Pseudo-code for irradiance splatting: Fragment Shader

```
// Texture containing hit positions
uniform sampler2D hitPositions;
// Texture containing hit normals
uniform sampler2D hitNormals;
// Texture containing hit diffuse
uniform sampler2D hitColors;
// Target accuracy
uniform float a;
// Squared target accuracy
uniform float squaredA;
// Determine the texture coordinate for fetching scene
// information from the textures
vec2 textureCoord = vec2(pixelCoord.x,pixelCoord.y);
// Retrieve the corresponding hit position and surface normal
vec3 hitPos = texture2D(hitpos, textureCoord).xyz;
vec3 hitNormal = texture2D(hitnorm, textureCoord).xyz;
// Calculate the vector between the hit position and the record
vec3 diff = hitPos-position;
// Estimate the weighting function
// Distance factor
float sqrDistance = dot(diff, diff);
float e1 = sqrDistance * sqInvHarmonicMean;
// Normal divergence factor
float e2 = 0.2 * (1.0 - dot(normal , hitNormal));
// First rejection test
if(( e1+e2 > squaredA ))
     discard;

// Evaluate the inverse of the weighting function
float invWeight = (sqrt(e1) + sqrt(e2));
// As the weighting function is infinite at the record
// location, perform clamping on the inverse weight
invWeight = max(EPSILON, invWeight);
// Test the final weight against a
if (a < invWeight )
      discard;


// From this point, the record is known as usable for
// interpolation at the pixel position
computeContribution();
```

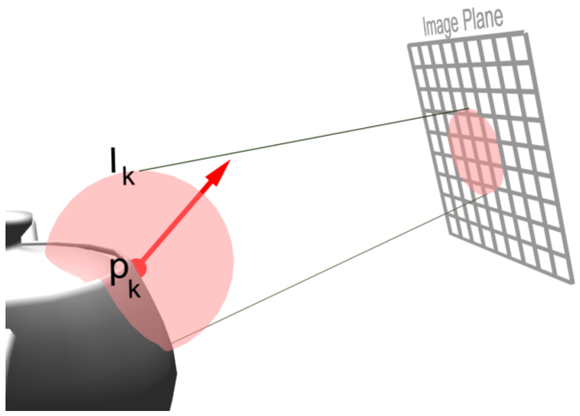


Figure 8.1: The sphere I_k around the position $\mathbf{p}_k$ of the record k is splatted onto the image plane. For each point within the sphere splat, the contribution of record k is accumulated into the irradiance splat buffer.

pixel passing this test, the algorithm computes the contribution of record k to the outgoing radiance estimate(Algorithm 3).

The outgoing radiance contribution L_o at a point $\mathbf{p}$ as seen through a pixel is obtained by evaluating the rendering equation:

$$L_o(\mathbf{p}, \omega_o) = \int_H L_i(\mathbf{p}, \omega_i) f(\omega_o, \omega_i) \cos(\theta_i) \mathrm{d}\omega_i \tag{8.5}$$

where ω_i and ω_o are respectively the incoming and outgoing directions. $L_i(\mathbf{p}, \omega_i)$ is the radiance incoming at $\mathbf{p}$ from direction ω_i. $f(\omega_o, \omega_i)$ is the surface BRDF evaluated for the directions ω_i and ω_o. In the case of irradiance caching, we only consider diffuse interreflections. Therefore, Equation 8.5 simplifies to:

$$L_o(\mathbf{p}) = \rho_d \int_H L_i(\mathbf{p}, \omega_i) \cos(\theta_i) \mathrm{d}\omega_i = \rho_d E(\mathbf{p}) \tag{8.6}$$

where ρ_d is the diffuse surface reflectance, and $E(\mathbf{p})$ is the irradiance at point $\mathbf{p}$. Therefore, the contribution of record k to the outgoing radiance at point $\mathbf{p}$ is

$$L_o = \rho_d E'_k(\mathbf{p}) \tag{8.7}$$

where $E'_k(\mathbf{p})$ is the irradiance estimate of record k at point $\mathbf{p}$, obtained through the use of irradiance gradients [WH92, KGBP05].

Note that this splatting approach can be used with any weighting function containing a distance criterion. In this chapter we focused on the weighting function defined in [WRC88], although the function proposed in [TL04] could be employed as well.

8.2 Reducing the Record Count in Interactive Applications

When computing an irradiance cache in the context of interactive or realtime applications, the number of records must be kept as low as possible to reduce the computation time. However, when a low number of records are used to render an image, circle-shaped artifacts appear at

Algorithm 4 Pseudo-code for computing irradiance contribution

```
// Compute the actual weight
float weight = 1.0 / invWeight;
// Fetch the diffuse reflectance
vec3 hitColor = texture2D(hitColors, textureCoord).xyz;
// Compute the position gradient component
vec3 diffDotPosGradient = vec3(
        dot(diff, posGradientRed),
        dot(diff, posGradientGreen),
        dot(diff, posGradientBlue)
        );
// Compute the rotation gradient component
vec3 normalsCrossProduct = cross(normal, hitNormal);
vec3 nCrossNiDotRotGradient = vec3(
        dot(normalsCrossProduct, rotGradientRed),
        dot(normalsCrossProduct, rotGradientGreen),
        dot(normalsCrossProduct, rotGradientBlue)
        );
FragmentColor.rgb = weight*(hitColor.xyz/π)*(
        irradiance +
        diffDotPosGradient +
        nCrossNiDotRotGradient
        );
FragmentColor.a = weight;
```

the edge of the zone of influence of the records. This is due to Equation 8.3, in which the contribution of a record is considered null if the $w(P) < 1/a$. The contribution is then abruptly cut at the edge of the zone of influence of the records, yielding the circles. This issue can be simply addressed by subtracting $1/a$ to the original weighting function proposed in [WRC88]. The record contribution now decreases to 0, yielding a smooth transition with the zones of influence of nearby records (Figure 8.2). Furthermore, the visual impact of potential errors due to over/underestimation of gradients can also be reduced.

8.3 Cache Miss Detection

The rendering process of irradiance caching requires the generation of a number of irradiance records to obtain a reasonable estimate of the indirect lighting for each point visible from the camera. Therefore, even though the values of the records are view-independent, their locations are determined according to the current viewpoint.

Usually, the irradiance cache records are stored in an octree for fast lookups. For a given pixel of the image, a ray is traced in the scene to find the point **p** visible through this pixel. At this point, the octree is queried to determine whether a new record is required. If yes, the actual indirect lighting is computed at **p**: a record is created and stored in the cache. Otherwise the cache is used to compute a satisfying estimate if the indirect lighting using interpolation. Since each visible point is processed only once, a newly created record cannot contribute to the indirect lighting of points for which the indirect lighting has already been computed or estimated. Therefore, unless an appropriate image traversal algorithm is employed, artifacts may be visible (Figure 8.3(a)). The traversal typically relies on hierarchical subdivision of the image to reduce and spread the errors. Artifact-free images can only be obtained in two passes: in the first pass,

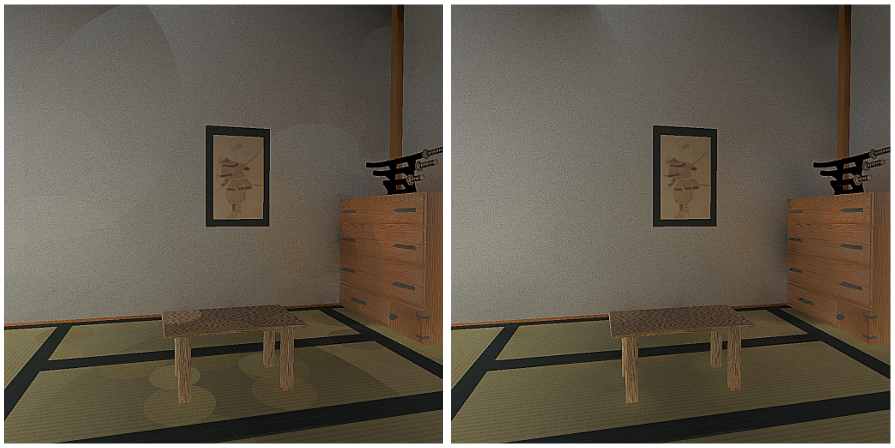

(a) Weighting: $w_k(\mathbf{p})$ (b) Weighting: $w_k(\mathbf{p}) - 1/a$

Figure 8.2: Indirect lighting computed using 89 irradiance records ($a = 0.8$) using the original and the new weighting function.

the records required for the current viewpoint are generated. In the second pass, the scene is rendered using the contents of the cache.

On the contrary, the irradiance splatting algorithm is based on the independent splatting of the records. The contribution of a record is splatted onto each visible point within its zone of influence, even though the points have been previously checked for cache miss (Figure 8.3(b)). Therefore, this method avoids the need of a particular image traversal algorithm without harming the rendering quality. For the sake of efficiency and memory coherence, a linear traversal of the image is recommended. Note that, however, any other image traversal algorithm could be used.

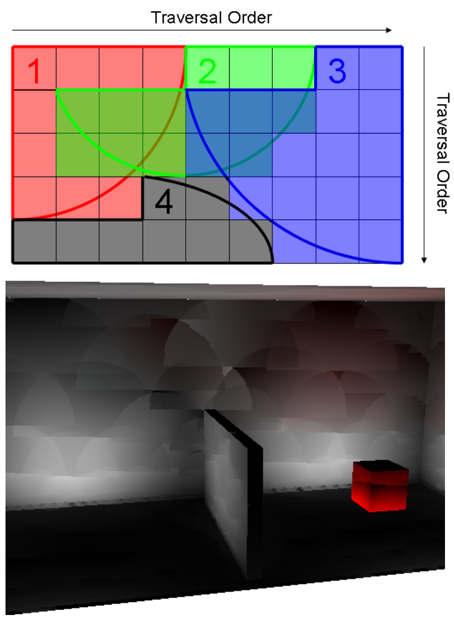
Once the algorithm has determined where a new record has to be generated, the irradiance value and gradients must be computed by sampling the hemisphere above the new record position. The following section details a simple method for hemispherical sampling using graphics hardware in the context of 1-bounce global illumination computation.

8.4 GPU-Based Hemisphere Sampling

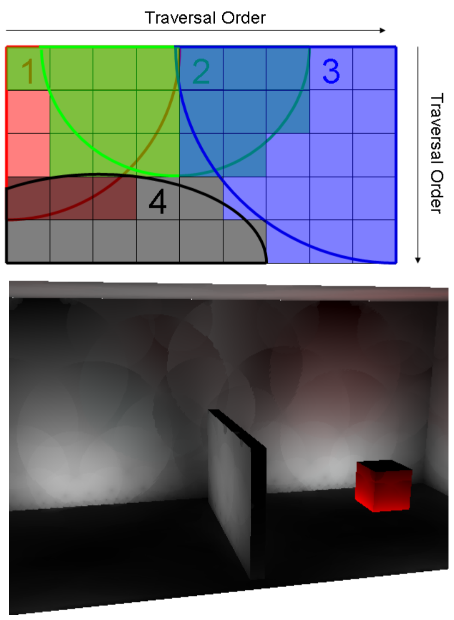
In the method described hereafter, the incoming irradiance associated with a record is generated using both the GPU and CPU. First, the GPU performs the actual hemisphere sampling. Then, the CPU computes the corresponding irradiance value.

8.4.1 Hemisphere Sampling on the GPU

In the context of global illumination, the visibility tests are generally the bottleneck of the algorithms. We propose to leverage the power of graphics hardware to speed up the computation. In the classical hemi-cube method [CG85], 5 rendering passes are necessary for full hemisphere sampling. For the sake of efficiency, we preferred a 1-pass method similar to [SP89, LC04]: using a virtual camera placed at the record location, the rasterization engine samples the hemisphere above the record to compute the radiances incoming from each direction. However, as shown in Figure 8.4, using a single rasterization plane cannot account for the radiances incoming from grazing angles. To compensate for the incomplete hemisphere coverage, Larsen *et al.* [LC04] divide the obtained irradiance by the plane coverage ratio. A more accurate method consists in virtually stretching border pixels to fill the remaining solid angle (Figure 8.4). This method



(a) Octree-based 1 pass



(b) Octree-based 2-pass/Splatting 1-pass

Figure 8.3: Upper row: the irradiance records 1, 2, 3, 4 and their respective zones of influence. We consider the use of a sequential traversal of the image for determining the pixels for which a new record is needed. In the classical, octree-based method, a newly created record cannot contribute to pixels which have already been examined. Therefore, discontinuities appear in the contribution zone of the records, yielding artifacts. Those artifacts are illustrated in the upper row: record 1 is created at the point visible through the first pixel, and used for estimating the indirect lighting in the next 3 pixels. When record 2 is created at the point visible through the fifth pixel, its contribution is not propagated to the previous pixels, hence creating a discontinuity. A second rendering pass is needed to generate an image without such discontinuities. Using the irradiance splatting method, each record contributes to all possible pixels, avoiding the need of a second rendering pass. Lower row: the indirect lighting obtained using these methods.

generally yields more accurate results than the approach of Larsen *et al.* (Table 8.1). Furthermore, a key aspect of this method is that the directional information of the incoming radiance remains plausible. Therefore, unlike in [LC04], the indirect glossy lighting could also be rendered correctly if radiance caching is used. A typical aperture size of the virtual camera is twice the aperture of the standard hemicube, that is approximately 126 degrees.

	Plane sampling	[LC04]	Border stretching
RMS Error	18.1%	10.4%	5.8%

Table 8.1: RMS error of 10000 irradiance values computed in the Sibenik Cathedral scene (Figure 8.9). The border stretching method yields more accurate results than previous approaches, while preserving the directional information of the incoming radiance. The reference values are computed by full hemisphere sampling using Monte Carlo integration.

The GPU-based hemisphere sampling requires a proper shading of the visible points, accounting for direct lighting and shadowing. This computation is also carried out by the graphics hardware using per-pixel shading and shadow mapping. We recommend using uniform shadow maps

[Wil78]: unlike perspective shadow maps [SD02] or shadow volumes [Hei91], uniform shadow maps are view-independent: in static scenes the shadow map is computed once and reused for rendering the scene from each viewpoint. Therefore, the algorithm reuses the same shadow map to compute all the records, hence reducing the shading cost. Furthermore, the incoming radiances undergo a summation to compute the irradiance. This summation is equivalent to a low-pass filter which blurs out the aliasing artifacts of the shadow maps, hence providing inexpensive, high quality irradiance values.

Once the hemisphere has been sampled, the algorithm computes the actual irradiance record.

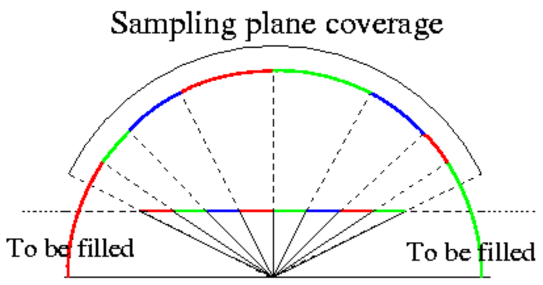


Figure 8.4: The hemisphere sampling is reduced to the rasterization of the scene geometry onto a single sampling plane. Since this plane does not cover the whole hemisphere, we use a border compensation method to account for the missing directions. Border pixels are virtually stretched to avoid zero lighting coming from grazing angles, yielding more plausible results.

8.4.2 Irradiance Computation

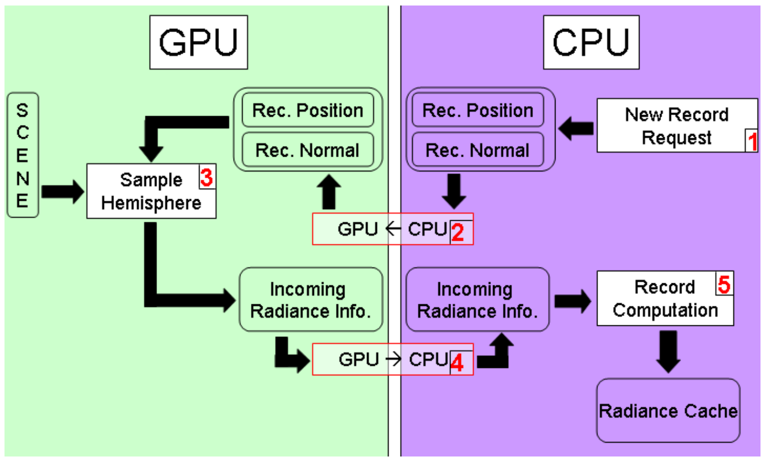


Figure 8.5: New record computation process. The numbers show the order in which the tasks are processed.

As shown in [LC04] the irradiance is defined as a weighted sum of the pixels of the sampling plane. This sum can be calculated either using the GPU and automatic mipmap generation [LC04], or using frame buffer readback and CPU-based summation (Figure 8.5). In the case of irradiance caching, a record contains the irradiance, the rotational and translational gradients, and the harmonic mean distance to the objects visible from the record location. Such values can typically stored within 22 floating-point values. Hence the method proposed by Larsen *et al.* would require the computation and storage of the mipmap levels of at least 6 RGBA

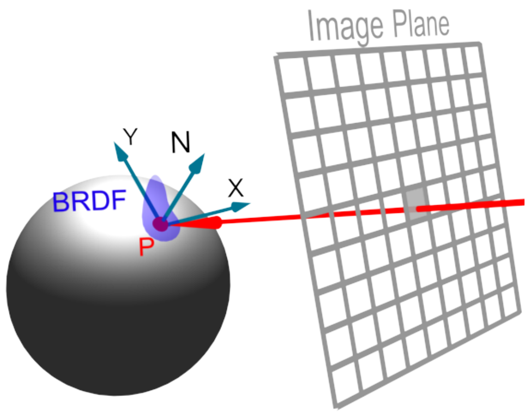


Figure 8.6: The irradiance splatting requires per-pixel information about geometry and materials: the hit point, local coordinate frame, and BRDF.

textures. Even with the latest graphics hardware, this solution turns out to be slower than a readback and CPU-based summation. Furthermore, using a PCI-Express-based graphics card and the asynchronous readbacks provided by OpenGL Pixel Buffer Objects, the pipeline stalls introduced by the readback tend to be very small. However, due to the fast evolution of graphics hardware, the possibility of using GPU-based summation should not be neglected when starting an implementation.

8.5 Global Illumination Rendering

The final image is generated in five main steps (Algorithm 5). Given a camera, the first step consists in obtaining per-pixel information about viewed objects: their position, local coordinate frame and BRDF (Figure 8.6).

In the second and third steps, the rendering process determines where new irradiance records are necessary to achieve the user-defined accuracy of indirect illumination computation. In Step 2, each existing record (possibly computed for previous frames) is splatted onto the splat buffer. Step 3 consists in reading back the irradiance splat buffer into the CPU memory. In Step 4, the algorithm traverses the irradiance splat buffer to determine where new irradiance records are required to achieve the user-defined accuracy (Cache miss detection). For each pixel (x, y) in the irradiance splat buffer, the cumulated weight is checked against the accuracy value a:

$$SPLATBUFF(x, y).w \geq a \tag{8.8}$$

If a pixel (x, y) fails this test, the existing cache records are insufficient to achieve the required accuracy. Therefore, a new record is generated at the location visible from pixel (x, y), and is splatted by the CPU onto the irradiance splat buffer (Figure 8.7).

Once $SPLATBUFF(x, y).w \geq a$ for every pixel, the data stored in the cache can be used to display the indirect illumination according to the accuracy constraint. At that time in the algorithm, the irradiance cache stored on the CPU memory differs from the cache stored on the GPU: the copy on the GPU represents the cache before the addition of the records described above, while the copy on the CPU is up-to-date.

The last rendering step is the generation of the final image using the cache contents (Figure 8.8). The irradiance cache on the GPU is updated, then the irradiance splatting algorithm is applied on each newly generated cache record. Hence the irradiance splat buffer contains the cumulated record weight and outgoing radiance contribution of all the irradiance records. Then,

Algorithm 5 Global illumination rendering

```
// Step 1
Generate geometric and reflectance data of objects viewed through pixels (GPU)
Clear the splat buffer
// Step 2
for all cache records do
   // The irradiance cache is empty for the first image,
   // and non empty for subsequent images
   Algorithm 1: splat the records onto the irradiance splat buffer (GPU)
end for
// Step 3
Read back the irradiance splat buffer from GPU to CPU
// Step 4
for all pixels $(x, y)$ in the irradiance splat buffer do
   if $SPLATBUFF(x, y).w < a$ then
      Compute a new incoming radiance record at corresponding hit point (GPU/CPU)
      Apply Algorithm 1: splat the new record (CPU)
   end if
end for
// Step 5
for all cache records do
   Apply Algorithm 1: splat all the newly generated records (GPU)
end for
//Normalize the irradiance splat buffer (GPU)
for all pixels $(x, y)$ in the irradiance splat buffer do
   $SPLATBUFF(x, y).L_o/ = SPLATBUFF(x, y).w$
end for
Combine the irradiance splat buffer with direct lighting (GPU)
```

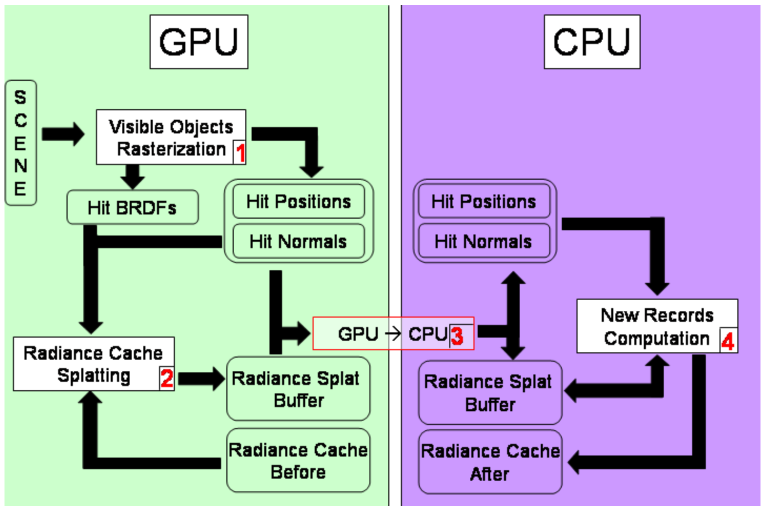


Figure 8.7: The irradiance cache filling process. The numbers show the steps defined in Algorithm 5. During this process, the irradiance cache stored on the CPU is updated whereas the copy on the GPU remains untouched.

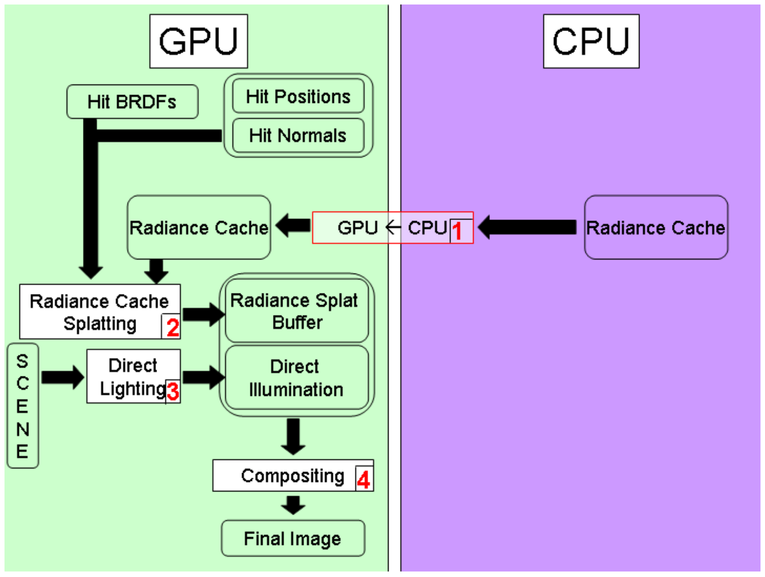


Figure 8.8: The final rendering task. The numbers show the processing order described in steps 4 and 5 of Algorithm 5.

the cumulated contribution at each pixel is divided by cumulated weight. This process yields an image of the indirect lighting in the scene from the current point of view. Combined with direct lighting, this fast method generates a high quality global illumination solution.

8.6 Implementation Details

This section gives technical information about the implementation of irradiance splatting. The rendering algorithm is based on deferred shading: the information on visible points is generated in a first pass by rasterizing the scene geometry. This information is used as the input of the irradiance splatting. Using this method, the most costly operations are only performed once for each pixel of the final image.

As shown in Figures 8.7 and 8.8, this rendering algorithm makes intensive use of the GPU computational power. The required base information (Figure 8.6) is generated by rasterizing the scene geometry on the GPU using a dedicated shader. Using multiple render targets, this shader renders all the necessary information in a single rendering pass. Figure 8.6 shows that this step includes the storage of per-pixel BRDF. In the case of irradiance caching, this information is the diffuse reflectance of the material (i.e. one RGB value). During splatting, the reflectance is combined with the irradiance contributions of the records.

The irradiance splatting can be performed either using the GPU or the CPU. Both are necessary, depending on the current context as explained hereafter. The irradiance splatting on the GPU is performed by drawing a quadrilateral and a dedicated vertex shader described above. Then, the splatting fragment shader processes each fragment so that its value represents the weighted contribution of the record. The fragment is accumulated in the irradiance splat buffer using the floating-point blending capabilities provided by graphics hardware. The final normalization (i.e. the division by the cumulated weight) is performed using a fragment shader in an additional pass for the final display.

The cache miss detection is performed by traversing the irradiance splat buffer sequentially using the CPU. Consequently, the GPU-based splatting cannot be used efficiently during the record addition step: this would require to read back the irradiance splat buffer from the GPU memory after the computation and splatting of each new record. The CPU implementation is designed the same way as for the GPU. Figure 8.6 shows that the irradiance splatting requires information about the visible objects. This information is computed on the GPU, then read back to the main memory once per frame. The overhead introduced by the data transfer from GPU to CPU turns out to be very small using PCI-Express hardware.

Once all necessary records are computed and splatted onto the irradiance splat buffer, the final picture containing both direct and indirect lighting has to be generated. The direct lighting computation is straightforwardly carried out by the GPU, typically using per-pixel lighting and shadow maps [Wil78, BP04]. To reduce the aliasing of shadow maps without harming the performance, we use high definition shadow maps (typically 1024×1024) with percentage-closer filtering [RSC87] using 16 samples. Hence the same shadow map can be used for both the record computation and the rendering of direct lighting in the final image. Note that higher quality images could be obtained using view-dependent shadowing methods such as shadow volumes for the visualization of the direct lighting in the final image.

The normalization of the irradiance splat buffer and the combination with direct lighting are finally performed in a single fragment shader which displays the final image.

8.7 Results

Note: the images and timings have been generated using an nVidia Quadro FX 3400 PCI-E and a 3.6 GHz Pentium 4 CPU with 1 GB RAM.

8.7.1 High Quality Rendering

This section details non-interactive high quality global illumination computation. First, we compare the results obtained with our GPU-based renderer with the well-known Radiance software [War94] in the context of irradiance caching.

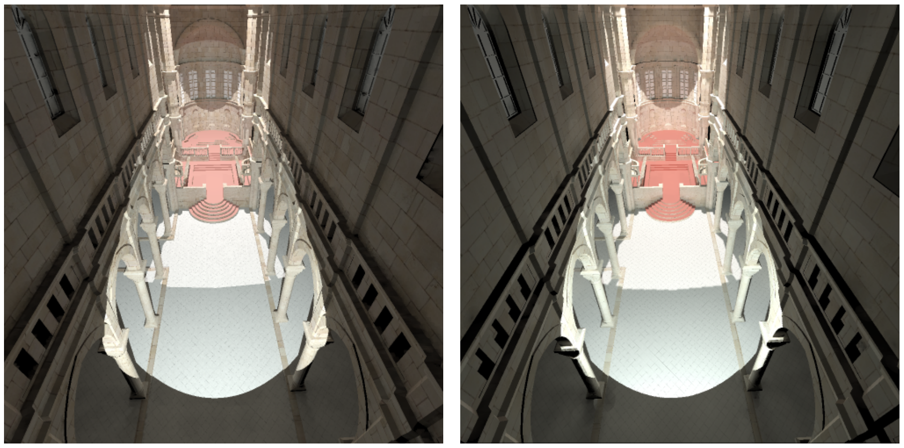
We have compared the rendering speed of the Radiance software and our renderer in diffuse environments: the *Sibenik Cathedral* and the *Sponza Atrium* (Figure 8.10). The images are rendered at a resolution of 1000×1000 and use a 64×64 resolution for hemisphere rasterization. The results are discussed hereafter, and summarized in Table 8.2.

a) *Sibenik Cathedral* This scene contains 80K triangles, and is lit by two light sources. The image is rendered with an accuracy parameter of 0.15. At the end of the rendering process, the irradiance cache contains 4076 irradiance records. The irradiance splatting on the GPU is performed in 188 ms. The Radiance software rendered this scene in 7 min 5 s while our renderer took 14.3 s, yielding a speedup of about 30×.

b) *Sponza Atrium* This scene contains 66K triangles and two light sources. Using an accuracy of 0.1, this image is generated in 13.71 s using 4123 irradiance records. These records are splatted on the GPU in 242.5 ms. Using the Radiance software with the same parameters, a comparable image is obtained in 10 min 45 s. In this scene, our renderer proves about 47× faster than the Radiance software.

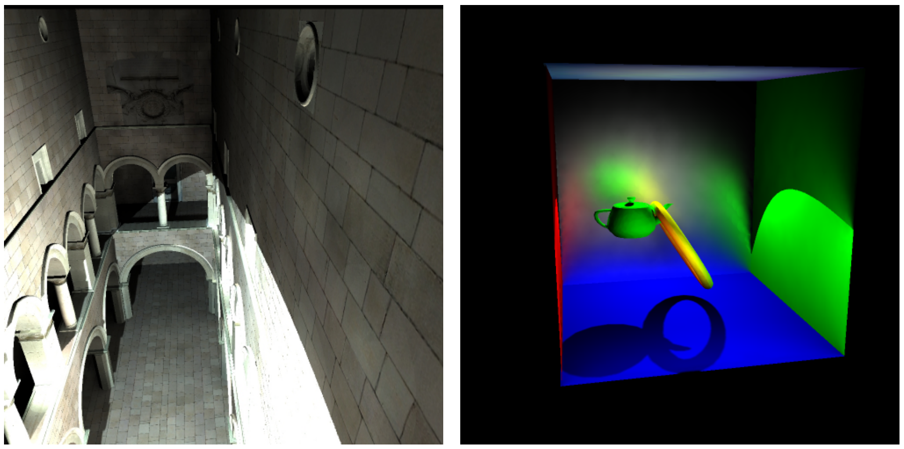
8.7.2 Interactive Global Illumination

An important aspect of the irradiance caching algorithm is that the values of the records are view-independent. In a static scene, records computed for a given viewpoint can be reused for other camera positions. Therefore, the irradiance splatting approach can also be used in the context of interactive computation of global illumination using progressive rendering. The direct lighting being computed independently, the user can walk through the environment while the irradiance cache is filled on the fly. Figure 8.11 shows sequential images of *Sam* scene



(a) Radiance (b) Our renderer

Figure 8.9: The Sibenik Cathedral scene (80K triangles). The images show first bounce global illumination computed with Radiance (a) and our renderer (b) (Model courtesy of Marko Dabrovic)



(a) Sponza Atrium (b) Cornell Box

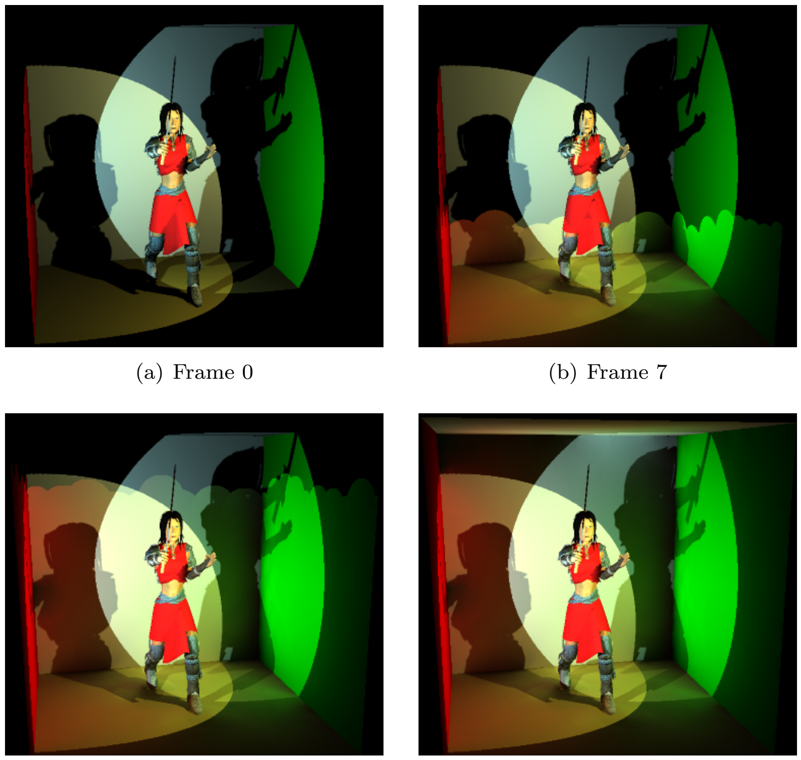
Figure 8.10: Images obtained with our renderer. The Sponza Atrium (66K triangles) is only made of diffuse surfaces (Model courtesy of Marko Dabrovic). The Cornell Box (1K triangles) contains a glossy back wall.

	Sibenik Cathedral	Sponza Atrium
Triangles	80K	66K
Accuracy	0.15	0.1
Radiance time (s)	425	645
Our renderer time (s)	14.3	13.7
Speedup	29.7	47.1

Table 8.2: Rendering times obtained using Radiance and irradiance splatting for high quality rendering of diffuse environments. Each image is rendered at resolution 1000×1000.

(63K triangles) obtained during an interactive session with an accuracy parameter of 0.5 and resolution 512×512. The global illumination is computed progressively, by adding at most 100 new records per frame. Our renderer provides an interactive frame rate (between 5 and 32 fps) during this session, allowing the user to move even though the global illumination computation is not completed. This method has been used for interactive walkthroughs in diffuse and glossy environments such as the *Sibenik Cathedral* and the *Castle*.

In this context, the irradiance splatting also proves useful for adjusting the value of the user-defined accuracy parameter a. In the classical irradiance caching method, the structure of the octree used to store the records is dependent on a. Therefore, the octree has to be regenerated for each new value of a. Using this approach the size of each splat is computed for each frame on the GPU, hence allowing the user to tune the value of a interactively to obtain the desired visual quality (Figure 8.12).

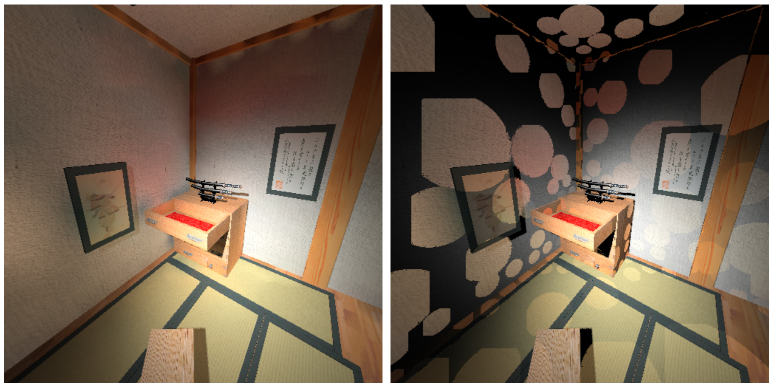

(a) Frame 0 (b) Frame 7

(c) Frame 11 (d) Frame 14

Figure 8.11: A progressive rendering session for interactive visualization of the *Sam* scene (63K triangles). Our renderer computes at most 100 new records per frame, hence maintaining an interactive frame rate (5 fps) during the global illumination computation. When the irradiance cache is full, the global illumination solution is displayed at 32 fps.

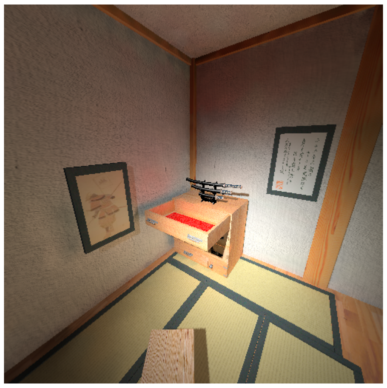
8.8 Conclusion

This chapter presented a reformulation of the irradiance caching algorithm by introducing irradiance splatting. In this method, the sphere of influence of each record is splatted onto the image plane. For each pixel within the splatted sphere, a fragment shader computes the contribution and weight of the record to the indirect lighting of the corresponding point. The record weight



(a) $a = 0.5$

(b) $a = 0.2$ using the records generated with $a = 0.5$



(c) $a = 0.2$

Figure 8.12: The irradiance splatting method allows for the dynamic modification of the user-defined accuracy value a, hence easing the parameter tuning process.

and weighted contribution are accumulated into the irradiance splat buffer using floating-point alpha blending. The final indirect lighting of visible points is obtained by dividing the weighted sum of the records by the cumulated weights in a fragment shader. Using this approach, each record contributes to all possible pixels, hence simplifying the cache miss detection algorithm. By avoiding the need of complex data structures and by extensively using the power of graphics hardware, the irradiance splatting allows for displaying global illumination in real-time.

Another speedup factor is the use of both GPU and CPU to compute the values of the irradiance records. While the GPU performs the costly hemisphere sampling, the CPU sums up the incoming radiances to compute the actual values and gradients of the records. This method requires many data transfers from the GPU to the CPU. Even though classical architectures such as AGP [Int02] would have introduced a prohibitive bottleneck [CHH02], the new PCI-Express 16× port [PCI03] provides transfer rates up to 8GB/s. Compared to AGP, the pipeline stalls are drastically reduced, making this method usable even in the case of on-the-fly computation of global illumination for interactive rendering.

Bibliography

[BP04] Michael Bunnell and Fabio Pellacini. *GPU Gems: Shadow map antialiasing*, pages 185–192. Addison Wesley, 1 edition, 2004.

[CG85] Michael Cohen and Donald P. Greenberg. The Hemi-Cube: A Radiosity Solution for Complex Environments. In *Proceedings of SIGGRAPH*, volume 19, pages 31–40, August 1985.

[CHH02] Nathan A. Carr, Jesse D. Hall, and John C. Hart. The ray engine. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 37–46. Eurographics Association, 2002.

[Hei91] Tim Heidmann. Real shadows, real time. *Iris Universe*, 18:23–31, 1991.

[Int02] Intel Corporation. *AGP v3.0 Specification.* 2002.

[KGBP05] Jaroslav Křivánek, Pascal Gautron, Kadi Bouatouch, and Sumanta Pattanaik. Improved radiance gradient computation. In *Proceedings of SCCG*, pages 149–153, 2005.

[LC04] Bent Dalgaard Larsen and Niels Jorgen Christensen. Simulating photon mapping for real-time applications. In A. Keller and H. W. Jensen, editors, *Proceedings of Eurographics Symposium on Rendering*, pages 123–131, June 2004.

[PCI03] PCI-SIG. *PCI-Express v1.0e Specification.* 2003.

[RSC87] W.T. Reeves, D.H. Salesin, and R.L. Cook. Rendering antialiased shadows with depth maps. In *Proceedings of SIGGRAPH*, volume 21, pages 283–291, August 1987.

[SD02] Marc Stamminger and George Drettakis. Perspective shadow maps. In *Proceedings of SIGGRAPH*, July 2002.

[SP89] Francois Sillion and Claude Puech. A General Two-Pass Method Integrating Specular and Diffuse Reflection. In *Proceedings of SIGGRAPH*, volume 23, pages 335–344, July 1989.

[TL04] Eric Tabellion and Arnauld Lamorlette. An approximate global illumination system for computer-generated films. In *Proceedings of SIGGRAPH*, August 2004.

[War94] Gregory J. Ward. The RADIANCE Lighting Simulation and Rendering System. In *Computer Graphics Proceedings, Annual Conference Series, 1994 (Proceedings of SIGGRAPH*, pages 459–472, 1994.

[WH92] Gregory J. Ward and Paul Heckbert. Irradiance Gradients. In *Proceedings of Eurographics Workshop on Rendering*, pages 85–98, Bristol, UK, May 1992.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of SIGGRAPH*, pages 270–274, 1978.

[WRC88] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A Ray Tracing Solution for Diffuse Interreflection. In *Proceedings of SIGGRAPH*, volume 22, pages 85–92, August 1988.

Temporal Radiance Caching

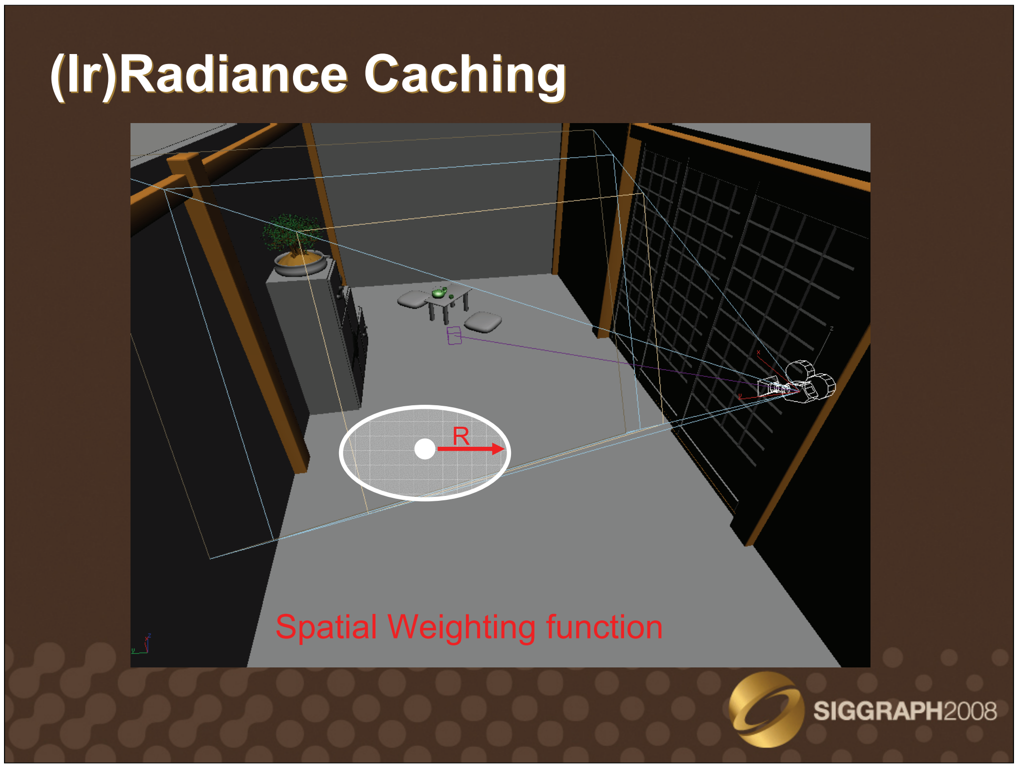
SIGGRAPH2008

Pascal Gautron

R&D Engineer

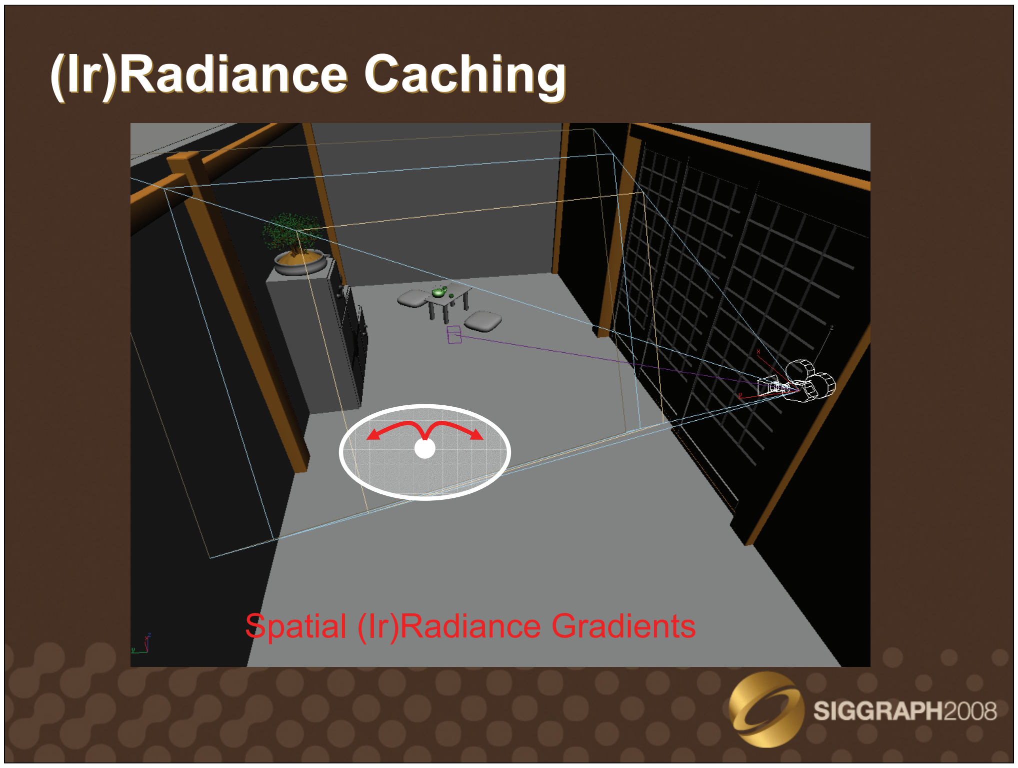
Thomson Corporate Research

France

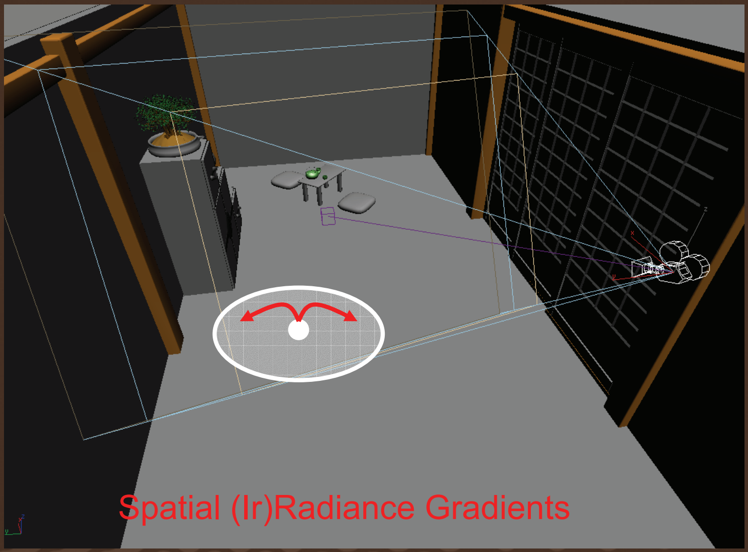


(Ir)Radiance Caching

As explained in the previous parts of this course, the zone of influence of a given irradiance record is defined by a spatial weighting function.

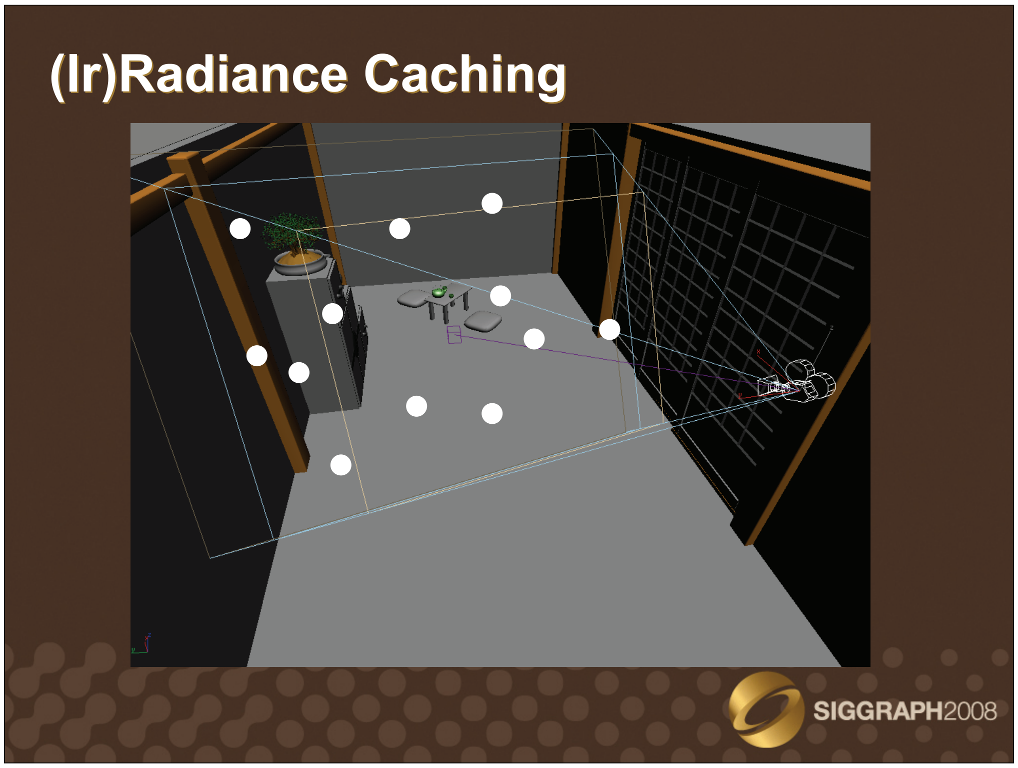


(Ir)Radiance Caching



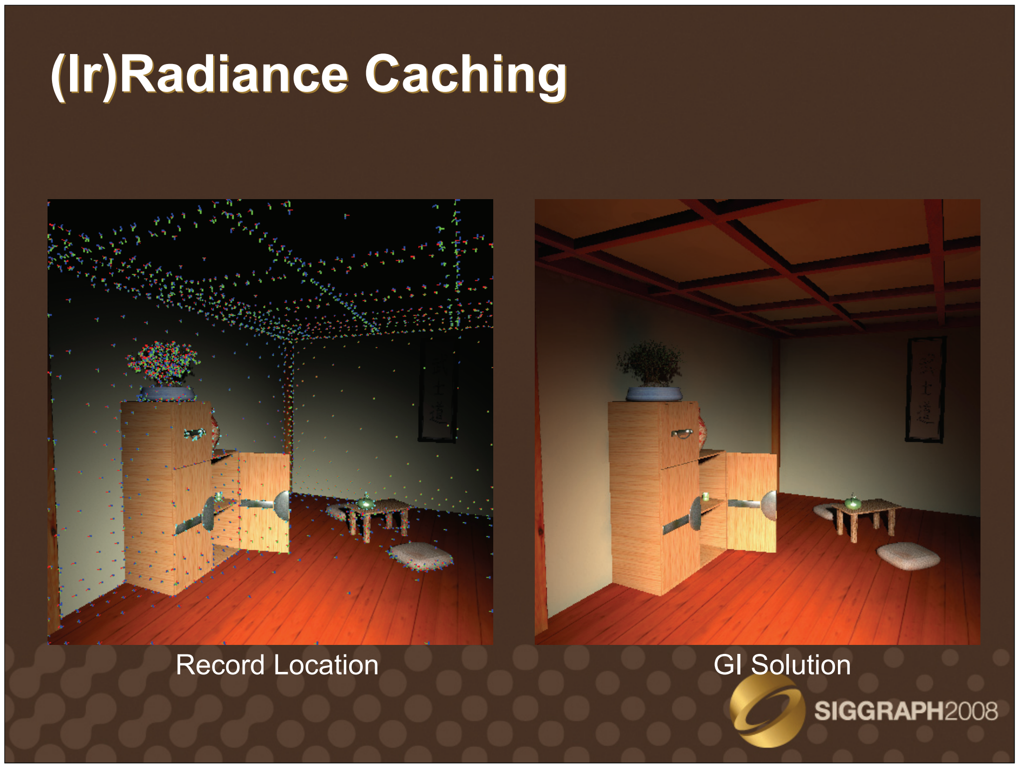
SIGGRAPH2008

The contribution of a record within its zone of influence is estimated using irradiance gradients [Ward92].



(Ir)Radiance Caching

The lighting of the visible points is then computed explicitly at the location of the records, and **extrapolated** for all the other visible points.



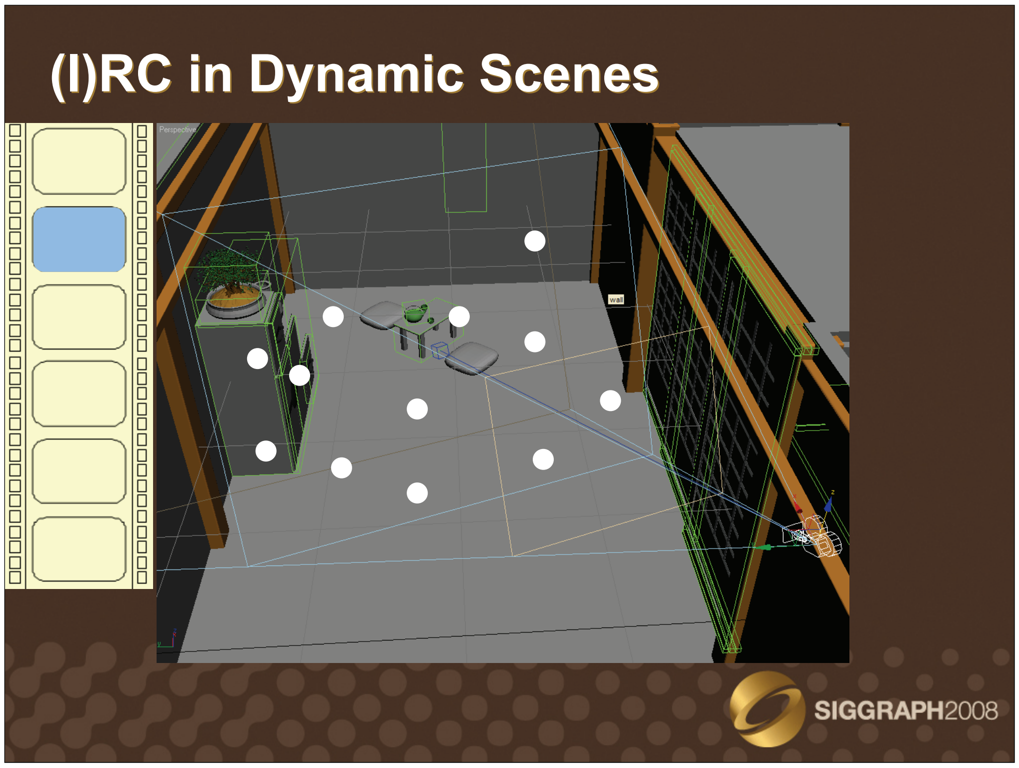
(Ir)Radiance Caching

Record Location

GI Solution

SIGGRAPH2008

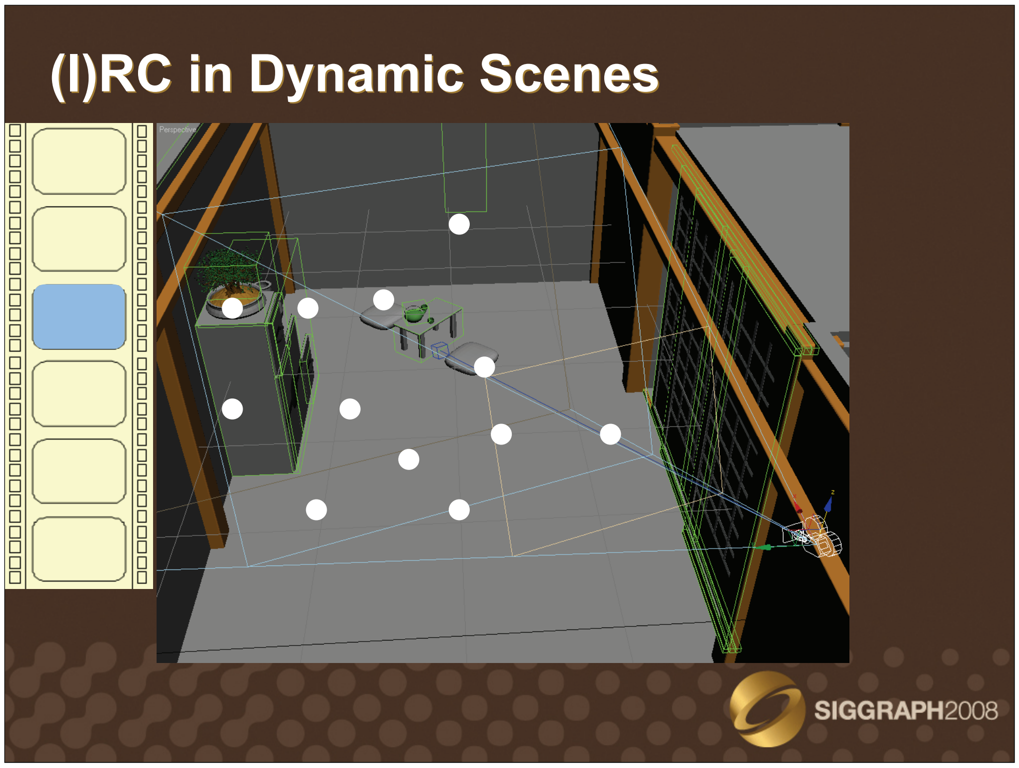
In a single image, this method provides high quality results even when using a very sparse sampling. However, since the estimated lighting is generally obtained through extrapolation from the location of the records, the result depends on the **distribution** of the records.



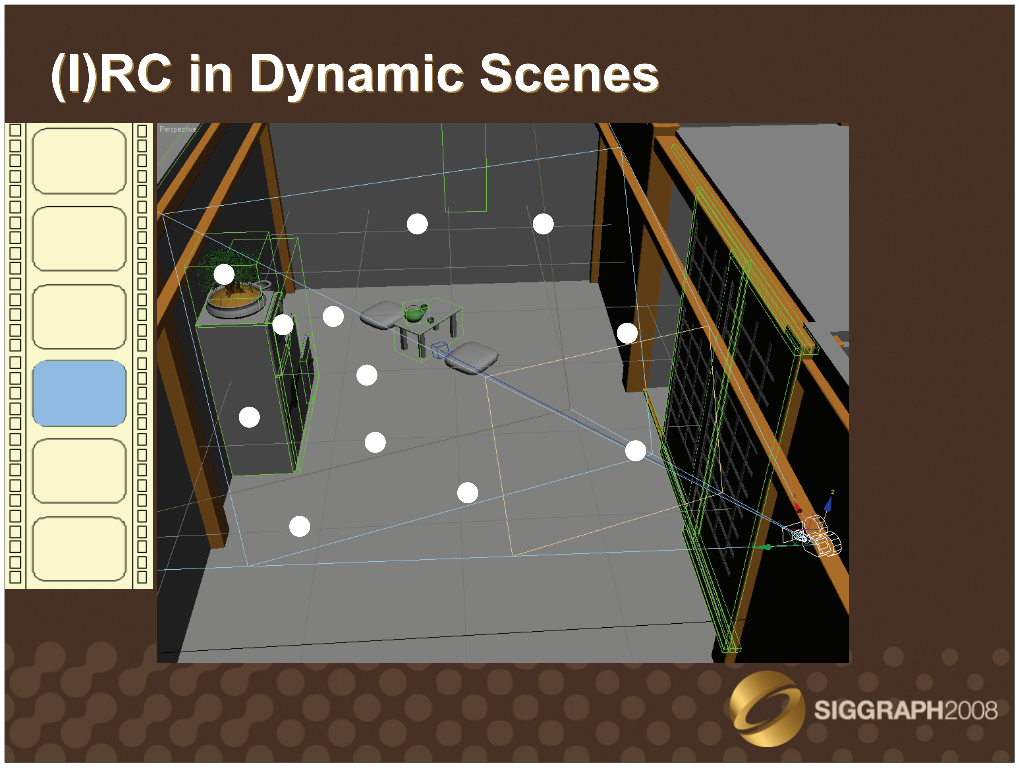
(I)RC in Dynamic Scenes

In dynamic scenes, the indirect lighting must be changes in each frame. A simple method for animation rendering using irradiance caching is the entire recomputation of the global illumination solution for each frame. The camera and objects being dynamic, the distribution of records is likely to change across frames, yielding flickering artifacts. A video illustrating this method can be found on [MyWebSite].

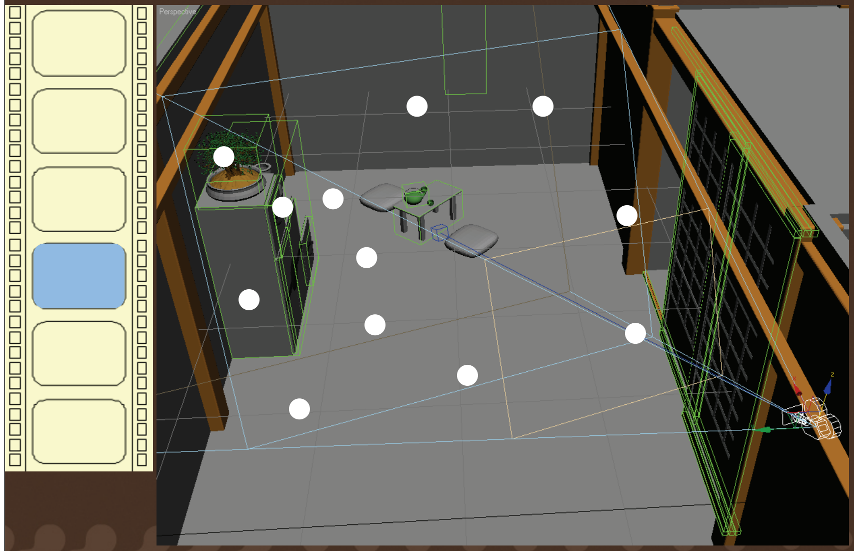
Furthermore, the indirect lighting tends to change slowly across frames. The indirect lighting computed at a given frame may thus be reused in several subsequent frames without degrading the quality. However, the lighting may change very quickly in some areas of the scene, and be nearly-static elsewhere. The method described in this course leverages these observations by assigning a distinct lifespan to each record in the cache.



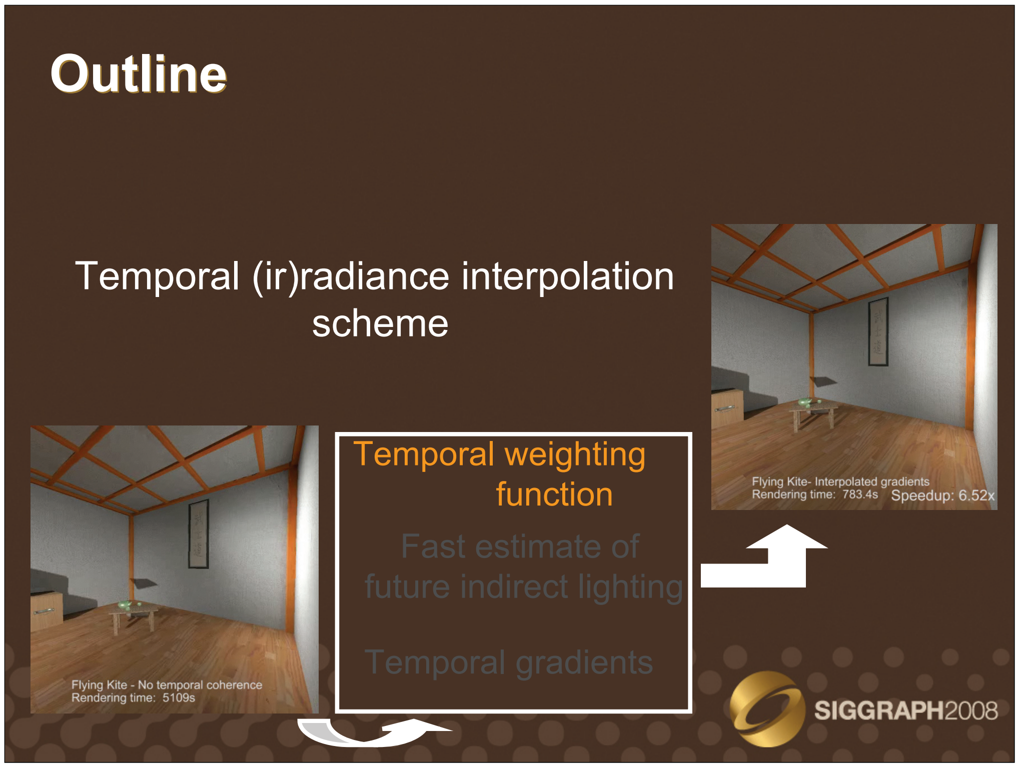
(I)RC in Dynamic Scenes



(I)RC in Dynamic Scenes



SIGGRAPH2008



Outline

Temporal (ir)radiance interpolation scheme

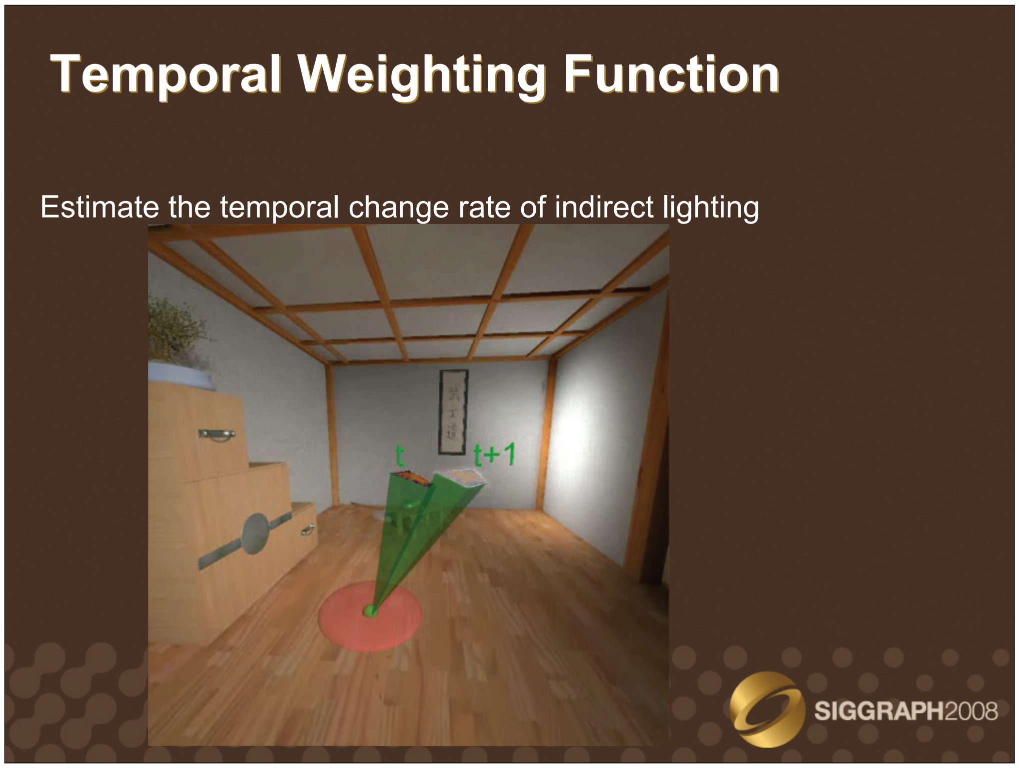
Temporal weighting function

Fast estimate of future indirect lighting

Temporal gradients

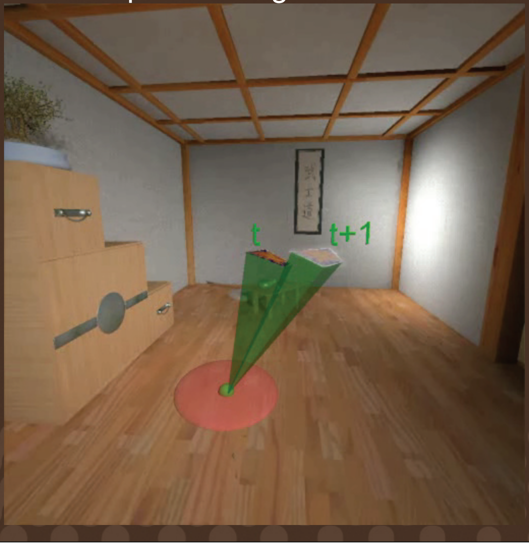
Flying Kite - No temporal coherence
Rendering time: 5109s
Flying Kite- Interpolated gradients
Rendering time: 783.4s
Speedup: 6.52x
SIGGRAPH2008

More precisely, the method described in this course is an extension of the irradiance caching interpolation scheme to the temporal domain. To this end, we devise a temporal weighting function to determine the lifespan of a record, e.g. the number of frames in which a record can be reused without degrading the rendering quality. The contribution of a record within its lifespan is estimated using temporal irradiance gradients.



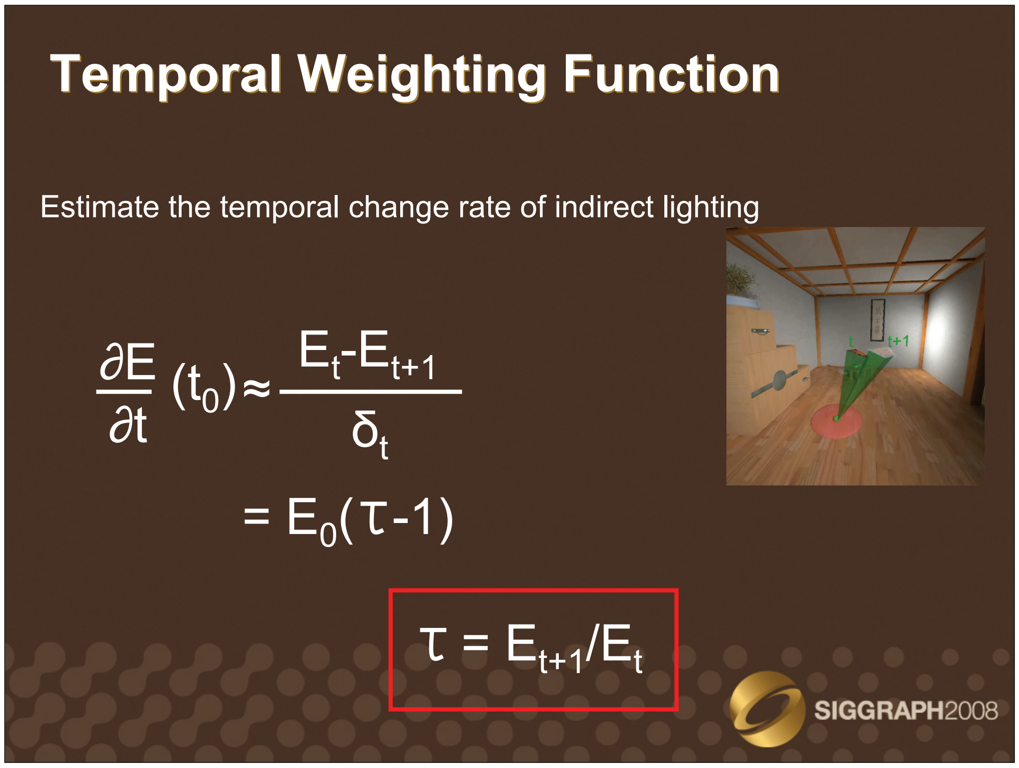
Temporal Weighting Function

Estimate the temporal change rate of indirect lighting



SIGGRAPH2008

The spatial weighting function described in [Ward88] is based on an estimate of the change of indirect lighting with respect to displacement and rotation. The temporal weighting function is thus based on an estimate of the **temporal change** of indirect lighting across frames.



Temporal Weighting Function

Estimate the temporal change rate of indirect lighting

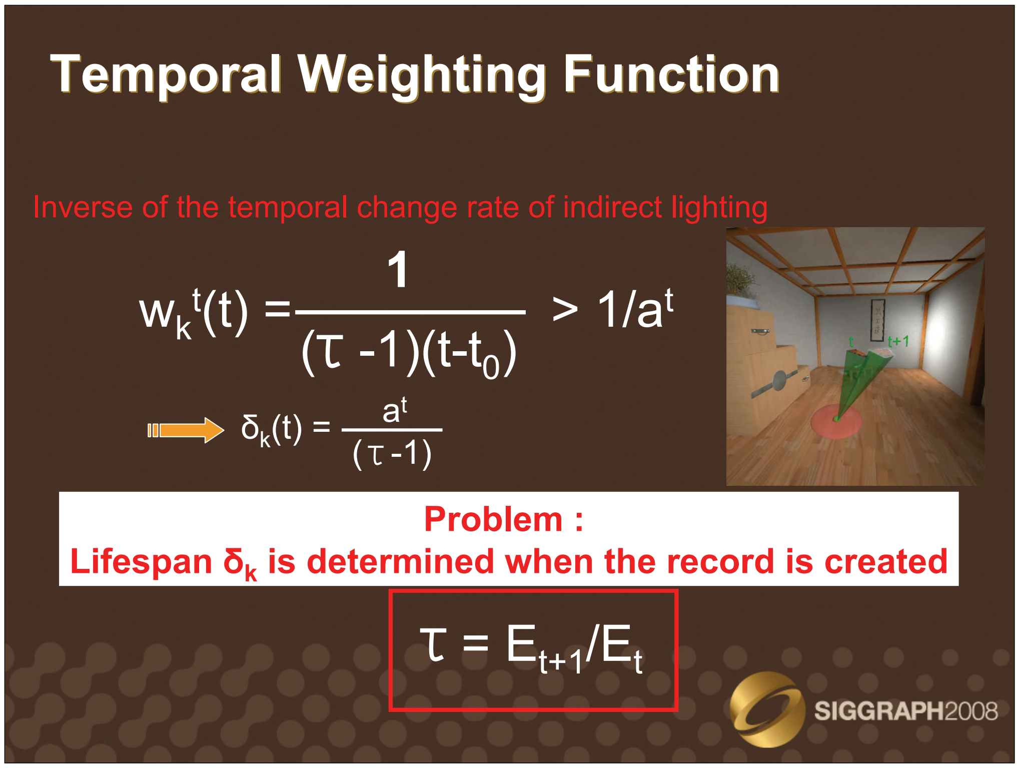
$$\frac{\partial E}{\partial t}(t_0) \approx \frac{E_t - E_{t+1}}{\delta_t}$$

$$= E_0(\tau - 1)$$

$$\boxed{\tau = E_{t+1}/E_t}$$

SIGGRAPH2008

Let us consider the irradiance E_t at current frame, and the irradiance E_{t+1} at next frame. The temporal change of indirect lighting can be estimated by a numerical estimation of the temporal derivative of the lighting. We define the value τ as the ratio of the future and current lighting.



Temporal Weighting Function

Inverse of the temporal change rate of indirect lighting

$$w_k^t(t) = \frac{1}{(\tau - 1)(t - t_0)} > 1/a^t$$

$$\Rightarrow \delta_k(t) = \frac{a^t}{(\tau - 1)}$$

Problem :
Lifespan δ_k is determined when the record is created

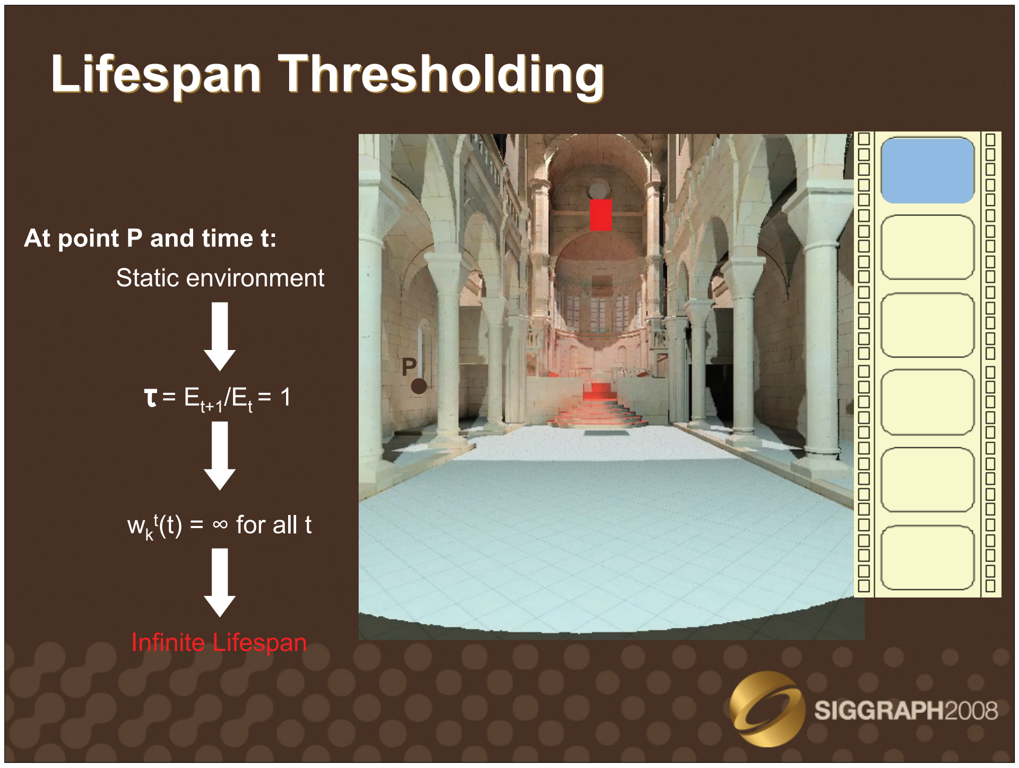
$$\boxed{\tau = E_{t+1}/E_t}$$

SIGGRAPH2008

Using a derivation similar to [Ward88], we obtain a temporal weighting function defined as the inverse of the change of indirect lighting over time. The lifespan of a record is thus adapted to the **local change** of indirect lighting: fast changes yield a short lifespan, while records can be reused across many frames if the temporal changes are slow. The user-defined threshold value a^t conditions the temporal accuracy of the computation: a high value leads to long lifespans, hence reducing the rendering time at the detriment of the quality. Conversely, a small value ensures frequent updates at the expense of rendering time.

At the end of its lifespan, a record is discarded and replaced by a new record located at the same point. This method strenghtens the temporal coherence of the distribution of records, hence avoiding flickering artifacts due to changes of record distribution.

However, the lifespan is determined using the indirect lighting at current and next frames only. Therefore, later changes do not affect this lifespan, hence harming the reactivity of the algorithm. An overestimation of the lifespan yields residual global illumination effects known as « ghosting artifacts ».



Lifespan Thresholding

At point P and time t:

Static environment

↓

$\tau = E_{t+1}/E_t = 1$

↓

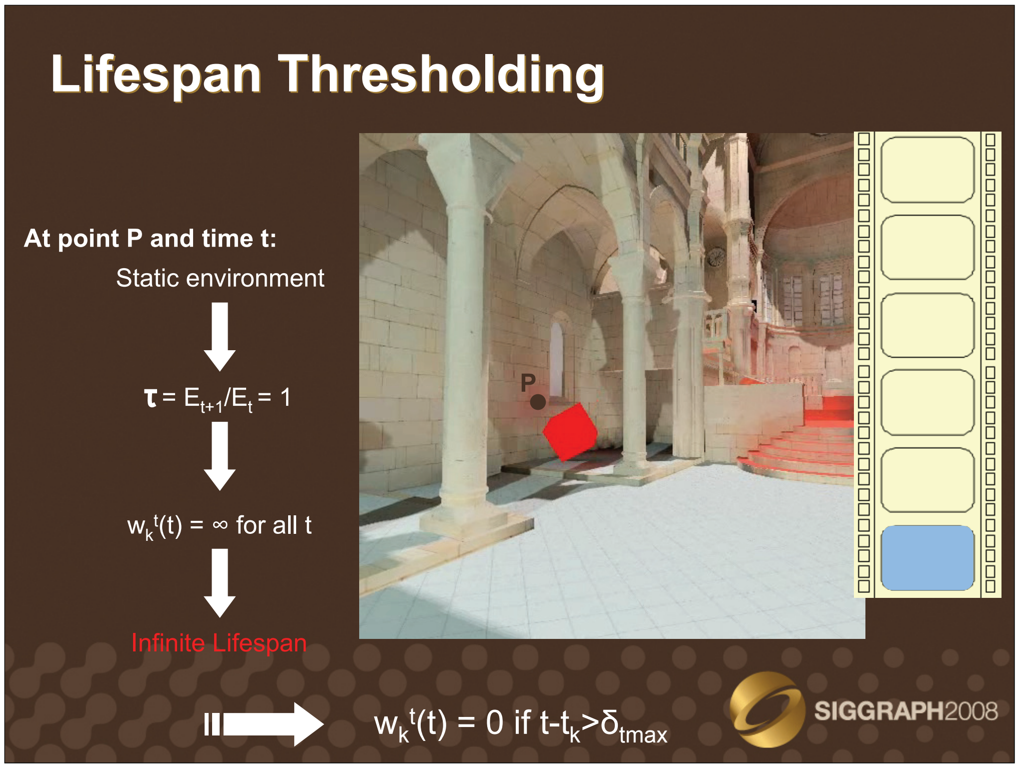
$w_k^t(t) = \infty$ for all t

↓

Infinite Lifespan

P
SIGGRAPH2008

Let us consider an example: at a point P and time t and t+1, the environment is static. Therefore, the indirect lighting is considered constant. In this case, the value of the temporal weighting function is infinite for any frame. The lifespan of the record is then considered infinite.



Lifespan Thresholding

At point P and time t:

Static environment

$\downarrow$

$\tau = E_{t+1}/E_t = 1$

$\downarrow$

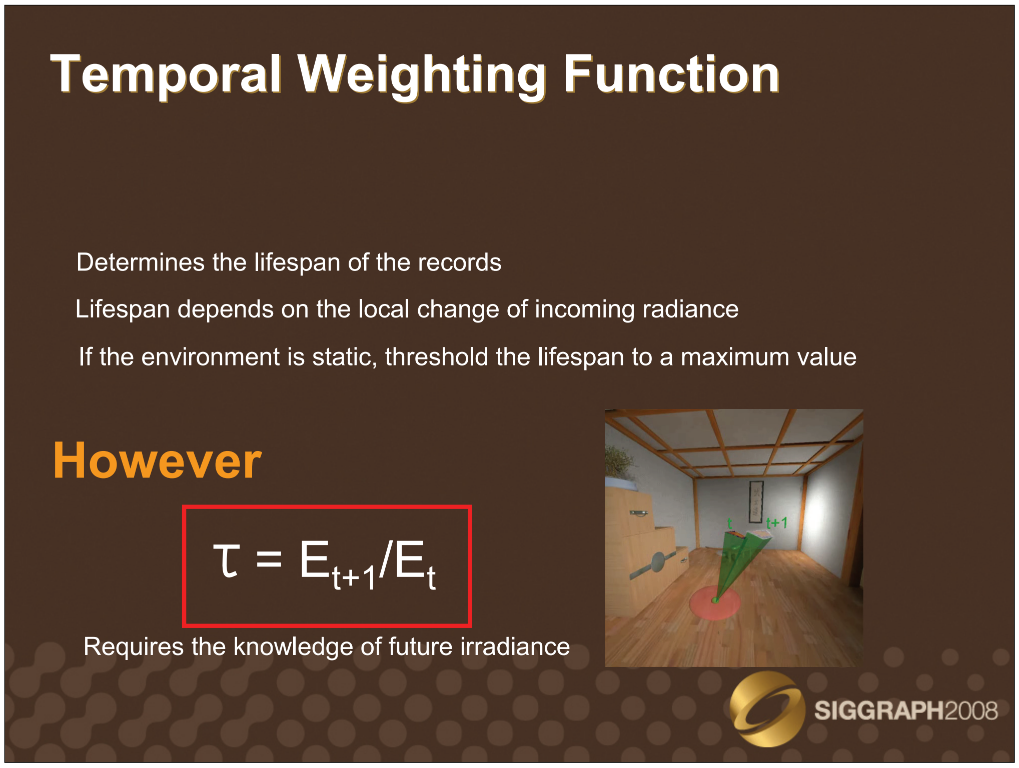
$w_k^t(t) = \infty$ for all t

$\downarrow$

Infinite Lifespan

$\Rightarrow$ $w_k^t(t) = 0$ if $t-t_k>\delta_{tmax}$

However, it is easy to show that the lighting at P may change afterwards. Since the weighting function returns an infinite value regardless of the frame, the value of the lighting at P will never be updated, yielding ghosting artifacts. We do not solve this problem, but we simply ask the user to define a maximum lifespan for all records. This maximum lifespan ensures the update of the ligthing at least after



Temporal Weighting Function

Determines the lifespan of the records

Lifespan depends on the local change of incoming radiance

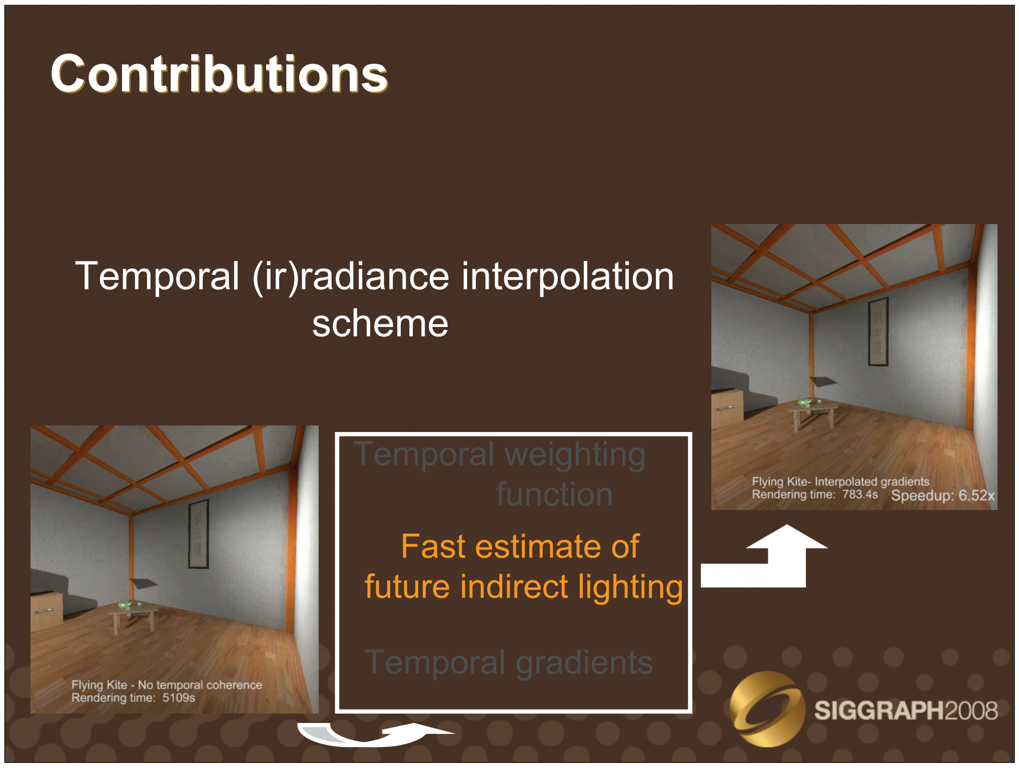
If the environment is static, threshold the lifespan to a maximum value

However

$$\boxed{\tau = E_{t+1}/E_t}$$

Requires the knowledge of future irradiance

The temporal weighting function is used to determine the lifespan of each record based on the local change of incoming radiance over time. The length of the lifespan is shortened or lengthened with respect to the magnitude of the change of indirect lighting. However, the estimate of this change is based on the knowledge of the indirect lighting at next frame, E_{t+1}. Since this value is generally unknown, next section will devise a method for fast estimation of the future incoming without actual sampling.



Contributions

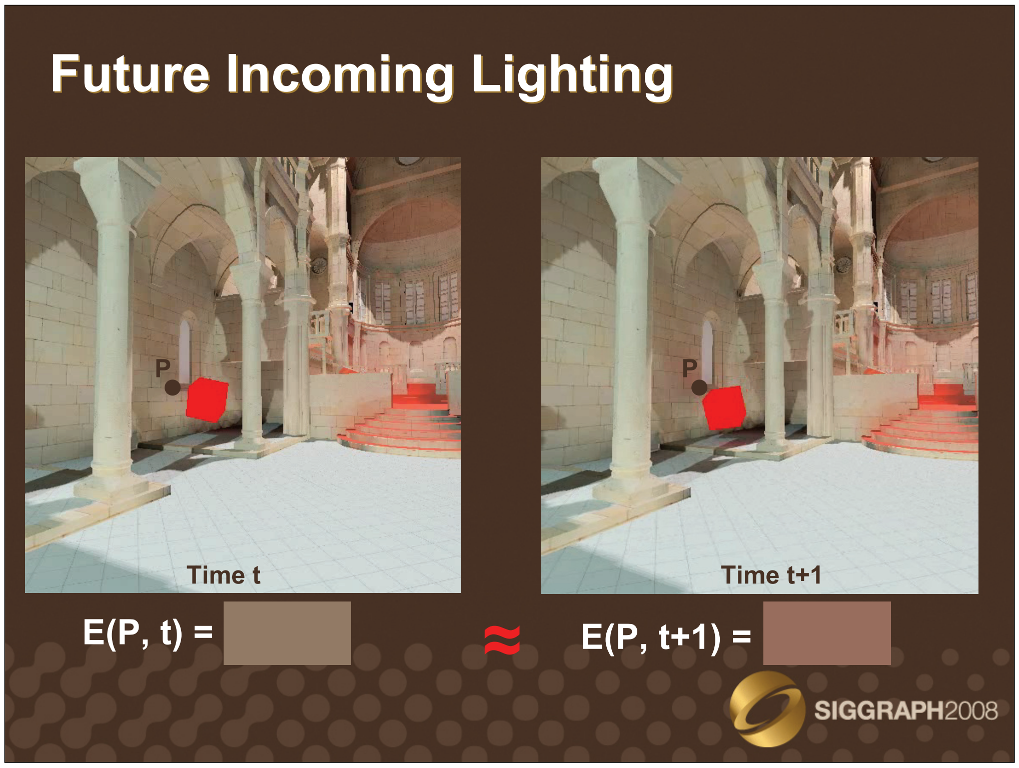
Temporal (ir)radiance interpolation scheme

Temporal weighting function

Fast estimate of future indirect lighting

Temporal gradients

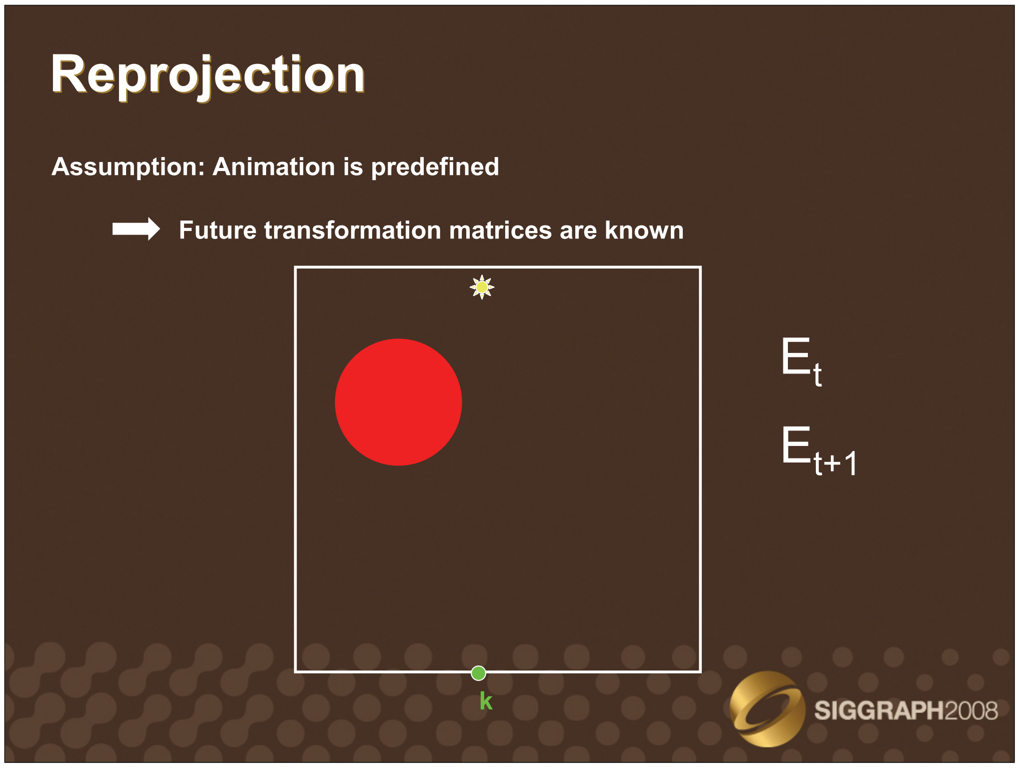
Flying Kite - No temporal coherence
Rendering time: 5109s
Flying Kite- Interpolated gradients
Rendering time: 783.4s
Speedup: 6.52x
SIGGRAPH2008



Future Incoming Lighting

E(P, t) = ▨ ≈ E(P, t+1) = ▨

This approach is based on the following observation: between time t and t+1, the change of lighting is low. Based on the knowledge of the dynamic properties of surrounding objects, we propose a method based on reprojection to estimate the future incoming lighting using only the data available at current frame.



Reprojection

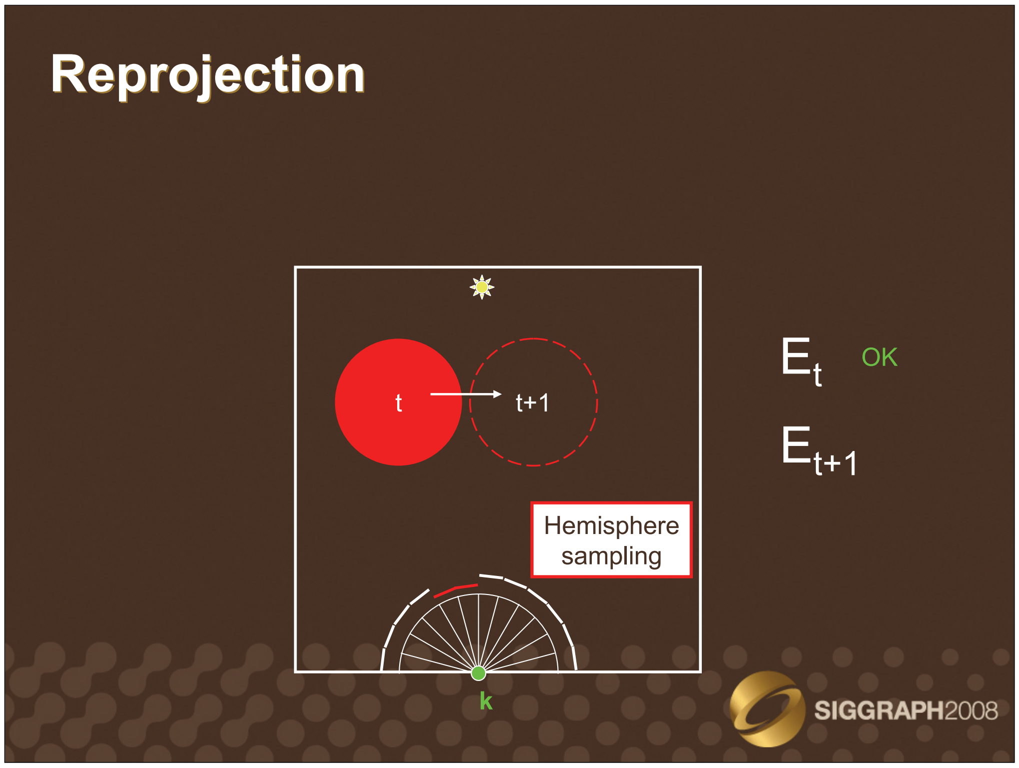
Assumption: Animation is predefined

➡ Future transformation matrices are known

E_t
E_{t+1}
k
SIGGRAPH2008

Our reprojection method is similar to the one used in the Render Cache [Walter99]. However, in the Render Cache, the reprojection is used to avoid tracing primary rays from the camera. In our case, the reprojection is only used to estimate the future incoming radiance at the location of a record.

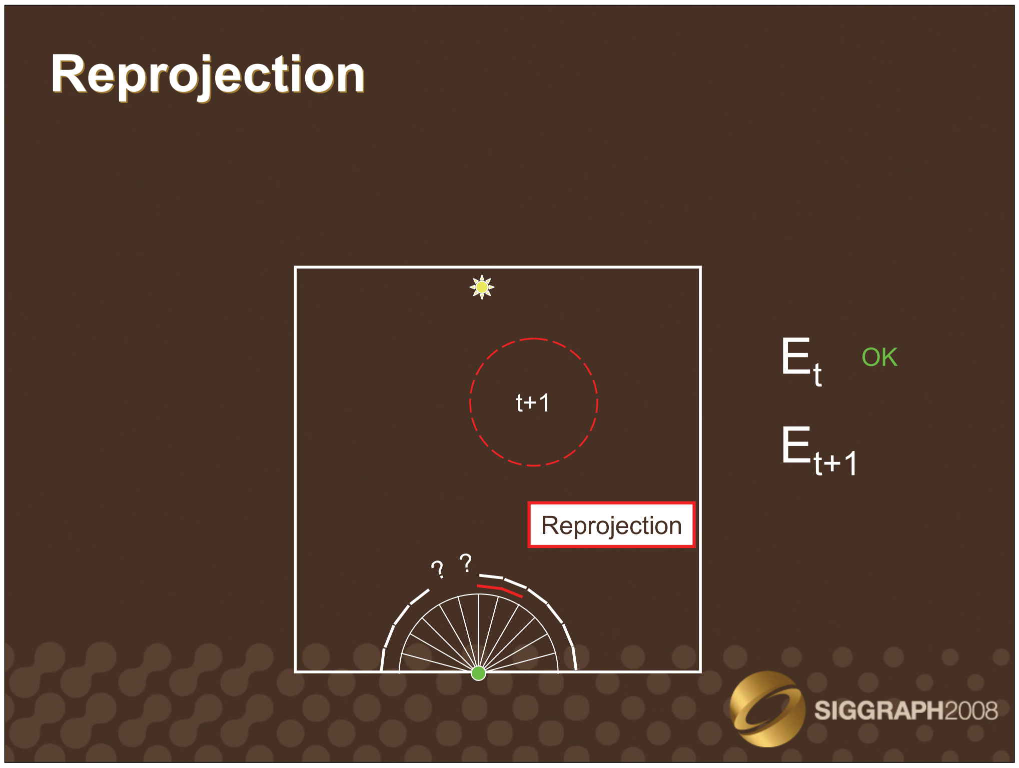
Let us consider a point k at which we want to create a record.



Reprojection

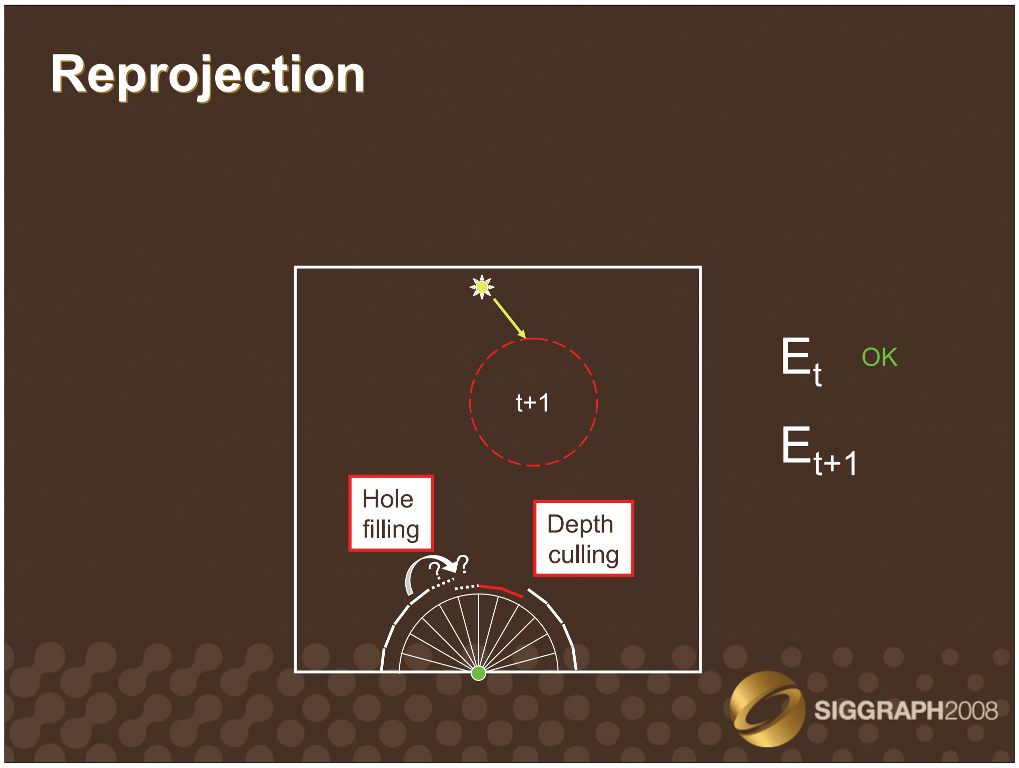
The first step is the computation of the incoming radiance in k at time t. This is performed by sampling the hemisphere above k either using ray tracing or GPU rasterization as explained in the previous chapter.

Assuming the animation is known in advance, the position of the red sphere at time t+1 is known.



Reprojection

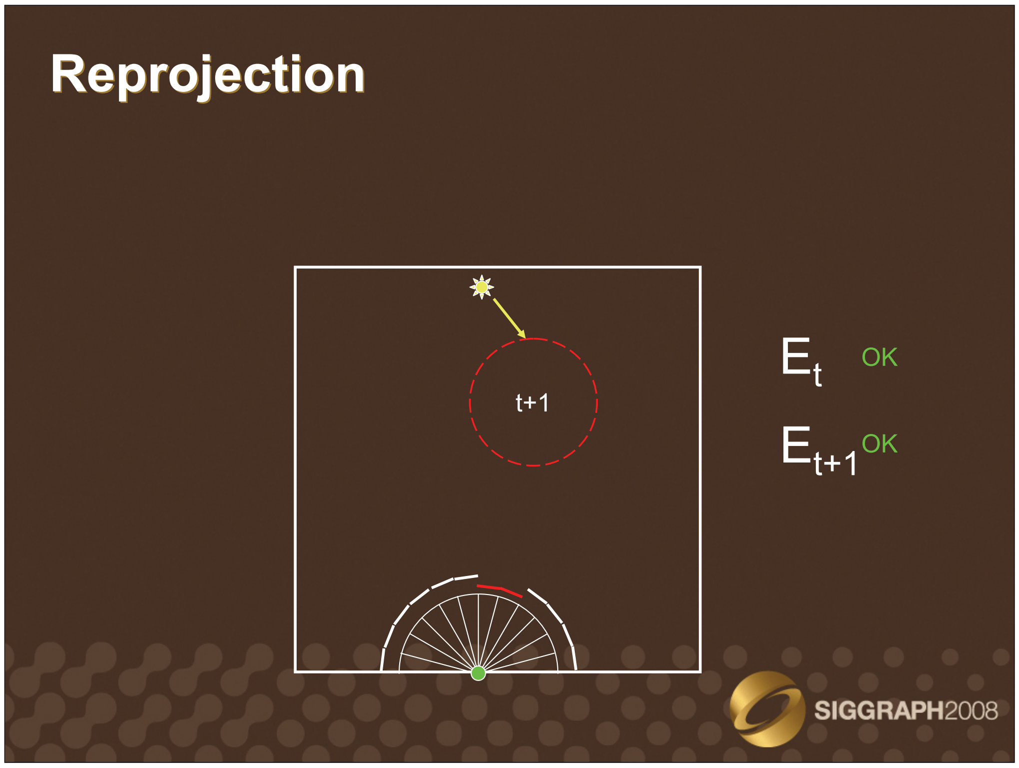
This future position of the red sphere is used to reproject the radiance samples corresponding to the sphere to the new position. This reprojection yields missing information at the former location of the sphere, and overlapping values at the novel position.



Reprojection

A depth culling step chooses the closest value in the case of overlapping radiance values. In this case, the background values are culled.

A simple hole-filling is applied in strata where information is missing. Since we only deal with small movements (1 frame), we simply fill the holes using the closest neighboring value.

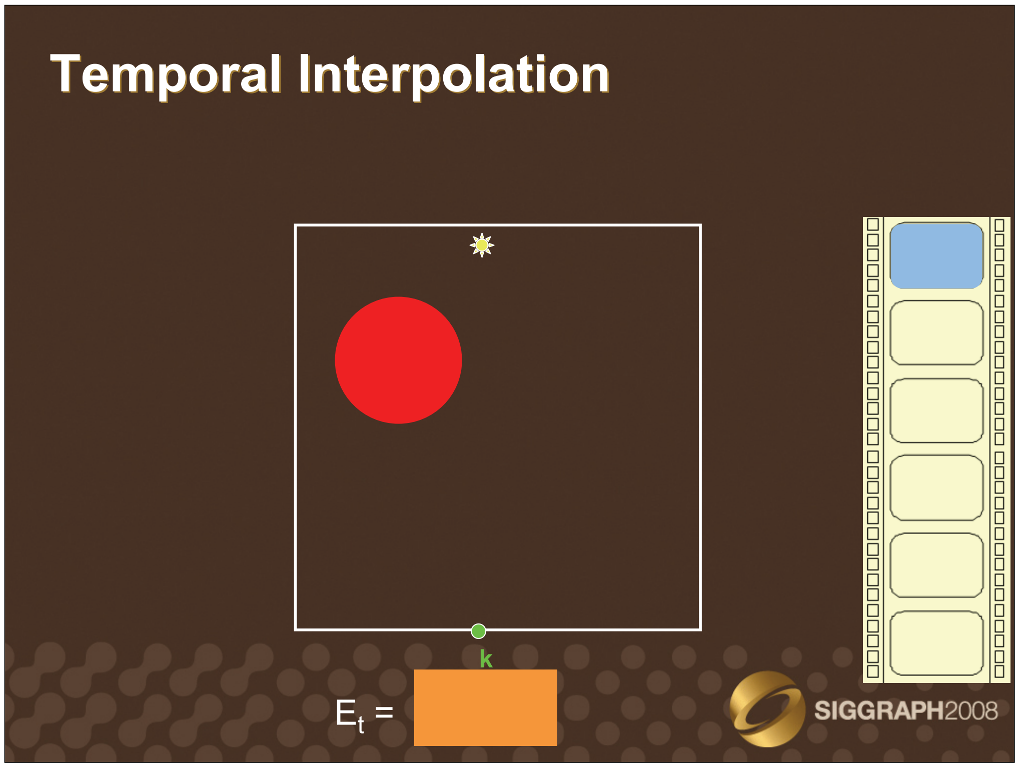


Reprojection

E_t OK

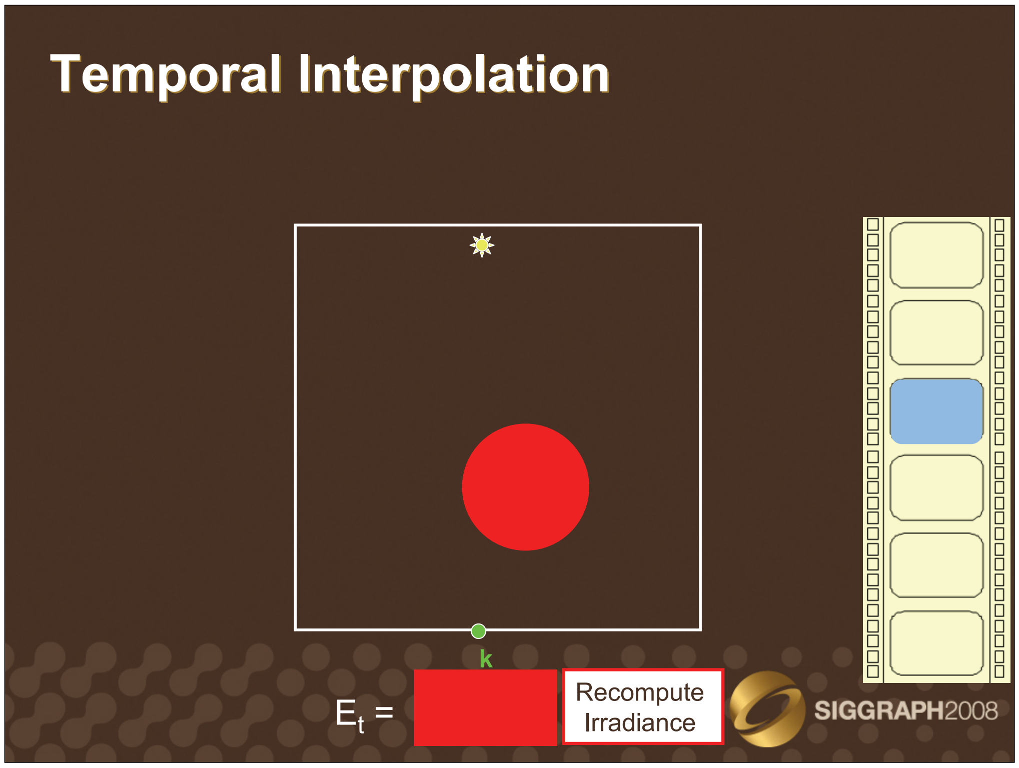
E_{t+1} OK

After depth culling and hole filling, our method provides a reliable estimate of the future incoming lighting. The lifespan of the record created in k is thus completely defined.



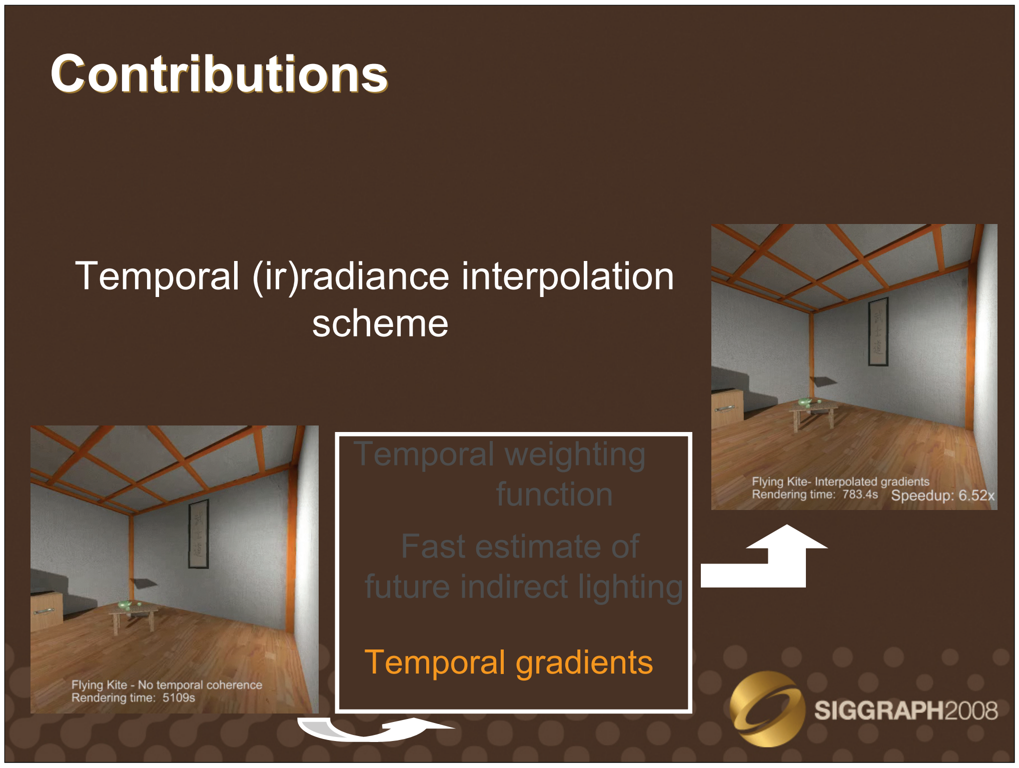
Temporal Interpolation

At this point, we know that the value of record k can be reused across n frames.



Temporal Interpolation

At the end of the record lifespan, a new record is computed, containing an up-to-date irradiance value. In this case the red sphere got closer to k, hence the irradiance at time t and t+n are noticeably different. A consequence of this brutal replacement is sudden changes in the color of image portions, known as « popping » artifacts.



Contributions

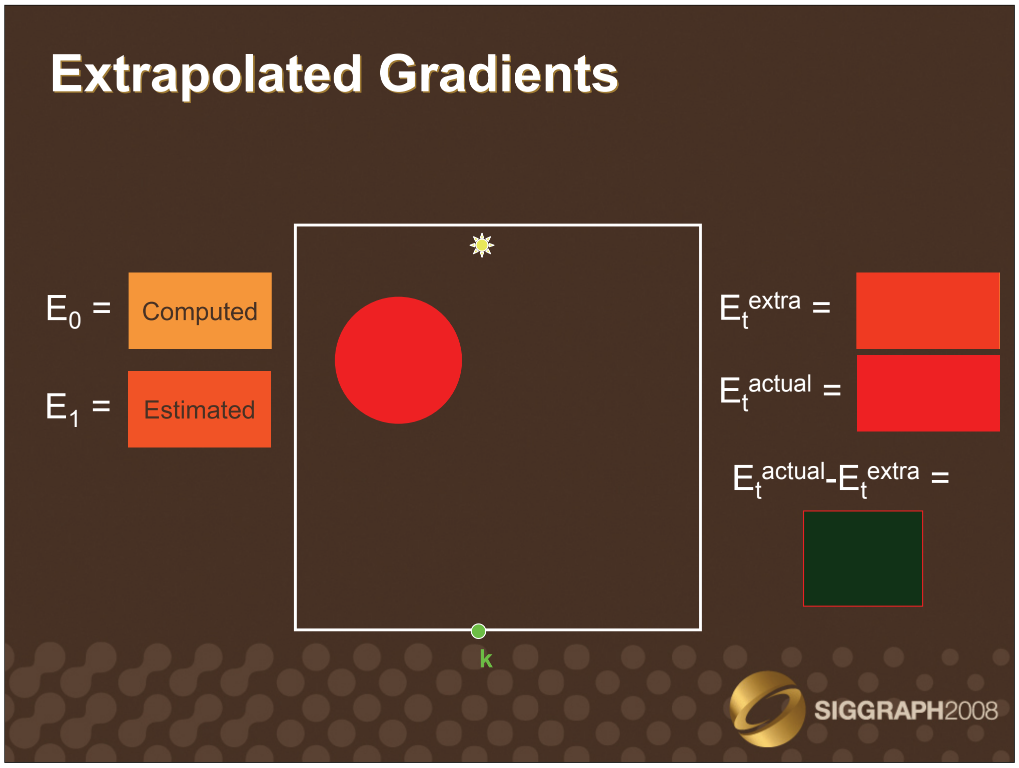
Temporal (ir)radiance interpolation scheme

Temporal weighting function

Fast estimate of future indirect lighting

Temporal gradients

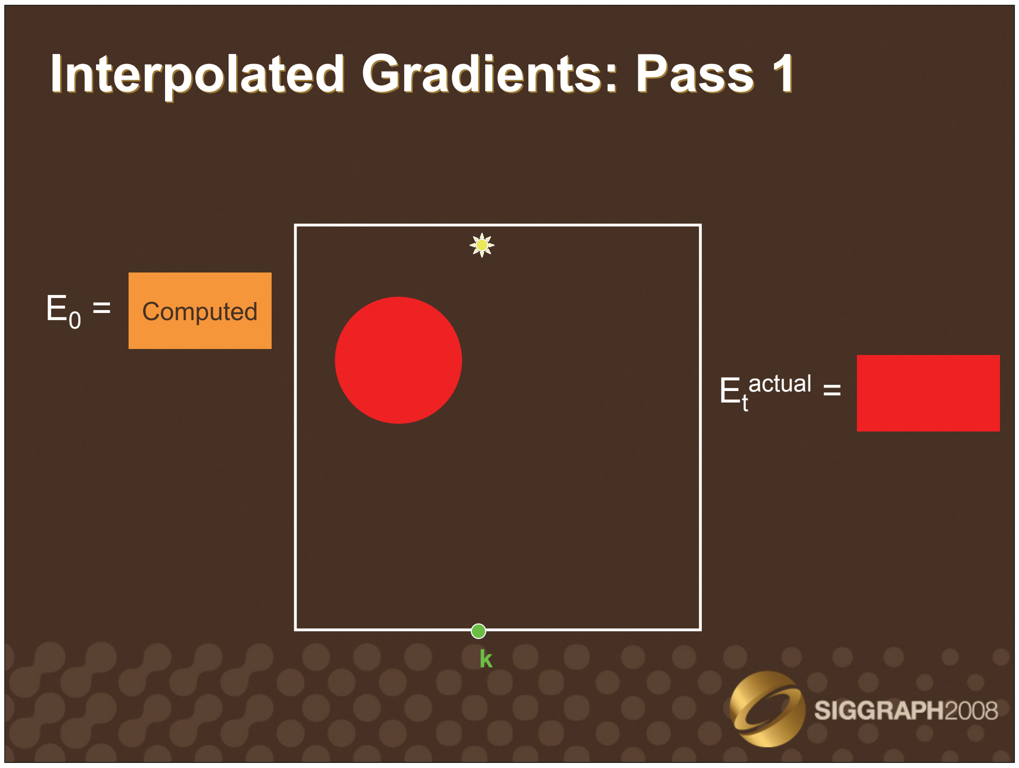
Flying Kite - No temporal coherence
Rendering time: 5109s
Flying Kite- Interpolated gradients
Rendering time: 783.4s
Speedup: 6.52x
SIGGRAPH2008



Extrapolated Gradients

First, we propose to use the estimate of the future incoming lighting to extrapolate the lighting over the entire lifespan of the record. While this method reduces the gap between the extrapolated and the actual irradiance values, the difference is still not negligible. As a consequence, some popping artifacts will remain visible.

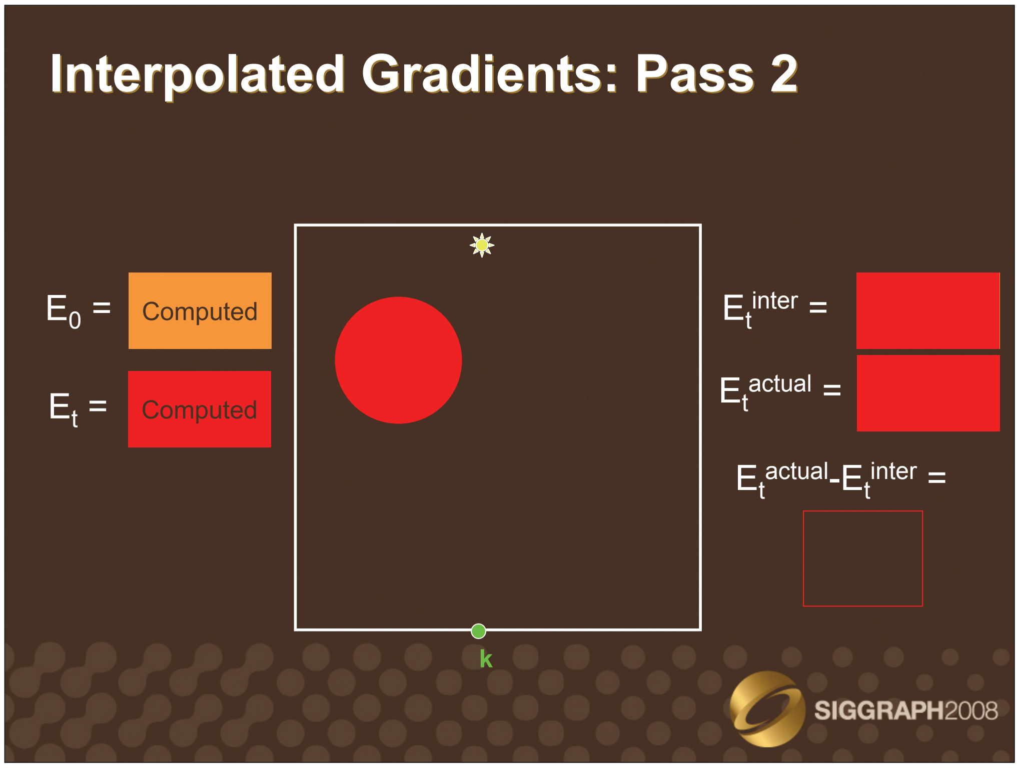
This method has one major advantage: the indirect lighting can be computed and displayed on the fly, as the animation is played. Particularly, such gradients could be used in the context of interactive global illumination computation.



Interpolated Gradients: Pass 1

Interpolated gradients completely avoid the popping artifacts by temporally interpolating the irradiance. In a first pass, records are generated as explained previously: each record is used within its lifespan, and new records are computed as the lifespans expire. Let us consider a record R_0 computed at time t_0 and located at point k. When this record expires after n frames, it is replaced by a new record R_1 also located at point k. The temporal gradient of R_0 is then deduced from $(E_{R1}-E_{R0})/n$.

This gradient approximates the change of lighting within the lifespan of a record by linearly interpolating the irradiance at the beginning and at the end of the lifespan. While completely removing popping artifacts, this linear approximation may smooth out high frequency changes that may happen during the lifespan of the record. Accounting for such changes either require a reduction of a^t and δ_{tmax}, or the definition of a higher order interpolation scheme.



Interpolated Gradients: Pass 2

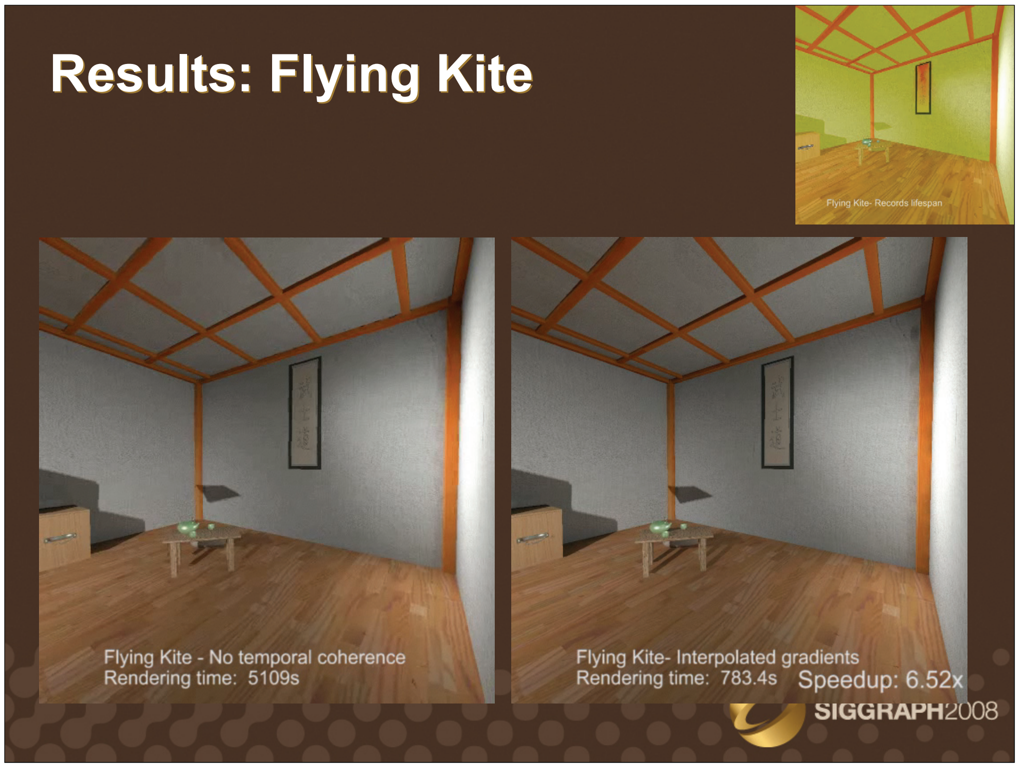
$E_0 =$ Computed

$E_t =$ Computed

$E_t^{inter} =$

$E_t^{actual} =$

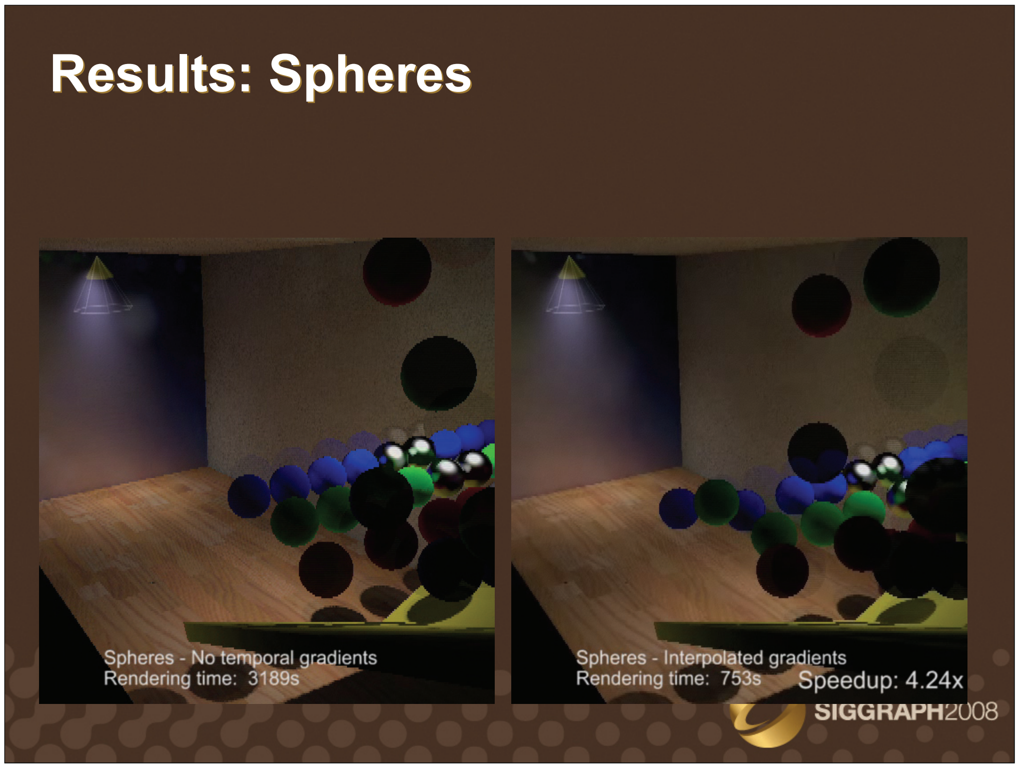
$E_t^{actual} - E_t^{inter} =$



Results: Flying Kite

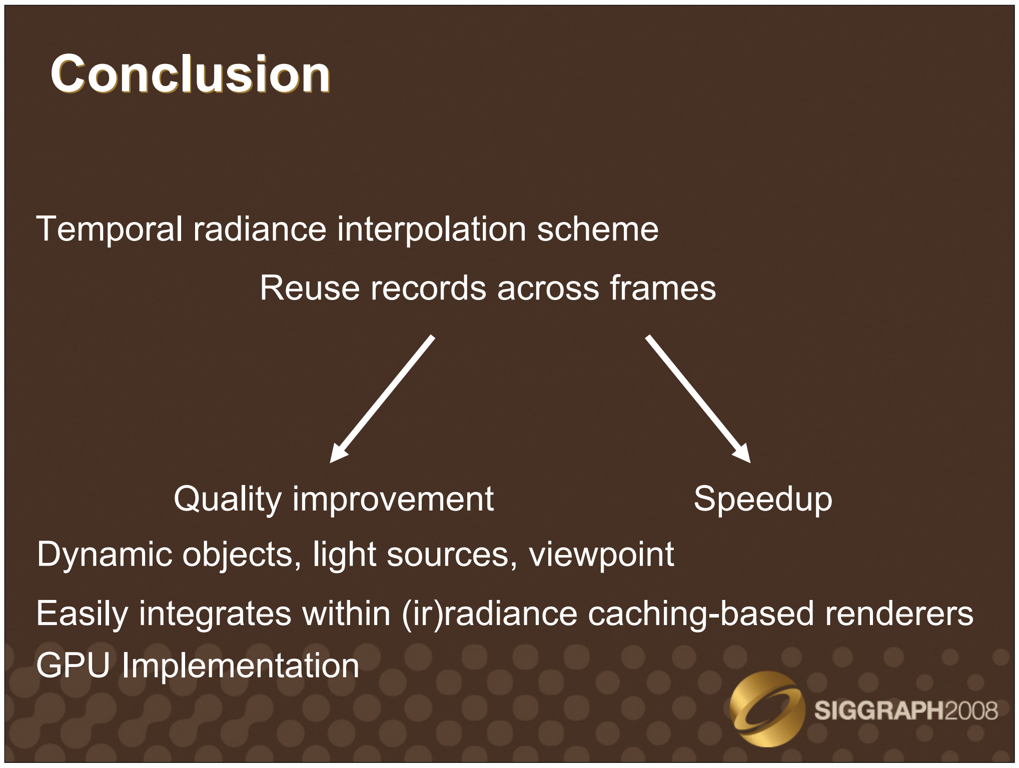
Flying Kite- Records lifespan
Flying Kite - No temporal coherence
Rendering time: 5109s
Flying Kite- Interpolated gradients
Rendering time: 783.4s
Speedup: 6.52x
SIGGRAPH2008

Videos illustrating the interpolated gradients and the adaptive record lifespan in different scenes can be found in [MyWebSite].



Results: Spheres

This method can be straightforwardly extended to nondiffuse interreflexions using radiance caching.



Conclusion

Temporal radiance interpolation scheme

Reuse records across frames

Quality improvement — Speedup

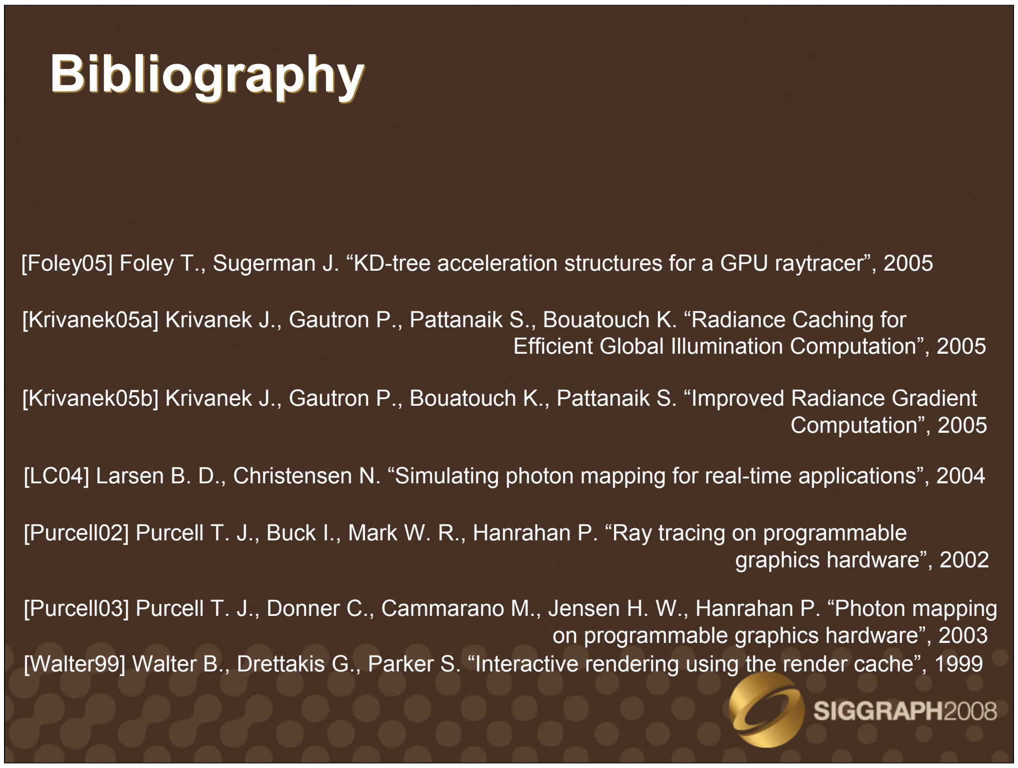
Dynamic objects, light sources, viewpoint

Easily integrates within (ir)radiance caching-based renderers

GPU Implementation

The method described in this part is based on the reuse of irradiance records across frames. While reducing the computational cost of animation rendering, the flickering artifacts are drastically reduced. This method supports any type of dynamic scene components, and can be easily integrated in existing renderers.

Future work will consider the elimination of the maximum lifespan δ_{tmax}. Also, we would like to devise methods for higher order temporal interpolation to account for sharp changes of indirect lighting.



Bibliography

[Foley05] Foley T., Sugerman J. “KD-tree acceleration structures for a GPU raytracer”, 2005

[Krivanek05a] Krivanek J., Gautron P., Pattanaik S., Bouatouch K. “Radiance Caching for Efficient Global Illumination Computation”, 2005

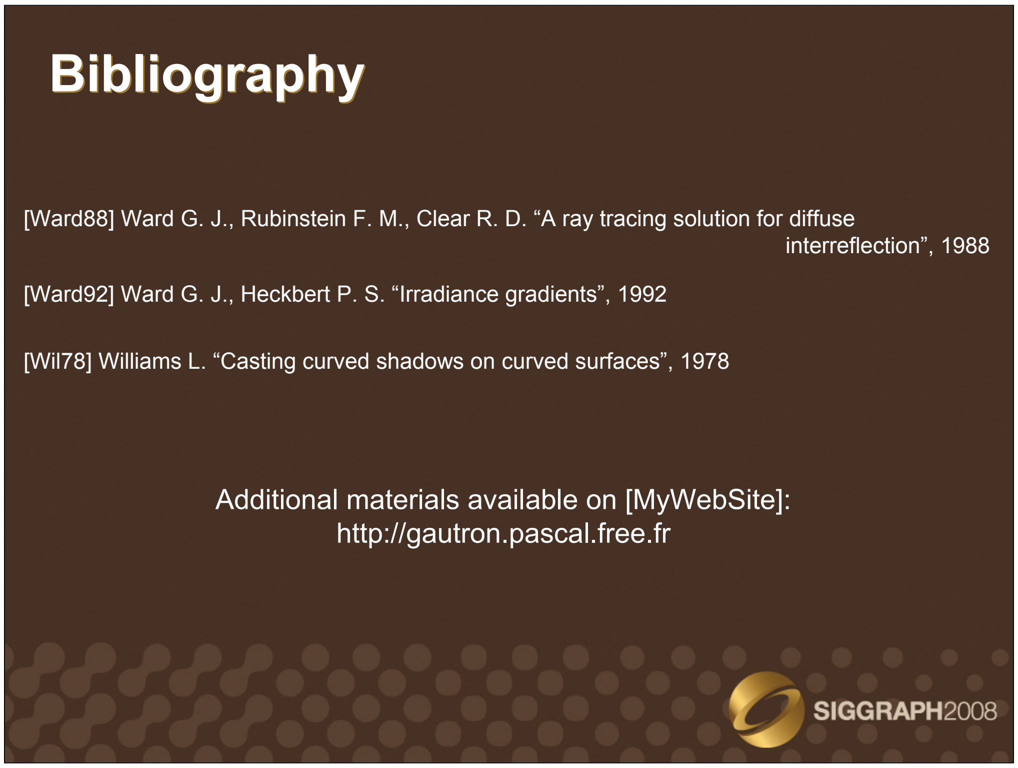
[Krivanek05b] Krivanek J., Gautron P., Bouatouch K., Pattanaik S. “Improved Radiance Gradient Computation”, 2005

[LC04] Larsen B. D., Christensen N. “Simulating photon mapping for real-time applications”, 2004

[Purcell02] Purcell T. J., Buck I., Mark W. R., Hanrahan P. “Ray tracing on programmable graphics hardware”, 2002

[Purcell03] Purcell T. J., Donner C., Cammarano M., Jensen H. W., Hanrahan P. “Photon mapping on programmable graphics hardware”, 2003

[Walter99] Walter B., Drettakis G., Parker S. “Interactive rendering using the render cache”, 1999



Bibliography

[Ward88] Ward G. J., Rubinstein F. M., Clear R. D. “A ray tracing solution for diffuse interreflection”, 1988

[Ward92] Ward G. J., Heckbert P. S. “Irradiance gradients”, 1992

[Wil78] Williams L. “Casting curved shadows on curved surfaces”, 1978

Additional materials available on [MyWebSite]:
http://gautron.pascal.free.fr

Chapter 8

Temporal Radiance Caching

Very often global illumination methods aim at simulating the light/matter interactions within static scenes. Accounting for the displacement of objects and light sources either require a complete recomputation of the global illumination solution for each frame of the animation, or involve complex data structures and algorithms for temporal optimization. Furthermore, the global illumination solutions commonly exhibit low temporal quality when used in dynamic scenes: flickering, popping, ... In the context of computer-assisted effects for movies, high quality global illumination is obtained through temporal filtering: a 30 fps animation is first rendered at 60 fps by recomputing the global illumination for each frame. Then, each frame of the 30 fps animation is generated by averaging two frames of the 60 fps animation, hence reducing the temporal artifacts at the cost of high computational cost. For interactive applications such as video games, the illumination must be computed interactively. In this case, approximate models are generally preferred, such as the precomputation of a static global illumination solution, and the update of direct lighting only at runtime.

This chapter describes a simple and accurate method based on temporal caching for the computation of global illumination effects in animated environments [GBP07], where viewer, objects and light sources move. This approach focuses on a temporal optimization for lighting computation based on the irradiance caching [WRC88] technique. As this algorithm leverages the spatial coherence of indirect lighting to reduce the cost of global illumination, we consider here an extension of these methods for sparse temporal sampling and interpolation. In [WRC88], Ward *et al.* propose a reuse an irradiance value in the neighborhood of the actual computation point. While the weighting function and gradients of [WRC88] account for the spatial change of irradiance, Temporal Radiance Caching considers the temporal change of the indirect lighting (Figure 8.2).

8.1 Temporal Irradiance Caching

8.1.1 Irradiance Caching in Dynamic Scenes

The irradiance caching algorithm leverages the spatial coherency of the indirect lighting, and thus reduces the computation time of global illumination. In dynamic scenes, a straightforward and commonly used method consists in computing the global illumination solution from scratch for every frame. Thus, the distributions of record location in frames n and $n+1$ are likely to be different. Figure 8.1 illustrates the consequence of the change of record distribution: since the gradients extrapolate the change of incoming radiance, they are not completely accurate compared to the ground truth. Therefore, the accuracy of the lighting reconstructed by irradiance caching is not constant over a surface: the accuracy is maximum at the record location, and decreases as the extrapolation point gets away from the record. Changing the distribution of

P. Gautron

records also changes the distribution of the accuracy, yielding flickering artifacts. Thus, simple use of irradiance caching in dynamic scenes gives rise to poor animation quality. Additionally, as the record computation is performed from scratch for every frame, a significant amount of computational effort is wasted.

The remainder of this chapter describes a simple and unified framework for view-dependent global illumination in dynamic environments with predefined animation in which objects, light sources, and cameras can move.

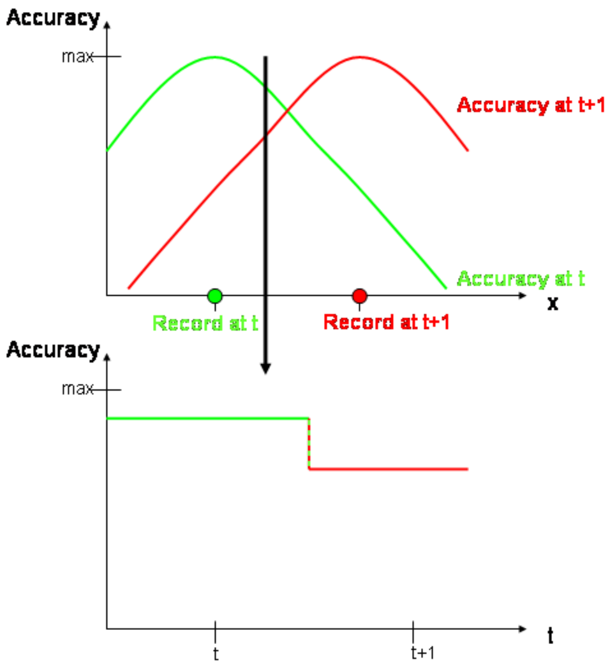
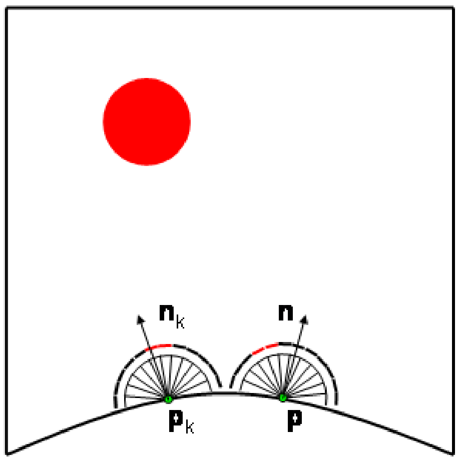


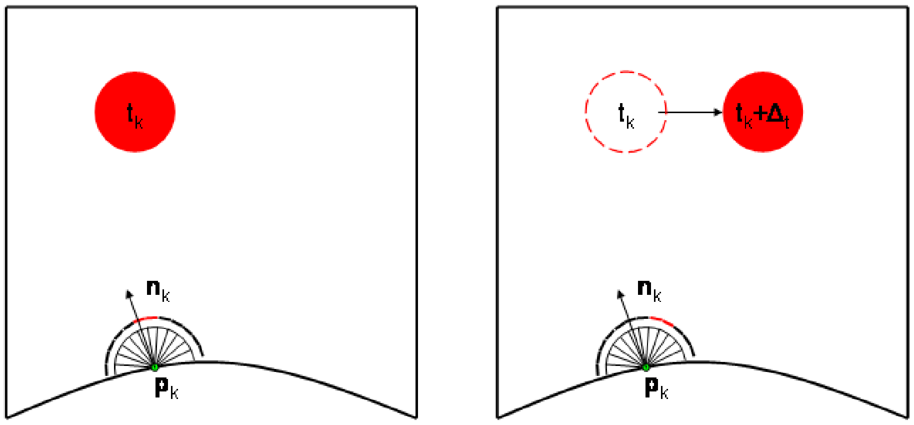
Figure 8.1: Changing the location of the records between successive frames can significantly modify the accuracy of the lighting at a given point (marked by the arrow). Therefore, the incoming radiance at this point can change noticeably between two frames, yielding flickering artifacts. Note that the maximum accuracy is obtained at the point and time of actual computation of the incoming radiance.

8.1.2 Overview of Temporal Radiance Caching

Following the spatial interpolation scheme of Ward *et al.* [WRC88, WH92], our aim is to reuse records across frames by performing a sparse temporal sampling and temporal interpolation of the irradiance (Algorithm 1). When a record k is created at frame n, the future incoming lighting is estimated. This estimate is first used to compute the temporal change rate of the irradiance. In the spirit of [WRC88], the temporal weighting function w_k^t is defined as the inverse of the temporal change. The number of frames in which k can contribute is then inversely proportional to the future change of incoming radiance. Since the actual computation of the future incoming lighting may be expensive, we use a simple reprojection technique for estimating the future lighting using the data sampled at the current frame. As the irradiance at a point is extrapolated using the irradiance and gradients of neighboring records, the temporal caching scheme also features temporal gradients for smooth temporal interpolation and extrapolation of the irradiance.



(a) Spatial change



(b) Temporal change

Figure 8.2: Change of incoming radiance with respect to space and time. The black-red zone above the hemisphere represents the incoming radiance function. Classical irradiance caching (a) estimate the change of incoming radiance with respect to displacement and normal divergence. In the case of temporal irradiance caching (b), the incoming radiance can change between two frames even though the record location remains constant.

8.1.3 Temporal Weighting Function

The temporal weighting function expresses the validity of a given record over time. Using a derivation similar to that of [WRC88], the temporal change ϵ^t of incoming radiance between time t and t_0 is:

$$\epsilon^t = \frac{\partial E}{\partial t}(t_0) \quad (t - t_0) \tag{8.1}$$

This derivative $\frac{\partial E}{\partial t}(t_0)$ can be approximated using estimates of incoming radiance at two successive times t_0 and t_1, denoted E_0 and E_1.

$$\frac{\partial E}{\partial t}(t_0) \approx \frac{E_1 - E_0}{t_1 - t_0} \tag{8.2}$$

$$= \frac{\tau E_0 - E_0}{t_1 - t_0} \quad \text{where } \tau = E_1/E_0 \tag{8.3}$$

Algorithm 1 Temporal Radiance Caching

```
for all frames $n$ do
  for all existing records $k$ do
    if $w_k^t(n)$ is sufficient then
      Use $k$ in frame $n$
    end if
  end for
  for all points $\mathbf{p}$ where a new record is needed do
    Sample the hemisphere above $\mathbf{p}$
    Estimate the future incoming lighting (Section 8.1.4)
    Generate $w_k^t$ (Section 8.1.3)
    Compute the temporal gradients (Section 8.1.5)
    Store the record in the cache
  end for
end for
```

$$= \quad E_0 \frac{\tau - 1}{t_1 - t_0} \tag{8.4}$$

The time range of the animation is discretized into integer frame indices. Therefore, we always choose $t_1 - t_0 = 1$, i.e. E_1 and E_0 represent the estimated irradiance at two successive frames.

As in [WRC88], the temporal weighting function is the inverse of the change, excluding the term E_0:

$$w_k^t(t) = \frac{1}{(\tau - 1)(t - t_0)} \tag{8.5}$$

where $\tau = E_1/E_0$ is the *temporal irradiance change rate.*

This weighting function expresses the confidence of the incoming radiance value estimated at time t_0 in subsequent frames. This function can be evaluated and tested against a user-defined accuracy value a^t. A record k created at t_0 is allowed to contribute to the image at t if

$$w_k^t(t) \geq 1/a^t \tag{8.6}$$

The temporal weighting function is used to adjust the time segment during which a record is considered as valid. Since a given record can be reused in several frames, the computational cost can be significantly reduced. The temporal radiance change rate is defined as:

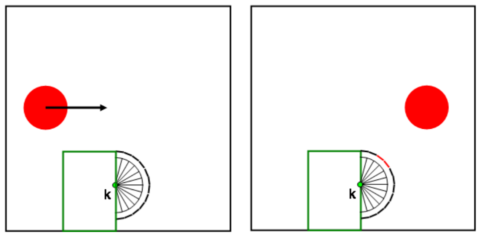
$$\tau_{radiance} = max\{\lambda_1^i/\lambda_0^i, 0 \geq i < n\} \tag{8.7}$$

where λ_0^i and λ_1^i are respectively the i^{th} projection coefficient of the current and future incoming lighting. The maximum rate of change is used to account for directional change of incoming radiance without loss of accuracy.

However, Equation 8.3 shows that if the environment remains static starting from frame t_0, we obtain $\tau = 1$. Therefore, Equation 8.6 shows that w_k^t is infinite for any frame, and hence record k is allowed to contribute at any time $t > t_0$. However, since part of the environment is dynamic, the inaccuracy becomes significant when $t - t_0$ gets high (see Figure 8.3). This is a limitation of this technique for estimating the temporal change of incoming radiance, which determines the lifespan of a record by only considering the change between E_t and E_{t+1}. A user-defined value δt_{max} limits the length of the validity time segment associated with each record to a reasonable, *ad hoc* value, such as 5 to 10 frames depending on the scene. If Equation 8.8 does not hold, we decide that the record cannot be reasonably used.

$$t - t_0 < \delta_{t_{max}} \tag{8.8}$$

This reduces the risk of having records from being used while they are obsolete, hence controlling the artifacts due to residual global illumination effects also known as “ghosts”, which commonly appear in interactive methods. However, as a and a^T, this value must be user-defined by trial and error to obtain the best results. If $\delta_{t_{max}}$ is too low, many records may be recomputed unnecessarily. Setting $\delta_{t_{max}} = 1$ implies the recomputation of each record for each frame. In this case, the resulting performance would be similar to the classical per-frame computation. Nevertheless, this would completely avoid the ghosting artifacts, the indirect lighting being permanently computed from scratch. If $\delta_{t_{max}}$ is set too high, the same records might be reused in too many frames. Hence artifacts due to the residual global illumination effects are likely to appear in the vicinity of the moving objects, degrading the quality of the rendered frame. Hence such a high value significantly reduces the rendering time at the cost of accuracy.



(a) Time t_k (b) Time $t_k + n$

Figure 8.3: When record k is created at time t_k, the surrounding environment is static ($\tau_k = 1$). However, the red sphere is visible from the k n frames later. The user-defined value $\delta_{t_{max}}$ prevents the record from contributing if $n > \delta_{t_{max}}$, reducing the risk of using obsolete records.

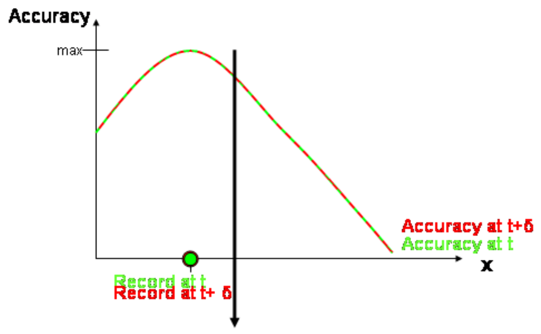
However, the flickering problem described in Section 8.1.1 still appears when a record is suddenly discarded. As proposed in [TMS02], this issue is limited by keeping track of the location of the records over time. Let us consider a record k located at point $\mathbf{p_k}$. If k was allowed to contribute to the previous frame and cannot be reused in current frame, a new record l is created at the same location, i.e. $\mathbf{p_l} = \mathbf{p_k}$ (Figure 8.4(b)). Since the location of visible records remains constant in space, the distribution of accuracy has less temporal variations, which reduces flickering artifacts. Note that the location of records remain constant even though they lie on dynamic objects (Figure 8.5).

The temporal weighting function provides a simple and adaptive way of leveraging temporal coherence by introducing an aging method based on the change of incoming radiance. However, as shown in Equation 8.3, the determination of our temporal weighting function relies on the knowledge of the incoming radiance at the next time step, E_{t_0+1}. Since the explicit computation of E_{t_0+1} would introduce a huge computational overhead, this value is estimated using a simple reprojection.

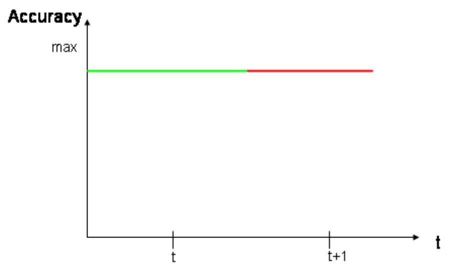
8.1.4 Estimating E_{t_0+1}

This method follows the reprojection approach proposed in [WDP99, WDG02]. However, while Walter *et al.* use reprojection for interactive visualization using ray tracing, the aim here is to provide a simple and reliable estimate of the incident radiance at a given point at time $t_0 + 1$ by using the data acquired at time t_0 only.

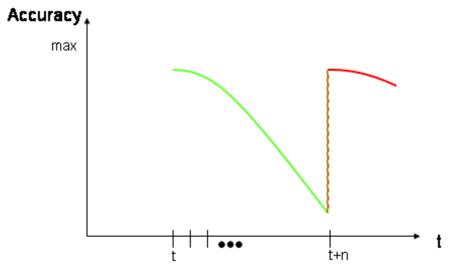
In the context of predefined animation, the changes in the scene are known and accessible at any time. When a record k is created at time t_0, the hemisphere above $\mathbf{p_k}$ is sampled (Figure 8.6(a)) to compute the incoming radiance and gradients at this point. Since the changes between times t_0 and $t_0 + 1$ are known, it is possible to reproject the points visible at time t_0 to obtain an estimate of the visible points at time $t_0 + 1$ (Figure 8.6(b)). The outgoing radiance



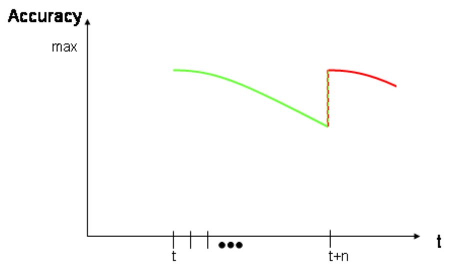
(a)



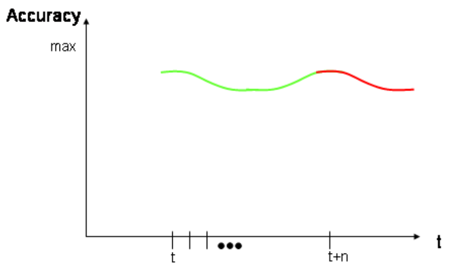
(b) New record after $\delta = 1$ frame



(c) New record after $\delta = n$ frames, no temporal gradient



(d) New record after $\delta = n$ frames, extrapolated temporal gradient



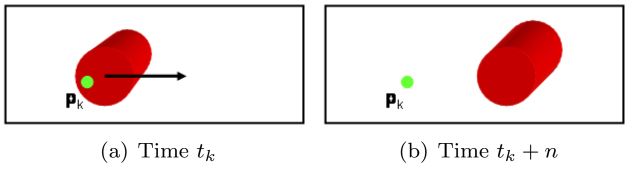
(e) New record after $\delta = n$ frames, interpolated temporal gradient

Figure 8.4: Empirical shape of the accuracy at a fixed time with respect to space (a), and at a fixed location with respect to time (b,c,d,e). If records are located at the same point between successive frames (a), the temporal accuracy is improved (b). However, flickering artifacts due to the temporal discontinuities of accuracy may appear when a record is reused in several frames, then recomputed (c). Extrapolated temporal gradients decrease the amplitude of the discontinuity in one pass, reducing flickering (d). Interpolated temporal gradients use two passes to eliminate the discontinuity by smoothing out the temporal changes (e).

of reprojected visible points can be estimated by accounting for the rotation and displacement of both objects and light sources. In overlapping areas, a depth test accounts for the occlusion change (Figure 8.6(c)).

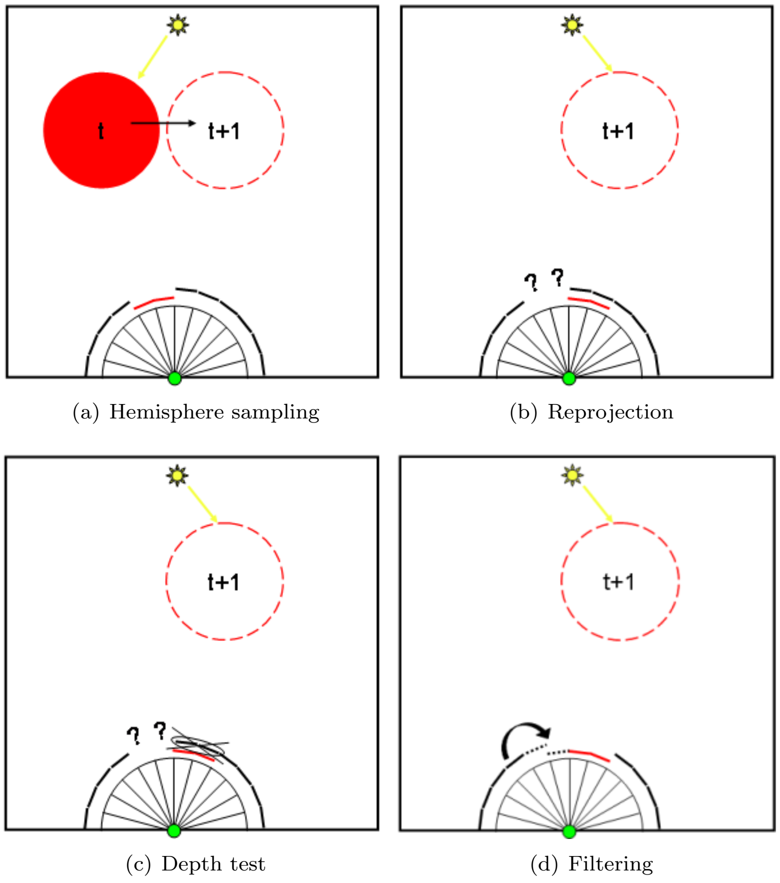
However, some parts of the estimated incoming radiance may be unknown (holes) due to displacement and overlapping of visible objects (Figure 8.6(d)). As proposed in [WDP99], we use a simple hole-filling method: each hole is filled using the background values, yielding a plausible estimate of the future indirect lighting.

Note that we use reprojection only to obtain an approximate of the incoming radiance at a



(a) Time t_k (b) Time $t_k + n$

Figure 8.5: Record k created at time t_k remains at point $\mathbf{p_k}$ even though it lies on a dynamic object.



(a) Hemisphere sampling (b) Reprojection

(c) Depth test (d) Filtering

Figure 8.6: The hemisphere is sampled at time t as in the classical irradiance caching process (a). For each ray, our method estimates where each visible point will be located at time $t+1$ by reprojection (b). Distant overlapping points are removed using depth test (c), while resulting holes are filled using the farthest neighboring values (d).

given point at time $t_0 + 1$. As shown in Figure 8.7 and Table 8.1, the reprojection reduces the error in the estimate of the future incoming lighting. Those errors were measured by comparing the irradiances at time $t + 1$ with the irradiances estimated using records created at time t.

The temporal weighting function and record replacement scheme, along with an estimation of future incoming radiance, allow to improve both the computational efficiency and the animation quality. Nevertheless, Figure 8.4(c) shows that the accuracy of the computation is still not continuous in the course of time. Replacing obsolete records by new ones creates a discontinuity

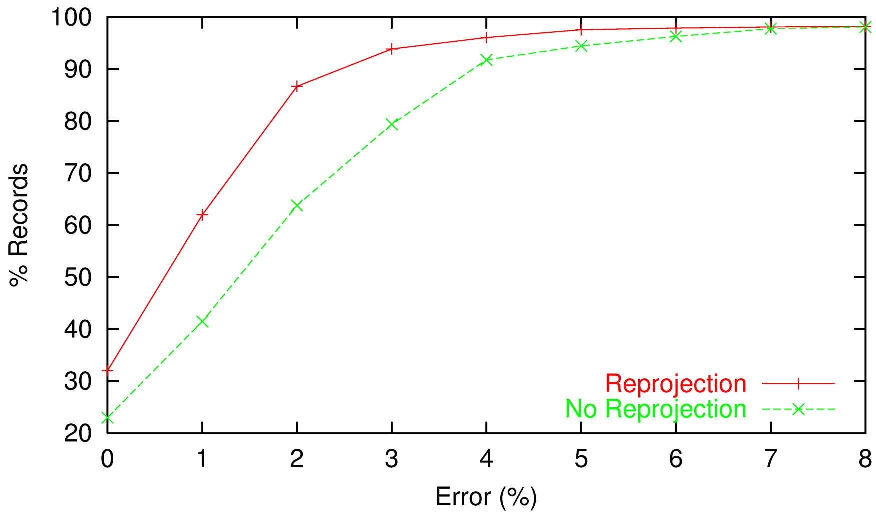


Figure 8.7: Percentage of records for which the estimate of the future incoming lighting is below a given RMS error level. The reprojection reduces the overall error in the estimate compared to a method without reprojection (i.e. the where the lighting is considered temporally constant). (Errors computed using 4523 values.)

Error	Reprojection	No Reprojection
Min	0%	0%
Max	30%	32%
Mean	2.9%	3.7%
Median	1.6%	2.4%

Table 8.1: RMS error of reprojection (Based on 4523 values).

of accuracy, which causes the visible flickering artifacts. The degree of the temporal irradiance interpolation is then raised using *temporal gradients*.

8.1.5 Temporal Gradients

Temporal gradients are conceptually equivalent to classical irradiance gradients. Instead of representing the incoming radiance change with respect to translation and rotation, those gradients represent how the incoming radiance gets altered over time.

In the context of irradiance caching, the irradiance at a point $\mathbf{p}$ is estimated using rotation and translation gradients. The temporal irradiance gradient of record k at a given point $\mathbf{p}$ with normal $\mathbf{n}$ is derived from the formulae proposed in [WH92]:

$$\nabla^{\mathbf{t}}(\mathbf{p}) = \frac{\partial}{\partial t}(E_k + (\mathbf{n_k} \times \mathbf{n}) \cdot \nabla_{\mathbf{r}} + (\mathbf{p} - \mathbf{p_k}) \cdot \nabla_{\mathbf{p}}) \tag{8.9}$$

where:

- $\nabla_{\mathbf{r}}$ and $\nabla_{\mathbf{p}}$ are the rotation and translation gradients
- $\mathbf{p_k}$ and $\mathbf{n_k}$ are the location and normal of record k

As described in section 8.1.3, the location of all records is kept constant over time. Let us choose any point of interest $\mathbf{p}$ with normal $\mathbf{n}$ which is also constant over time. Therefore, $\mathbf{n_k} \times \mathbf{n}$ and $\mathbf{p} - \mathbf{p_k}$ are constant with respect to time. The equation for temporal gradients becomes:

$$\nabla^{\mathbf{t}}(\mathbf{p}) = \nabla^{\mathbf{t}}_{\mathbf{E_k}} + (\mathbf{n_k} \times \mathbf{n}) \cdot \nabla^{\mathbf{t}}_{\nabla_{\mathbf{r}}} + (\mathbf{p} - \mathbf{p_k}) \cdot \nabla^{\mathbf{t}}_{\nabla_{\mathbf{p}}} \tag{8.10}$$

where:

- $\nabla^{\mathbf{t}}_{\mathbf{E_k}} = \frac{\partial E_k}{\partial t}$ is the *temporal gradient of irradiance*
- $\nabla^{\mathbf{t}}_{\nabla_{\mathbf{r}}} = \frac{\partial \nabla_{\mathbf{r}}}{\partial t}$ is the *temporal gradient of rotation gradient*
- $\nabla^{\mathbf{t}}_{\nabla_{\mathbf{p}}} = \frac{\partial \nabla_{\mathbf{p}}}{\partial t}$ is the *temporal gradient of translation gradient*

Using Equation 8.10, the contribution of record k created at time t_k to the incoming radiance at point $\mathbf{p}$ at time t is estimated by:

$$\begin{aligned} E_k(\mathbf{p}, t) &= E_k + \nabla^{\mathbf{t}}_{\mathbf{E_k}}(t - t_k) + \\ &\quad (\mathbf{n_k} \times \mathbf{n}) \cdot (\nabla_{\mathbf{r}} + \nabla^{\mathbf{t}}_{\nabla_{\mathbf{r}}}(t - t_k)) + \\ &\quad (\mathbf{p_k} - \mathbf{p}) \cdot (\nabla_{\mathbf{p}} + \nabla^{\mathbf{t}}_{\nabla_{\mathbf{p}}}(t - t_k)) \end{aligned} \tag{8.11}$$

This formulation represents the temporal change of the incoming radiance around p_k as 3 vectors. These vectors represent the change of the incoming radiance at point $\mathbf{p_k}$ and the change of the translation and rotation gradients over time. The values of these vectors can be easily computed using the information generated in Section 8.1.4. Since our method estimates the incoming radiance at time $t + 1$ using the information available at time t, we define *extrapolated* temporal gradients as:

$$\nabla^{\mathbf{t}}_{\mathbf{E_k}} \approx E(t_k + 1) - E(t_k) \tag{8.12}$$

$$\nabla^{\mathbf{t}}_{\nabla_{\mathbf{r}}} \approx \nabla_{\mathbf{r}}(t_k + 1) - \nabla_{\mathbf{r}}(t_k) \tag{8.13}$$

$$\nabla^{\mathbf{t}}_{\nabla_{\mathbf{p}}} \approx \nabla_{\mathbf{p}}(t_k + 1) - \nabla_{\mathbf{p}}(t_k) \tag{8.14}$$

However, as illustrated in Figure 8.4(d), these temporal gradients do not remove all the discontinuities in the animation. When a record k is replaced by record l, the accuracy of the result exhibits a possible discontinuity, yielding some flickering artifacts. As explained in section 8.1.3, this problem can be avoided by keeping track of the history of the records: when record k gets obsolete, a new record l is created at the same location. Since $t_l > t_k$, we can use the value of incoming radiance stored in l to compute *interpolated* temporal gradient for record k:

$$\nabla^{\mathbf{t}}_{\mathbf{E_k}} \approx (E_l - E_k)/(t_l - t_k) \tag{8.15}$$

$$\nabla^{\mathbf{t}}_{\nabla_{\mathbf{r}}} \approx (\nabla_{\mathbf{r}}(t_l) - \nabla_{\mathbf{r}}(t_k))/(t_l - t_k) \tag{8.16}$$

$$\nabla^{\mathbf{t}}_{\nabla_{\mathbf{p}}} \approx (\nabla_{\mathbf{p}}(t_l) - \nabla_{\mathbf{p}}(t_k))/(t_l - t_k) \tag{8.17}$$

As illustrated in Figure 8.4(e), the temporal gradients enhance the continuity of the accuracy, hence removing the flickering artifacts. However, the gradients only account for the first derivative of the change of incoming lighting, hence temporally smoothing the changes. While this method proves accurate in scenes with smooth changes, it should be noted that the gradient-based temporal interpolation may introduce ghosting artifacts when used in scenes with very sharp changes of illumination. In this case, a^t and $\delta_{t_{max}}$ must be reduced to obtain a sufficient update frequency.

This method provides a simple way of extending irradiance caching to dynamic objects and light sources, by introducing a temporal weighting function and temporal gradients. In the next section, we discuss the implementation of our method on a GPU for increased performance.

8.2 GPU Implementation Details

This section first details the implementation of the incoming radiance estimate by reprojection (Section 8.1.4). Then, we describe how the GPU can be simply used in the context of radiance cache splatting to discard useless records and avoid their replacement.

Reprojection of Incoming Radiance As shown in section 8.1.3, the computation of the temporal weighting function and temporal gradients for a given record k requires an estimate of the radiance reaching point $\mathbf{p_k}$ at the next time step. This estimate is obtained through reprojection (section 8.1.4), provided that the position of the objects at next time step is known. Therefore, for a given vertex v of the scene and a given time t, we assume that the transformation matrix corresponding to the position and normal of v at time $t+1$ is known. We assume that such matrices are available for light sources as well. Using the hemisphere sampling method described in the previous chapter, a record k can be generated by rasterizing the scene on a single plane above point $\mathbf{p_k}$. In a first pass, during the rasterization at time t, shaders can output an estimate of the position and incoming lighting at time $t+1$ of each point visible to $\mathbf{p_k}$ at time t. This output can be used to reconstruct an estimate of the incoming radiance function at time $t+1$.

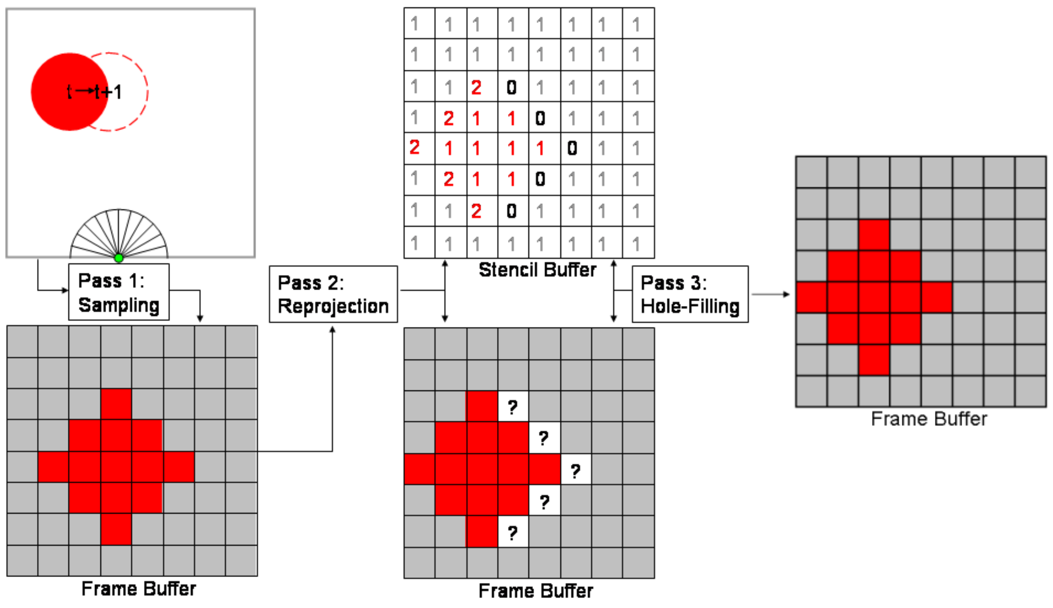


Figure 8.8: Reprojection using the GPU. The first pass samples the scene to gather the required information. Then, the visible points are reprojected to their estimated position at next time step. During this pass, each rendered fragment increments the stencil buffer. Finally, the holes (i.e. where the stencil value is 0) are filled using the deepest neighboring values.

This estimate is obtained in a second pass: each projected visible point generated in the first pass is considered as a vertex. This operation can be easily implemented using either the OpenGL Pixel Buffer Objects or OpenGL PBuffer rendering and per-vertex texture mapping. Each of those vertices is sent to the graphics pipeline as a pixel-sized point. The result of the rasterization process is an estimate of the incoming radiance function at time $t+1$. Since the size of the sampling plane is usually small (typically 64×64), this process is generally much faster than resampling the whole scene.

During the reprojection process, some fragments may overlap. Even though the occlusion can be simply solved by classical Z-Buffer, the resulting image may contain holes (Figure 8.6(d)). These holes are created at the location of dynamic objects. Since the time shift between two successive frames is very small, the holes are also small. As described in Section 8.1.4, a third pass fills the holes using the local background (that is, the neighboring value with the highest depth). This computation can be performed efficiently on the GPU using the stencil buffer with an initial value of 0. During the reprojection process, each rasterized point increments the stencil buffer. Therefore, the hole-filling algorithm must be applied only on pixels where the stencil buffer is still 0. The final result of this algorithm is an estimate of the incoming radiance at time $t + 1$, generated entirely on the GPU (Figure 8.8). This estimate is used in the extrapolated temporal gradients and the temporal weighting function. As shown in Equation 8.6, this latter defines a maximum value of the lifespan of a given record, and triggers its recomputation. However, this recomputation is not always necessary.

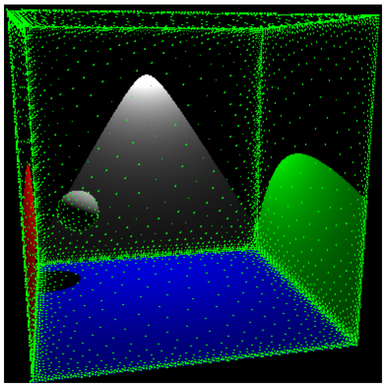
Culled Replacement/Deletion As described in the previous sections, the flickering artifacts of the lighting come from the temporal discontinuities of the accuracy. Therefore, if a record cannot contribute to the current image (i.e. out of the view frustum, or occluded), it can be simply deleted instead of being replaced by a novel, up-to-date record. This avoids the generation and update of a “trail” of records following dynamic objects (Figure 8.9), hence reducing the memory and computational costs. In the context of radiance cache splatting, this decision can be easily made using hardware occlusion queries: during the last frame of the lifespan of record k, an occlusion query is issued as the record is rasterized. In the next frame, valid records are first rendered. If a record k is now obsolete, the result of the occlusion query is read from the GPU. If the number of covered pixels is 0, the record is discarded. Otherwise, a new record l is computed at location $\mathbf{p_l} = \mathbf{p_k}$.

The hardware occlusion queries are very useful, but they suffer from high latency. However, in our method, the result of a query is not needed immediately. Between the query issue and the reading of the record coverage, the renderer renders the other records, then switches the scene to next frame and renders valid records. In practice, the average latency appeared to be negligible (less than 0.1% of the overall computing time). Besides, in the following test scenes, this method reduces the storage and computational costs by up to 25-30%.

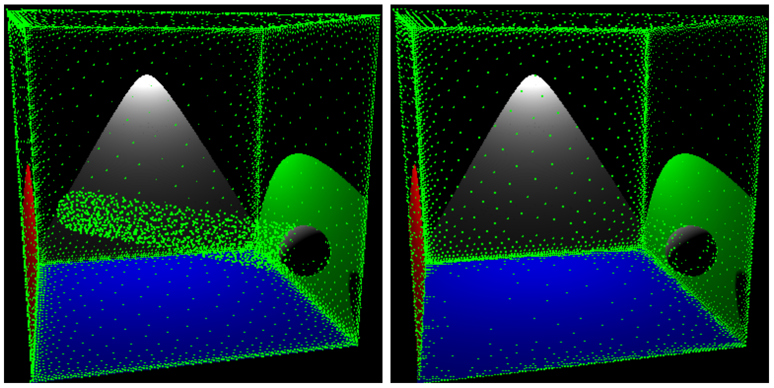
8.3 Results

This section discusses the results obtained using temporal radiance caching compared to the classical approach in which a new cache is computed for each frame. This method is referred to as *per-frame computation* in the remainder of this section. The images, videos and timings have been generated using a 3.8GHz Pentium 4 with 2 GB RAM and an nVidia GeForce 7800 GTX 512MB. The scene details and timings are summarized in Table 8.2.

Cube in a Box This very simple, diffuse scene (Figure 8.12(a)) exhibits high flickering when no temporal gradients are used. Along with a significant speedup, our method reduces the flickering artifacts by using extrapolated temporal gradients. Such artifacts are unnoticeable with interpolated gradients. The animations are generated using a temporal accuracy parameter $a^t = 0.05$, and a maximum lifespan of 20 frames. The per-frame computation requires about 138 MB (772K records) to store all the irradiance records. In our method, the memory load is only 12.4 MB (50K records). Figure 8.10 shows the accuracy values obtained with and without temporal gradients. The remaining flickering of the extrapolated temporal gradients are due to the discontinuities of accuracy. Since our aim is high quality rendering, the following results focus on interpolated temporal gradients which avoid discontinuities.



(a) Time 1



(b) Time 100, systematic update (c) Time 100, our record removal method

Figure 8.9: The sphere moves from the left to the right of the Cornell Box. At time 1 (a), records (represented by green points) are generated to compute the global illumination solution. When the sphere moves, new records are created to evaluate the incoming radiance on the sphere. If every record is permanently kept up-to-date, a "trail" of records lies on the path of the dynamic sphere (b). Using culling, only useful records are updated (c).

Moving Light A similar scene (Figure 8.12(b)) illustrates the behavior of our algorithm in the context of dynamic light sources. The bottom of the box is tiled to highlight the changes of indirect lighting. Due to the highly dynamic indirect lighting, the lifespan of the records is generally very short, yielding frequent updates of irradiance values. Compared to per-frame computation, our method renders the animation with higher quality in a comparable time.

Flying Kite In a more complicated, textured scene (Figure 8.12(c)), our algorithm also provides a drastic quality improvement while significantly reducing the computation time. In the beginning of the animation the change of indirect lighting is small, and hence the records can be reused in several frames. However, when the kite gets down, its dynamic reflection on the ceiling and wall is clearly noticeable. Using our temporal weighting function, the global illumination solution of this zone is updated at a fast pace, avoiding ghosts in the final image (Figure 8.11).

Japanese Interior In this complex scene (Figure 8.12(d)), the glossy and diffuse objects are rendered using respectively the radiance and irradiance caching algorithms. The animation illustrates the features of our method: dynamic nondiffuse environment, and important changes of indirect lighting. In the beginning of the animation, the scene is lit by a single, dynamic light source. In this case, temporal gradients suppress the flickering artifacts present in per-frame computation, but do not provide a significant speedup (1.25×). In the remainder of the animation, most of the environment is static, even though some dynamic objects generate strong changes in the indirect illumination. Our temporal gradients take advantage of this situation by adaptively reusing records in several frames. The result is the elimination of flickering and a significant speedup (up to 9×) compared to per-frame computation. During the generation of this animation, the average latency introduced by occlusion queries is 0.001% of the overall rendering time.

Spheres This scene features complex animation with 66 diffuse and glossy bouncing spheres and a glossy back wall (Figure 8.12(e)). Temporal radiance caching eliminates the flickering while reducing the computational cost of a factor 4.24. The temporal accuracy value is $a^t = 0.05$ and the maximum record lifespan $\delta_{t_{max}} = 5$.

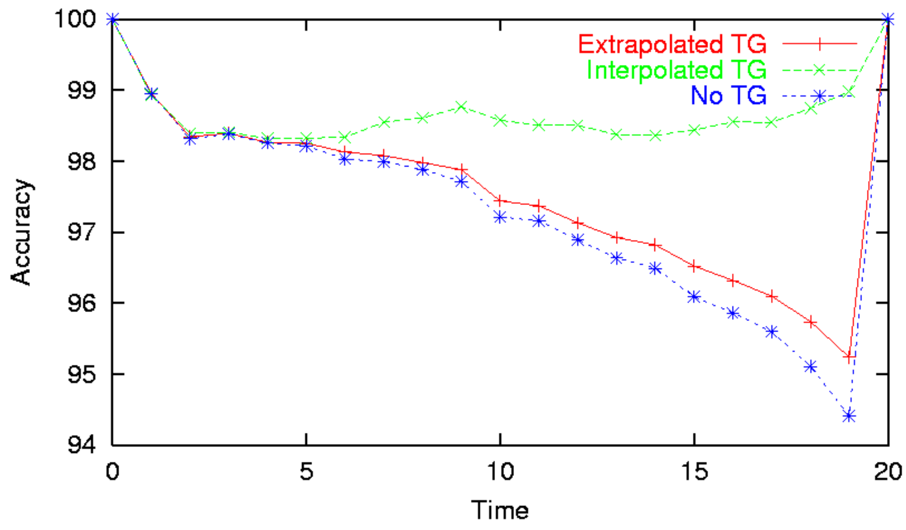


Figure 8.10: Temporal accuracy values obtained in scene Cube in a Box by creating records at time 0 and extrapolating their value until time 19. New records are recomputed at time 20. The temporal gradients (TG) provide a better approximation compared to the approach without those gradients. Using interpolated gradients, the accuracy is continuous and remains above 98%.

Scene	Nb. Poly	Nb. Frames	Per-Frame Comp.	Our Method	Speedup
Cube in a Box	24	400	2048s	269s	7.62
Moving Light	24	400	2518s	2650s	0.95
Flying Kite	28K	300	5109s	783s	6.52
Japanese Interior	200K	750	13737s	7152s	1.9
Spheres	64K	200	3189s	753s	4.24

Table 8.2: Test scenes and timings

Computational Overhead of Reprojection During the computation of a record, this method evaluates the value of the incoming lighting for both current and next time steps. As

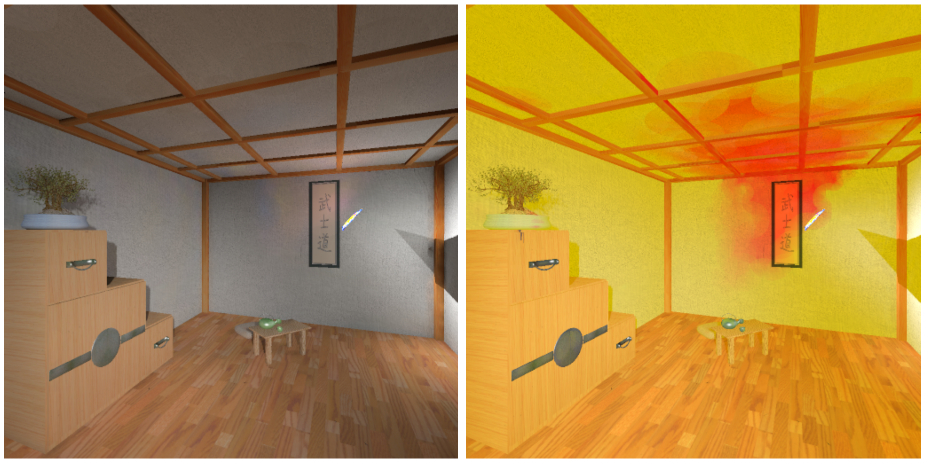

(a) Actual sampling frame (b) Records lifespan

Figure 8.11: When computing the global illumination solution for the current frame (a), the temporal caching scheme estimates where the lighting changes. The lifespan of each generated record is computed by estimating the future change of lighting (b). Green and red colors respectively represent long and short lifespan.

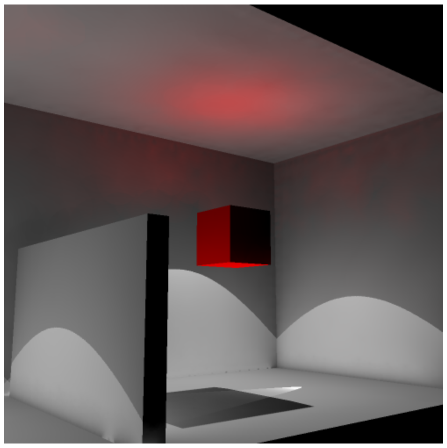
shown in Section 8.1.4, the estimation of the future incoming lighting is performed by simple reprojection. Therefore, the related computational overhead is independent of the scene geometry. In our tests, each record was computed at resolution 64×64. On our system, the reprojection is performed in approximately 0.46 ms. For comparison, the time required to compute the actual incoming lighting at a given point in our 200K polygons scene is 4.58 ms. In this case, the overhead due to the reprojection is only 10% of the cost of the actual hemisphere sampling. Even though this overhead is not negligible, this estimate reduces the overall rendering time by reusing the records in several frames.

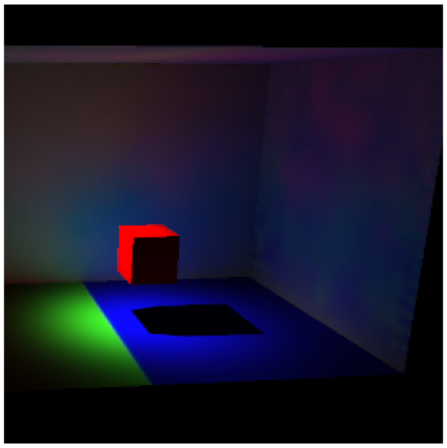
8.4 Conclusion

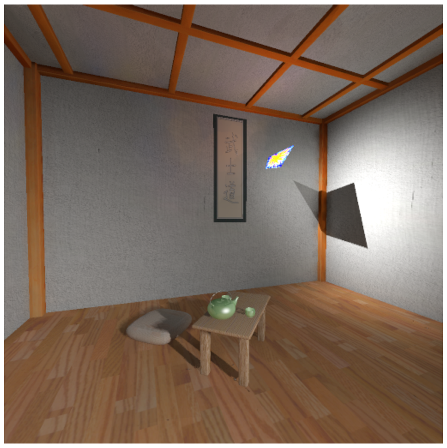
This chapter presented a novel method for exploiting temporal coherence in the context of irradiance caching. When used to render each frame of an animation, irradiance and radiance caching create the temporal discontinuities of the accuracy of the lighting, resulting in disturbing flickering artifacts. This approach is based on sparse sampling and interpolation of the indirect lighting in the temporal domain. The irradiance and radiance records are adaptively reused in several frames using our temporal weighting function and temporal gradients. The temporal weighting function determines the lifespan of each record depending on the change rate of the local incoming radiance: if strong changes of incoming radiance are detected, the corresponding lifespan is reduced to ensure a sufficient temporal sampling rate. At the end of the record lifespan, a new record is generated at the same location. This method reduces the flickering artifacts while allowing for the computation of interpolated temporal gradients for smooth temporal interpolation of the indirect lighting.

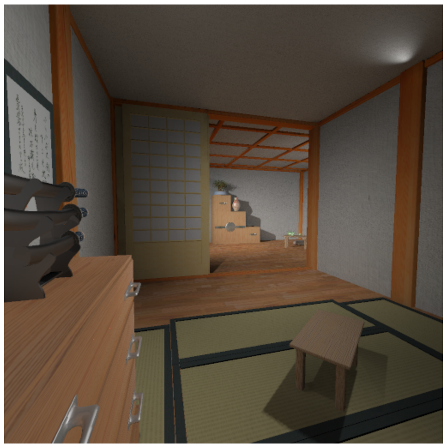
The evaluation of the temporal weighting function and of the extrapolated temporal gradients rely on an approximate knowledge of the future incoming radiance, which can be performed using GPU-accelerated reprojection. The GPU is also used to determine which records must be updated at the end of their respective lifespans.

The results show both a significant speedup and an increased quality compared to per-frame computation. Due to sparse temporal sampling, the incoming radiance values for the entire animation segment can be stored within main memory.


(a) Cube in a Box


(b) Moving Light


(c) Flying Kite


(d) Japanese Interior

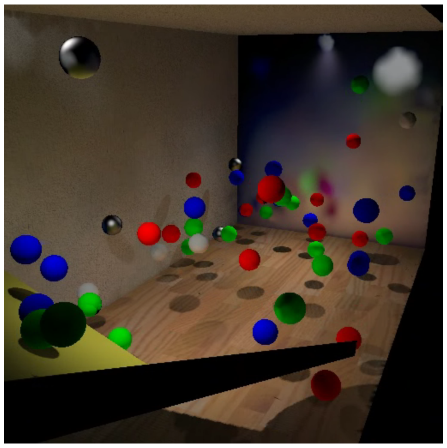

(e) Spheres

Figure 8.12: Images of scenes discussed in Section 8.3.

Bibliography

[GBP07] Pascal Gautron, Kadi Bouatouch, and Sumanta Pattanaik. Temporal radiance caching. *IEEE Transactions on Visualization and Computer Graphics*, 2007.

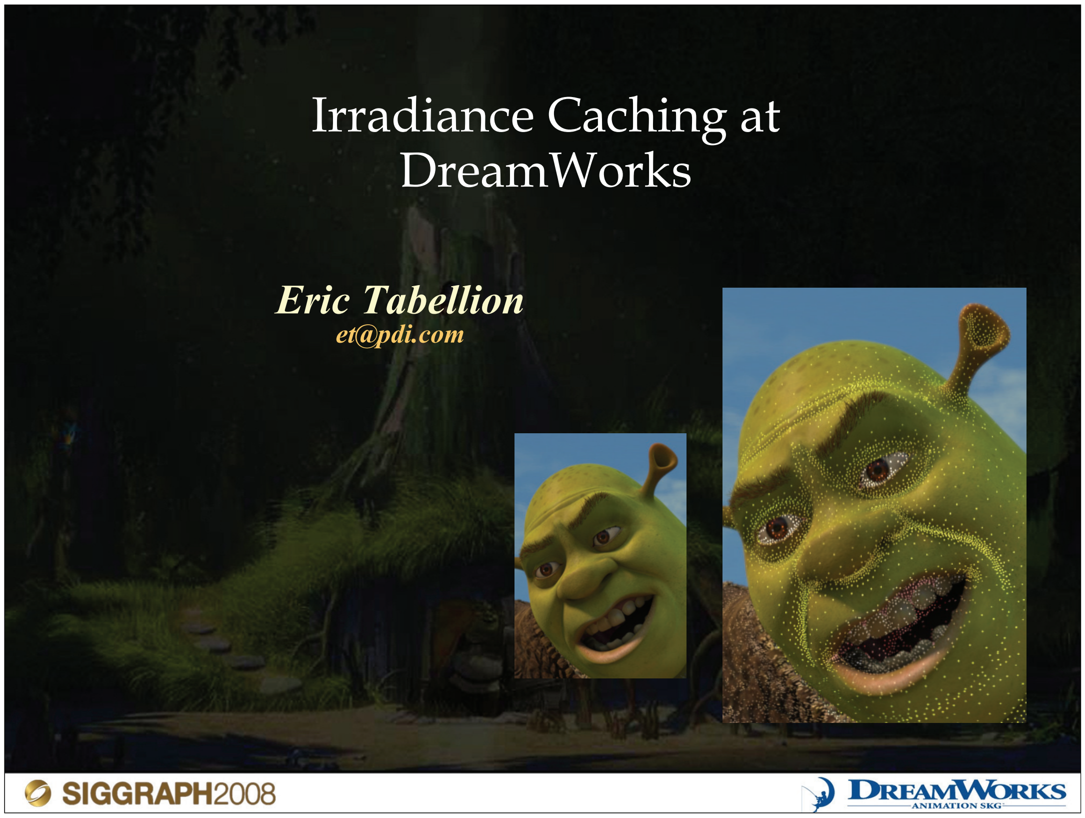
[TMS02] Takehiro Tawara, Karol Myszkowski, and Hans-Peter Seidel. Localizing the final gathering for dynamic scenes using the photon map. In *VMV*, 2002.

[WDG02] Bruce Walter, George Drettakis, and Donald P. Greenberg. Enhancing and optimizing the render cache. In *Proceedings of Eurographics Workshop on Rendering*, pages 37–42, 2002.

[WDP99] Bruce Walter, George Drettakis, and Steven Parker. Interactive rendering using the render cache. In *Proceedings of Eurographics Workshop on Rendering*, pages 235–246, 1999.

[WH92] Gregory J. Ward and Paul Heckbert. Irradiance Gradients. In *Proceedings of Eurographics Workshop on Rendering*, pages 85–98, Bristol, UK, May 1992.

[WRC88] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A Ray Tracing Solution for Diffuse Interreflection. In *Proceedings of SIGGRAPH*, volume 22, pages 85–92, August 1988.

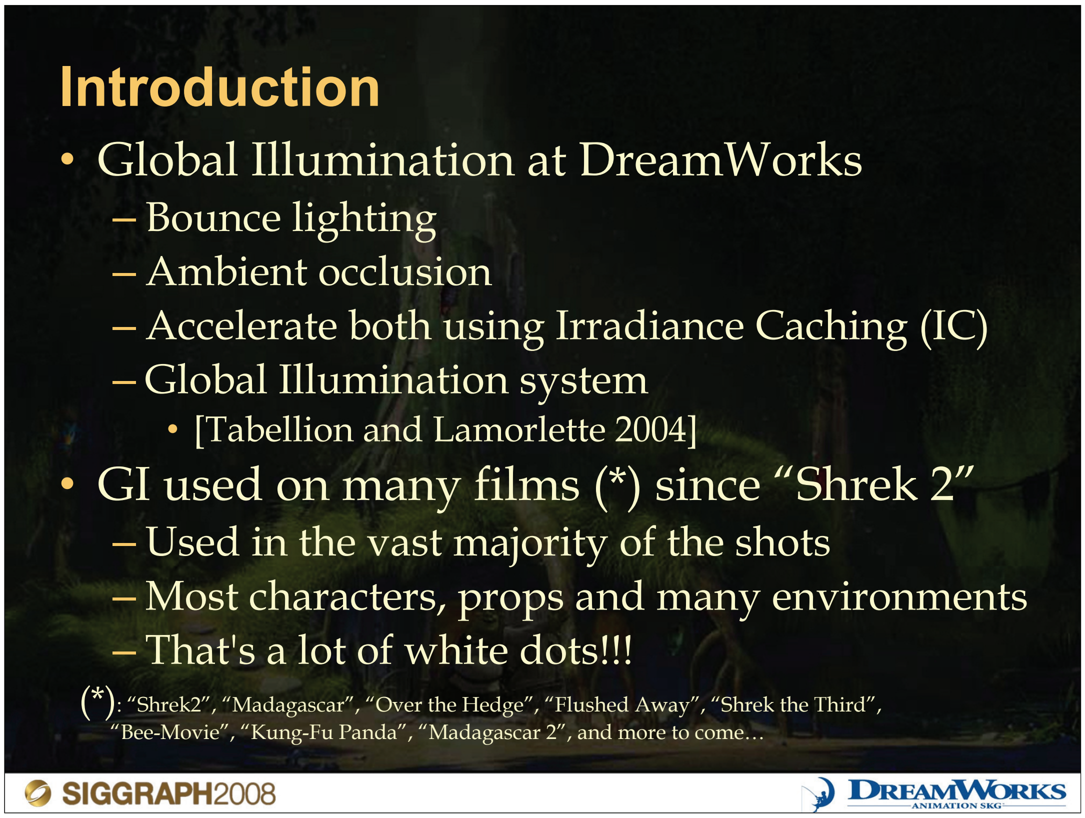


Irradiance Caching at DreamWorks

Eric Tabellion
et@pdi.com

SIGGRAPH2008

DREAMWORKS ANIMATION SKG



Introduction

- Global Illumination at DreamWorks
  - Bounce lighting
  - Ambient occlusion
  - Accelerate both using Irradiance Caching (IC)
  - Global Illumination system
    - [Tabellion and Lamorlette 2004]
- GI used on many films (*) since “Shrek 2”
  - Used in the vast majority of the shots
  - Most characters, props and many environments
  - That's a lot of white dots!!!

(*): “Shrek2”, “Madagascar”, “Over the Hedge”, “Flushed Away”, “Shrek the Third”, “Bee-Movie”, “Kung-Fu Panda”, “Madagascar 2”, and more to come…

SIGGRAPH2008 DREAMWORKS ANIMATION SKG

Global Illumination at DreamWorks refers to rendering one bounce of diffuse interreflection, as well as ambient occlusion.

We use a proprietary renderer to render all our films. Many implementation details of this global illumination system are published in our 2004 Siggraph paper [Tabellion and Lamorlette 2004]. In our toolset, both bounce lighting and ambient occlusion are computed using irradiance caching to reduce the amount of raytracing and shading required.

GI has been used extensively at DreamWorks to light and render many animated films since “Shrek 2”. If you consider the list of films above and the number of frames per film, we are glad to have been using irradiance caching to render all these images!

Examples: “Shrek 2”

SIGGRAPH2008

DreamWorks Animation SKG

Here are some example images from the movie “Shrek 2”...

Examples: “Madagascar”

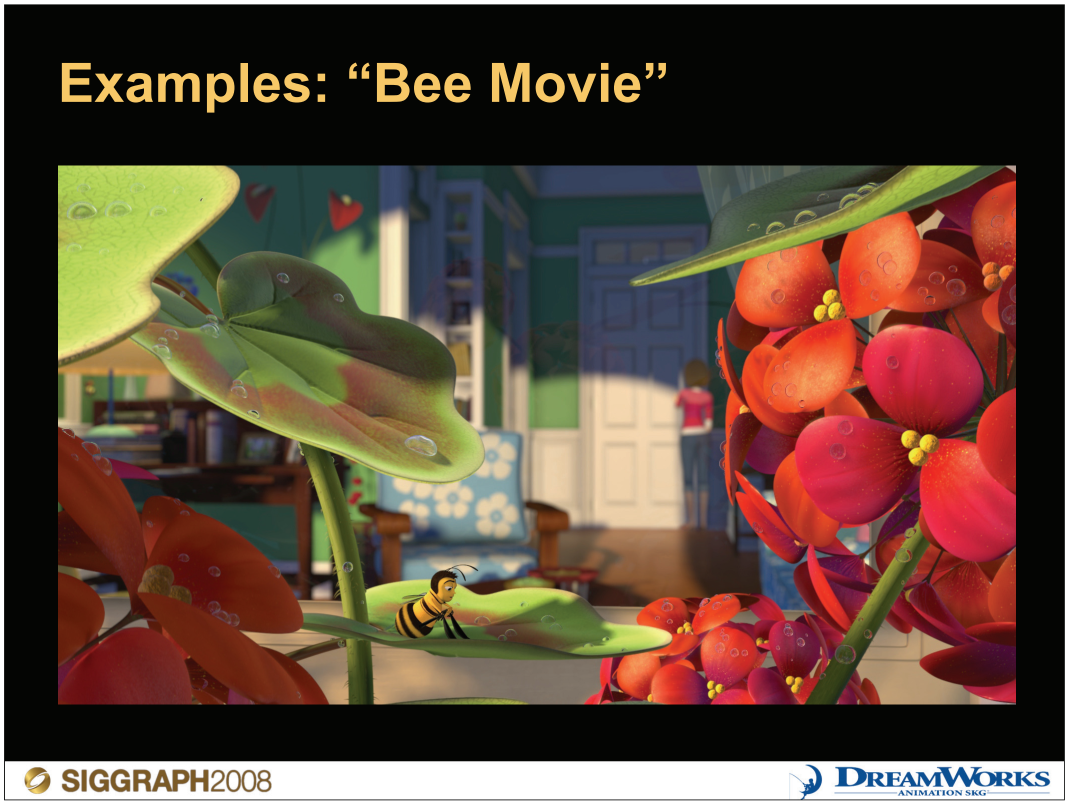
SIGGRAPH2008 DREAMWORKS ANIMATION SKG

... some more examples from movies released since then…

Examples: "Over the Hedge"

SIGGRAPH2008 DREAMWORKS ANIMATION SKG

... some more examples from movies released since then…



Examples: “Bee Movie”

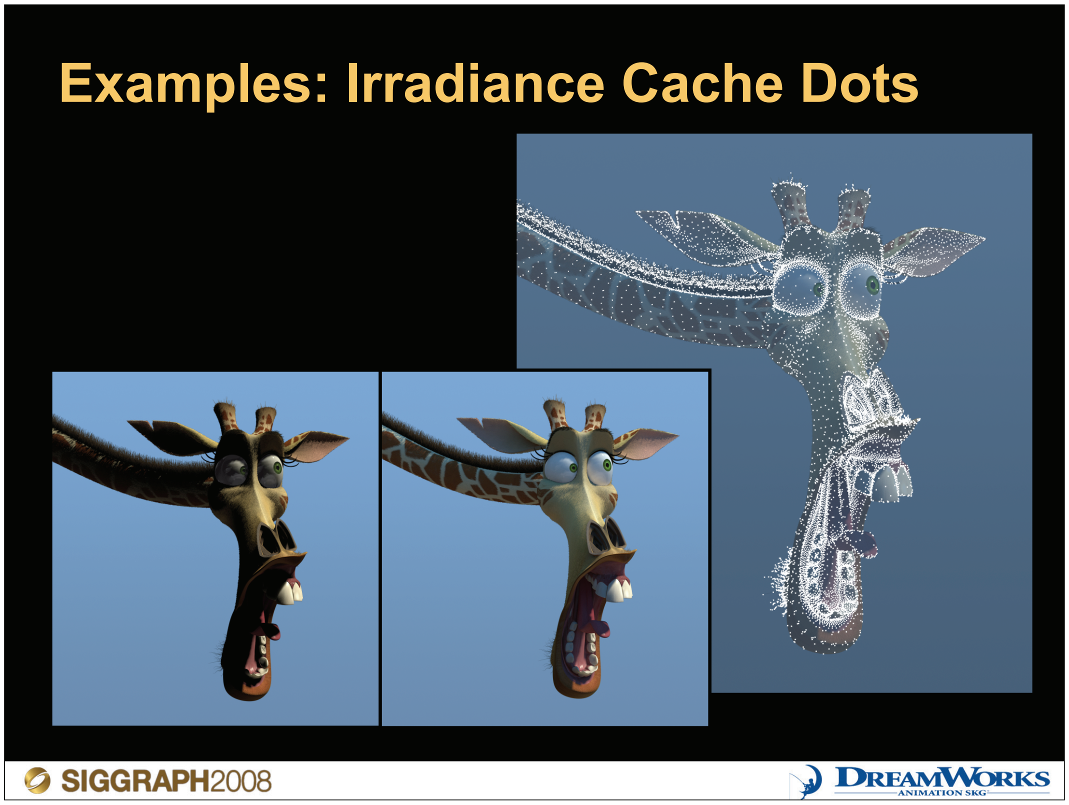
SIGGRAPH2008 DreamWorks Animation SKG

... some more examples from movies released since then…

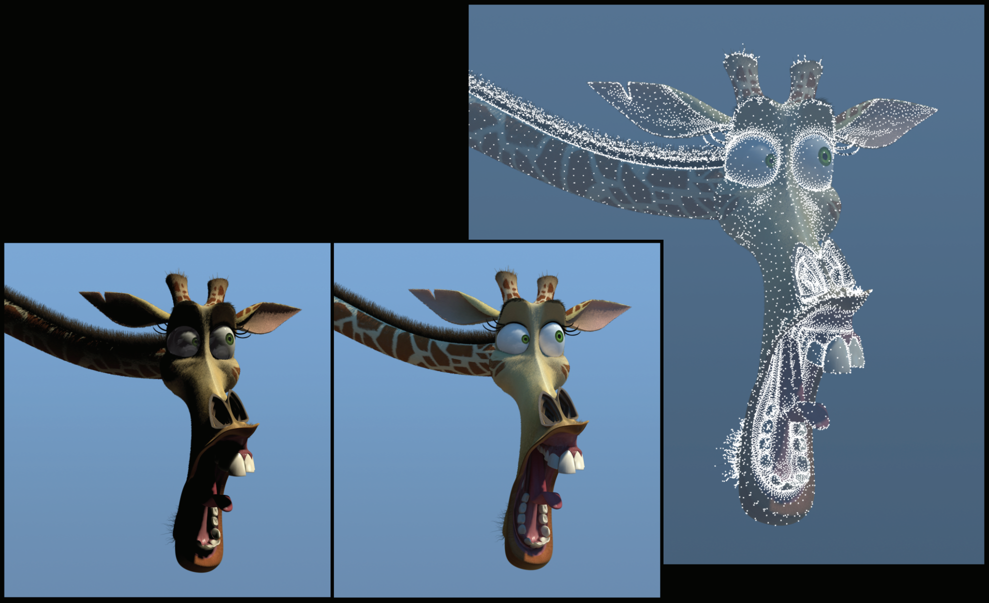
Examples: “Kung Fu Panda”

SIGGRAPH2008 DreamWorks Animation SKG

... and some more examples from the movie “Kung Fu Panda”. All these images use irradiance caching under the hood.

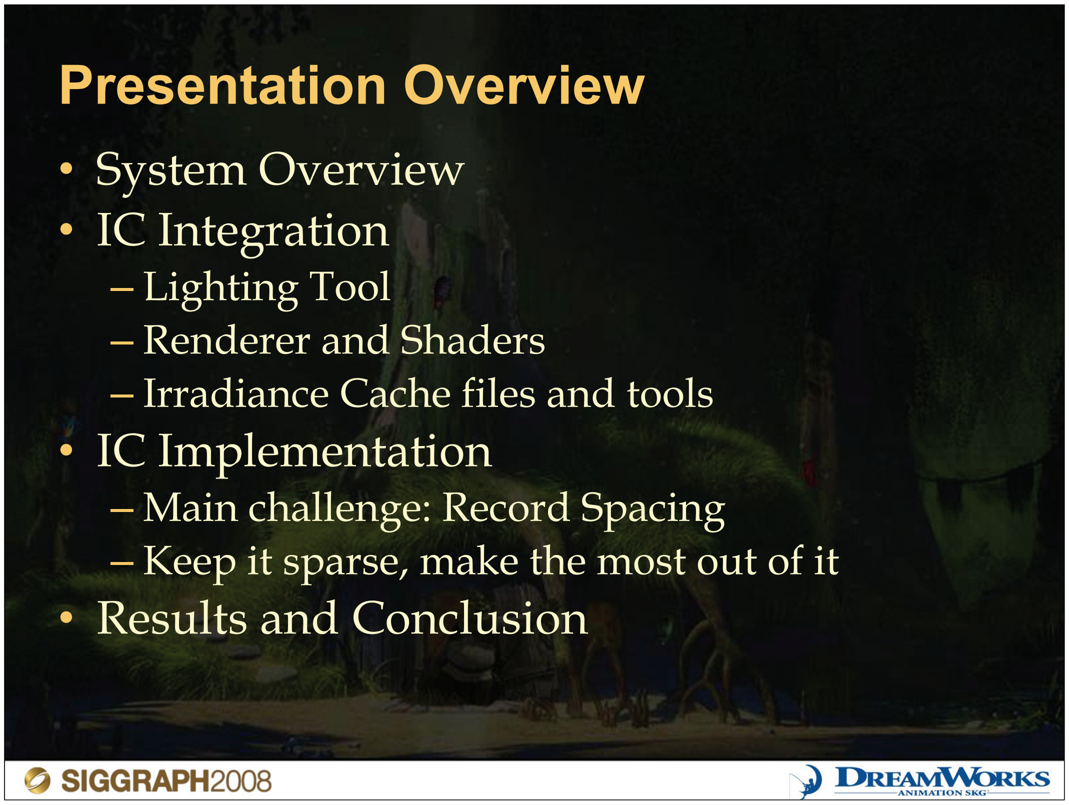


Examples: Irradiance Cache Dots



SIGGRAPH2008 DreamWorks Animation SKG

Here are image that show Melman with and without global illumination, as well as the corresponding irradiance cache.



Presentation Overview

- System Overview
- IC Integration
  - Lighting Tool
  - Renderer and Shaders
  - Irradiance Cache files and tools
- IC Implementation
  - Main challenge: Record Spacing
  - Keep it sparse, make the most out of it
- Results and Conclusion

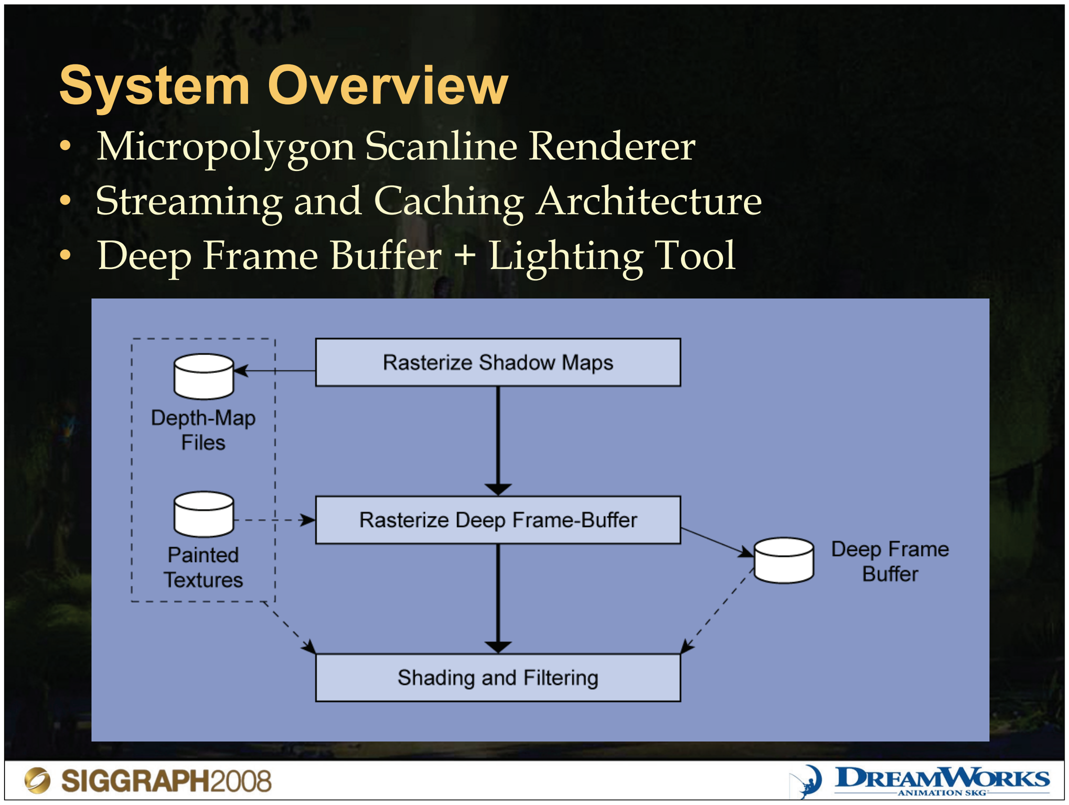
SIGGRAPH2008
DREAMWORKS ANIMATION SKG

In this section of the course, we will share some of our practical experience of implementing and using irradiance caching in a film production environment.

First we will review our global illumination system, and describe how irradiance caching fits within our rendering pipeline and how it integrates within a parallel production renderer. We will also talk and demo some of our tools to handle Irradiance Cache files.

We will then review some of the implementation details, covering the main challenge of Irradiance Caching: record spacing. We will show how specific solutions can help make IC up to an order of magnitude faster compared to final gathering at every pixel.

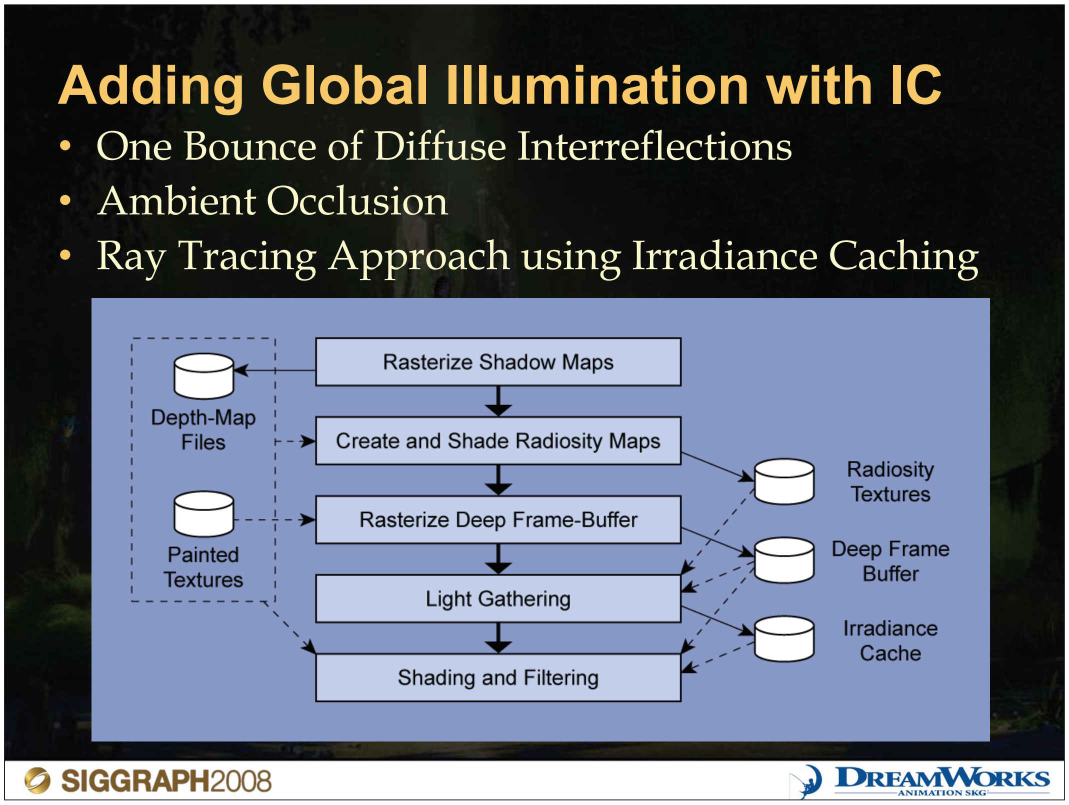
System Overview



System Overview

- Micropolygon Scanline Renderer
- Streaming and Caching Architecture
- Deep Frame Buffer + Lighting Tool

Our system is built around a proprietary micropolygon scanline renderer. It produces a deep frame buffer that can be used in an interactive lighting tool to shade an image multiple times while adjusting shader parameters. The deep frame buffer itself acts as a primary visibility rasterization cache and is also used when rendering images in batch for final quality renders.

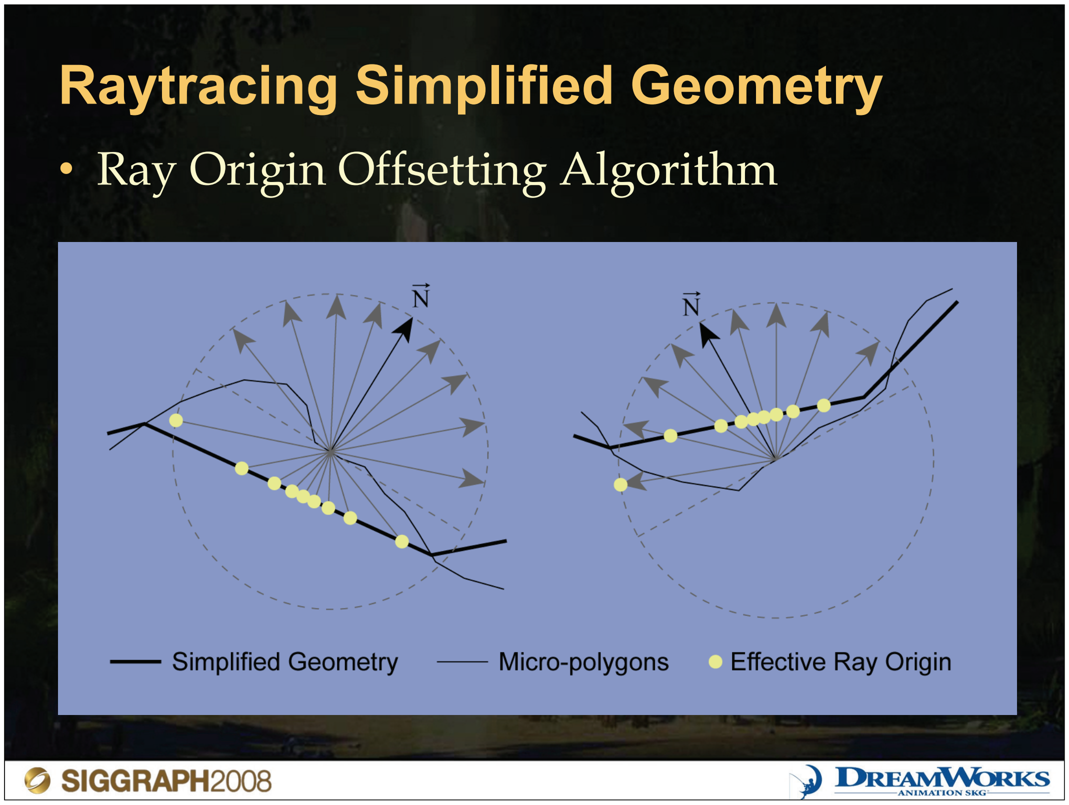


Adding Global Illumination with IC

- One Bounce of Diffuse Interreflections
- Ambient Occlusion
- Ray Tracing Approach using Irradiance Caching

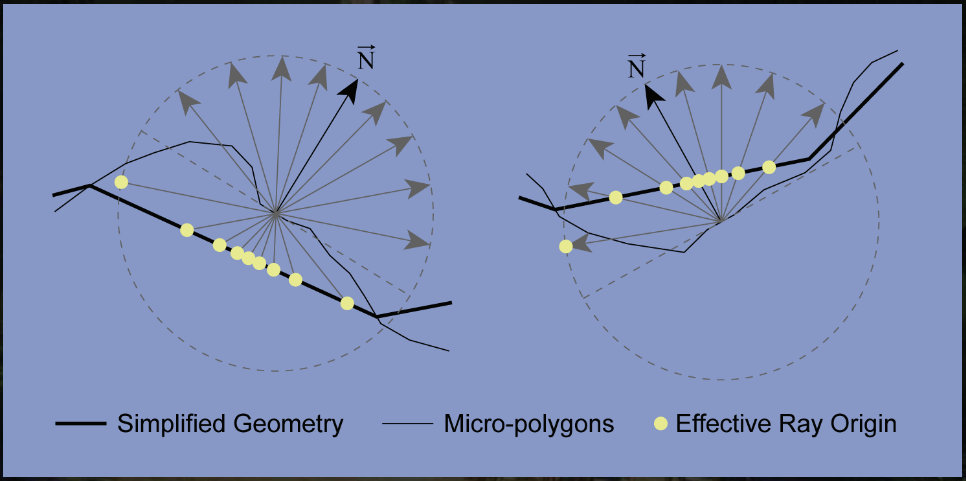
We enhanced our renderer with single-bounce global illumination capability by using a raytracing engine (also proprietary). We use it to perform final gathering calculations, as well as render reflections and refractions. To accelerate the expensive light gathering process, we implemented Irradiance Caching. The irradiance caching approach fits naturally into our rendering pipeline, which is using many other stages of caching. This strategy comes in handy when only a few steps need to be run again in the user's workflow to accomplish a task. Irradiance caching also helps by speeding up ambient occlusion calculations as we will see in later slides.

To further accelerate final gathering, the user has the ability to pre-compute a set of radiosity texture maps. These "baked" textures are called radiosity textures not because they have been computed using the radiosity algorithm, but rather because they contain a radiometric quantity known as radiosity. The textures essentially look like the result of direct lighting evaluations on purely diffuse textured surfaces. They are used to accelerate the shading of each gathering ray-intersection by replacing potentially complex shader network evaluations by a simple texture lookup. They can be thought of as the equivalent of a photon map, simplified to the context of a single bounce of diffuse interreflections.



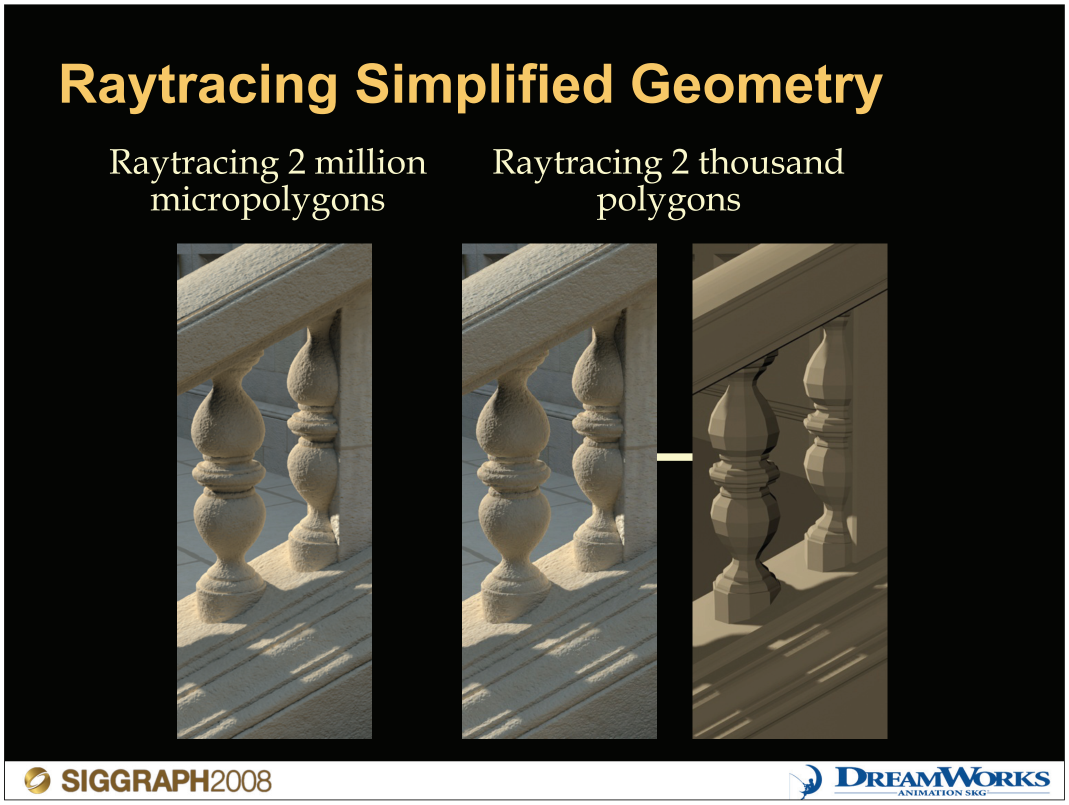
Raytracing Simplified Geometry

- Ray Origin Offsetting Algorithm



Here we illustrate how we integrate our raytracing engine within our micropolygon scanline renderer. When rays are cast, we do not actually attempt to intersect geometry finely tessellated down to pixel-size micropolygons - this would be indeed very expensive and use a lot of memory. Instead we tessellate geometry down to a simplified resolution and the raytracing engine only needs to deal with a much coarser set of polygons.

Since rays originate from positions on the micropolygon tessellation of the surface and can potentially intersect a coarser tessellation of the same surface, self intersection problems can happen. The image above illustrates cross-section examples of a surface tessellated both into micropolygons and into a coarse set of polygons. It also shows a few rays that originate from a micropolygon whith directions distributed over the hemisphere. To prevent self intersection problems to happen, we use a ray origin offsetting algorithm. In this algorithm, we adjust the ray origin to the closest intersection before / after the ray origin along the direction of the ray, within a user-specified distance. The ray intersection returned as a result of the ray intersection query is the first intersection that follows the adjusted ray origin. Please see our paper [TL04] for more details.



Raytracing Simplified Geometry

Raytracing 2 million micropolygons

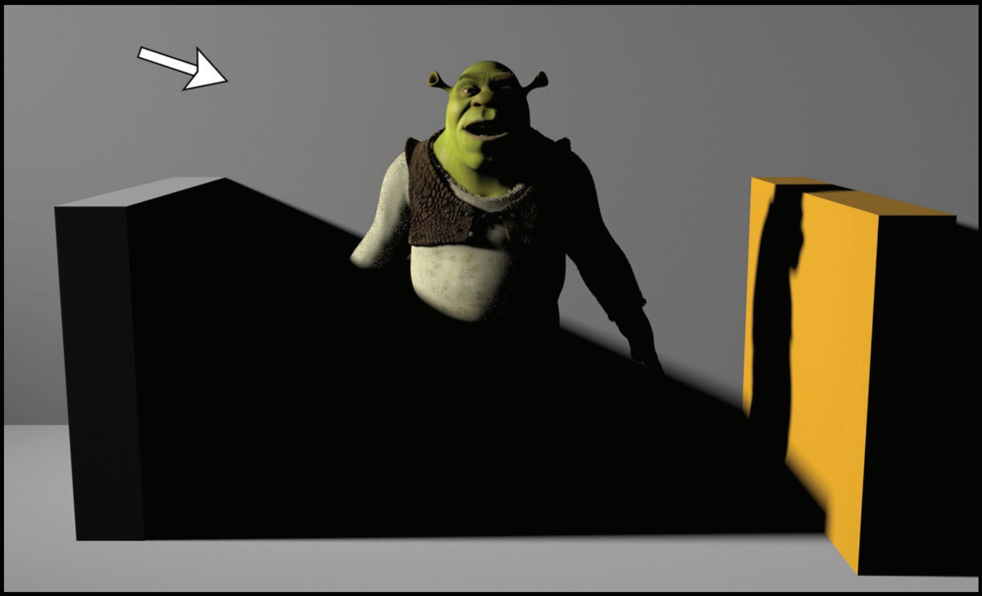
Raytracing 2 thousand polygons

SIGGRAPH2008

DreamWorks Animation SKG

Here is a comparison between our technique (center), and raytracing micropolygons (left). The image in the right shows the coarse tessellation used to perform ray intersections.

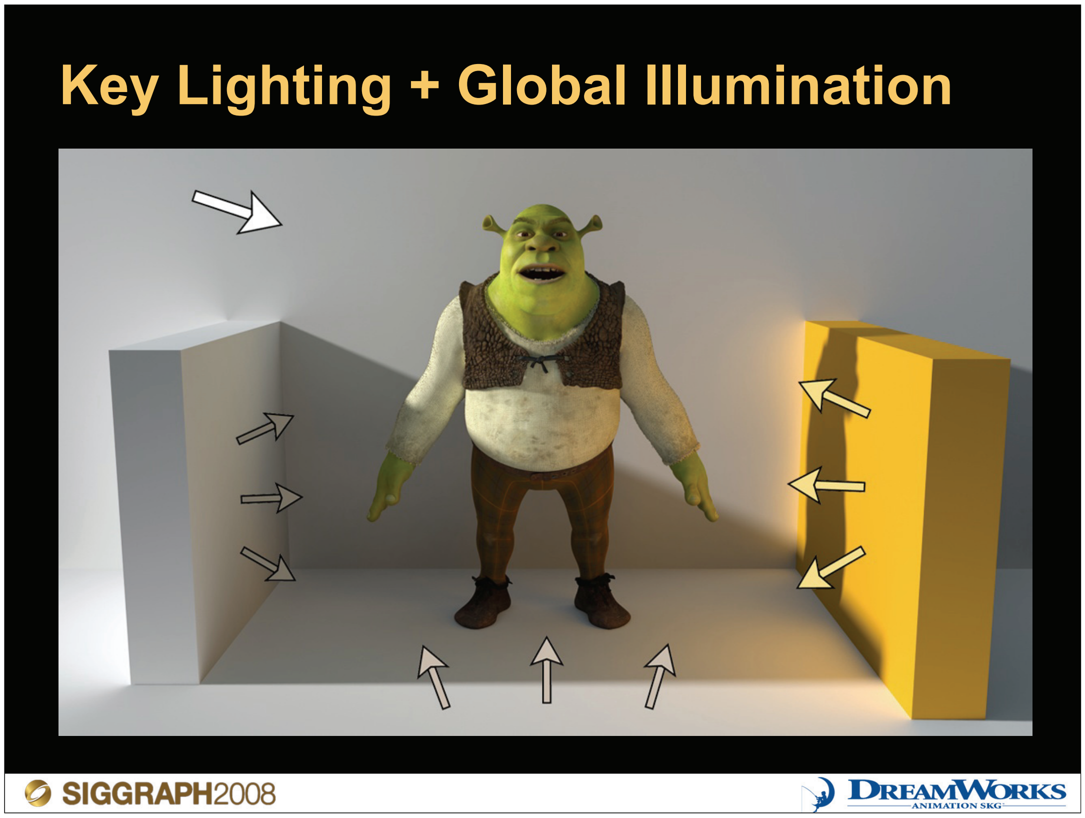
Key Lighting Only



SIGGRAPH2008 DreamWorks Animation SKG

Let’s go over a couple of examples to see our system in action.

Here is a simple example to illustrate direct and indirect illumination. In this image, direct illumination is coming from the left side of the image as indicated by the white arrow. Any surface area not exposed to the light directly is completely dark.

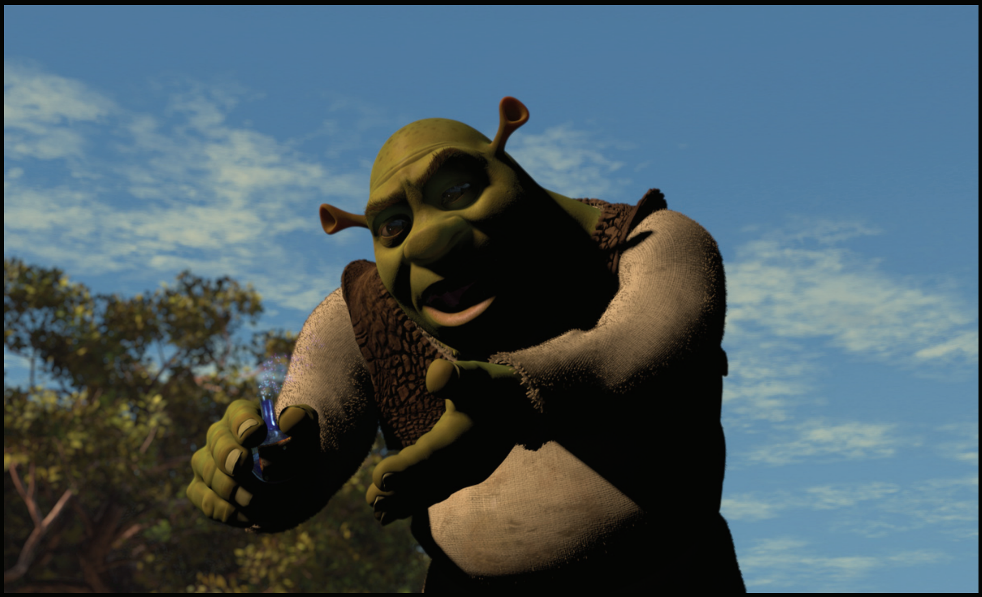


Key Lighting + Global Illumination

Once we let the direct light bounce on the geometry in the scene, we get this image. We call indirect illumination the secondary lighting that naturally fills up the dark areas and produces very soft-looking images. Notice also the color bleeding effects alongside the walls. You can compare by flipping back and forth between both slides.

One very nice feature bounce lighting provides for free, is the ability to control the lighting on a subject using distant off-screen bounce cards or reflectors. This offers a similar control to what a director of photography would expect on a live action set.

Key Lighting Only

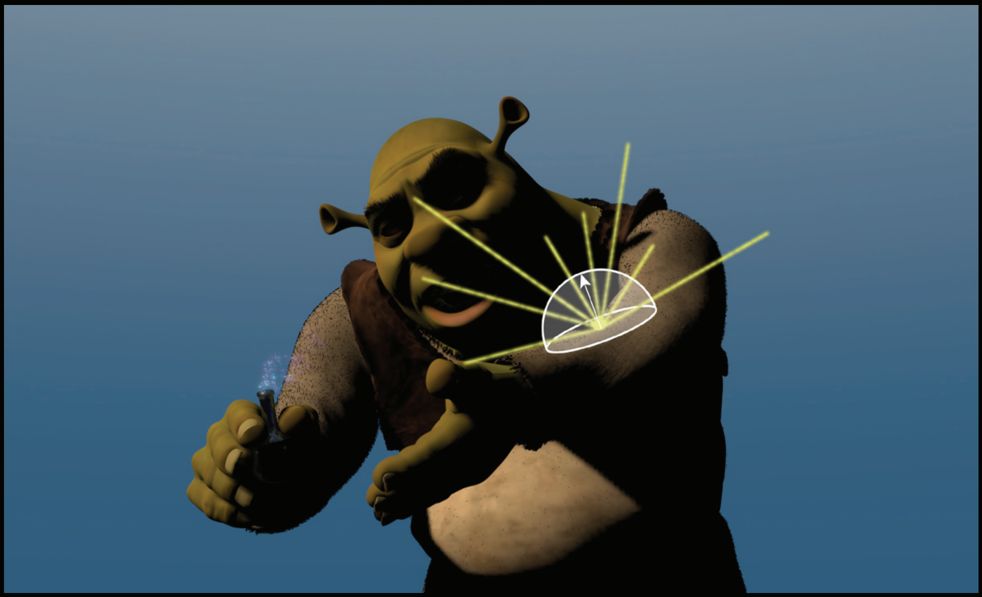


SIGGRAPH2008

DreamWorks Animation SKG

This example from the movie “Shrek2” illustrates final gathering using irradiance caching. As in the previous example, this image contains direct lighting only.

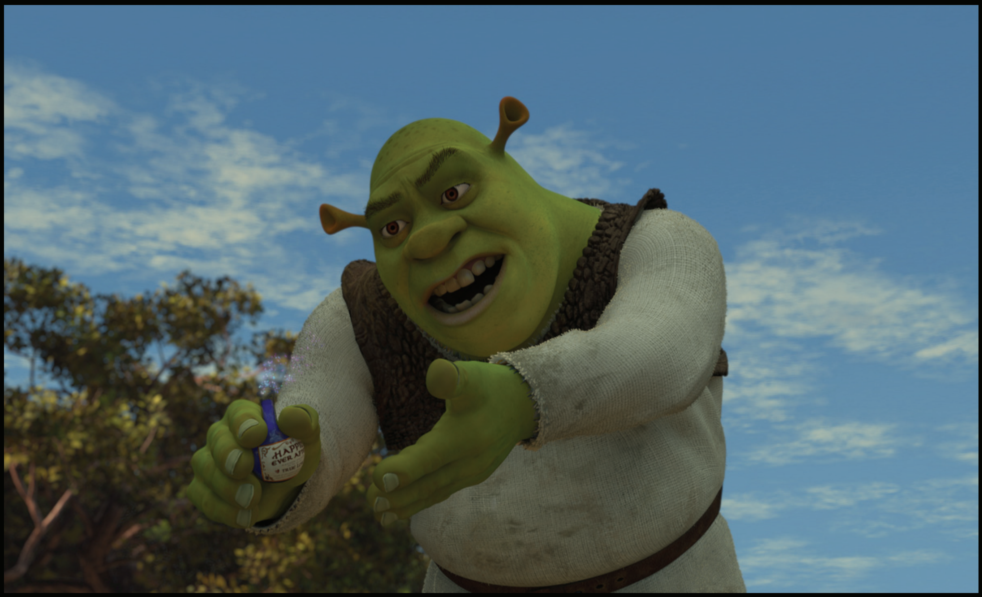
Light Gathering



Without using Irradiance Caching, gathering calculations happen for each pixel in the image where we want to compute indirect illumination. At each pixel we send several hundred rays from the surface seen through that pixel, with a cosine-weighted directional distribution over the hemisphere above it. The intersection of each ray is then shaded and the result of these evaluations are averaged together, producing an irradiance value.

This image illustrates a few of the rays cast to compute the irradiance for a single pixel on Shrek's arm. The image was intentionally rendered using the pre-computed radiosity textures that are used to shade each ray. Notice the overall blurry low-quality of the textures. These textures don't need to be pre-computed at very high resolution. In the background, there is also a textured sky-dome that rays can also intersect, as well as an off-screen ground plane.

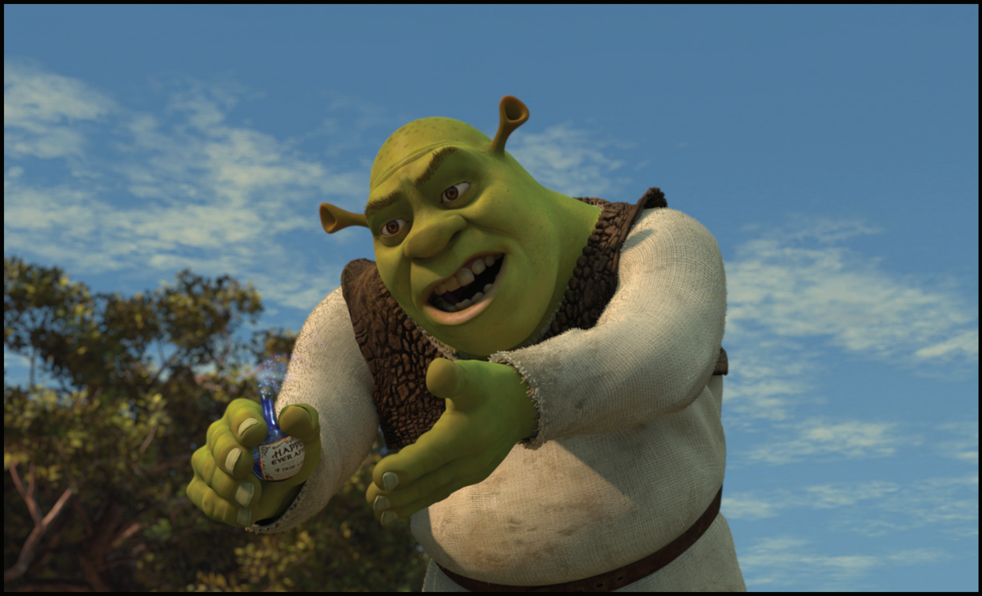
Indirect Lighting Only



SIGGRAPH2008 DREAMWORKS ANIMATION SKG

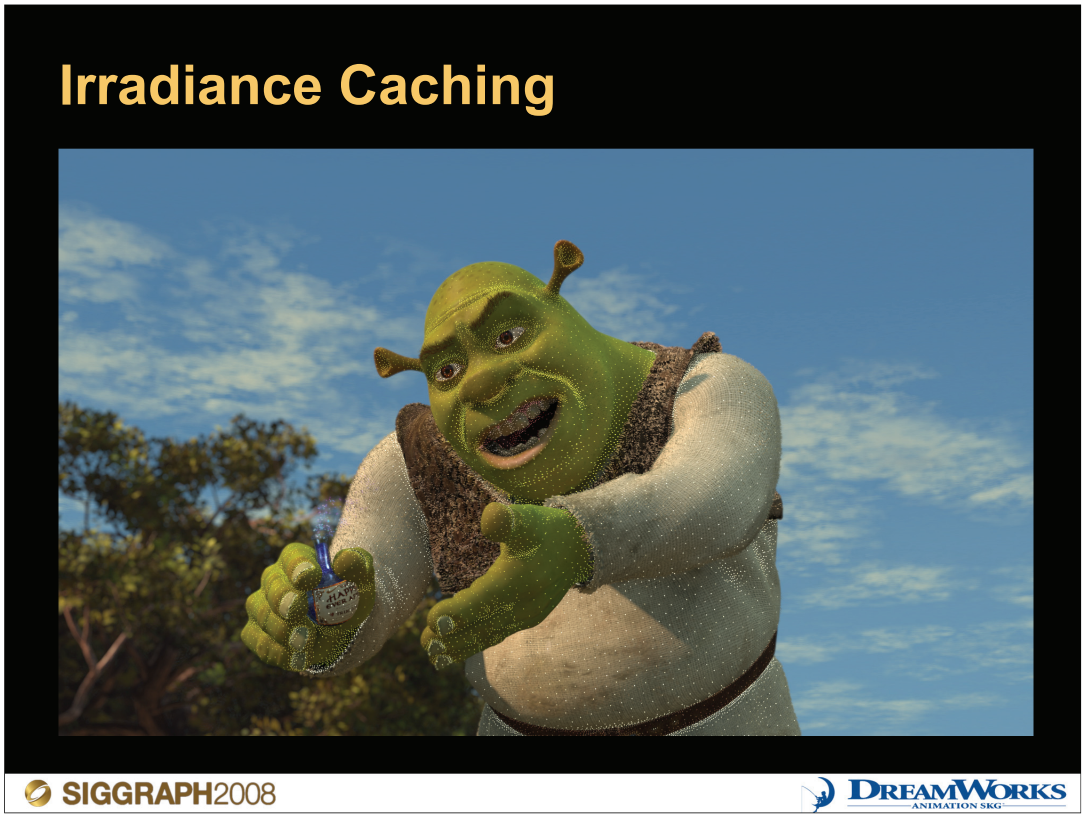
Here is the same image shaded only with indirect lighting, using the irradiance computed as described in the previous slide.

Key Lighting + Global Illumination



SIGGRAPH2008 DREAMWORKS ANIMATION SKG

And again, adding the direct lighting back in the equation.



Irradiance Caching

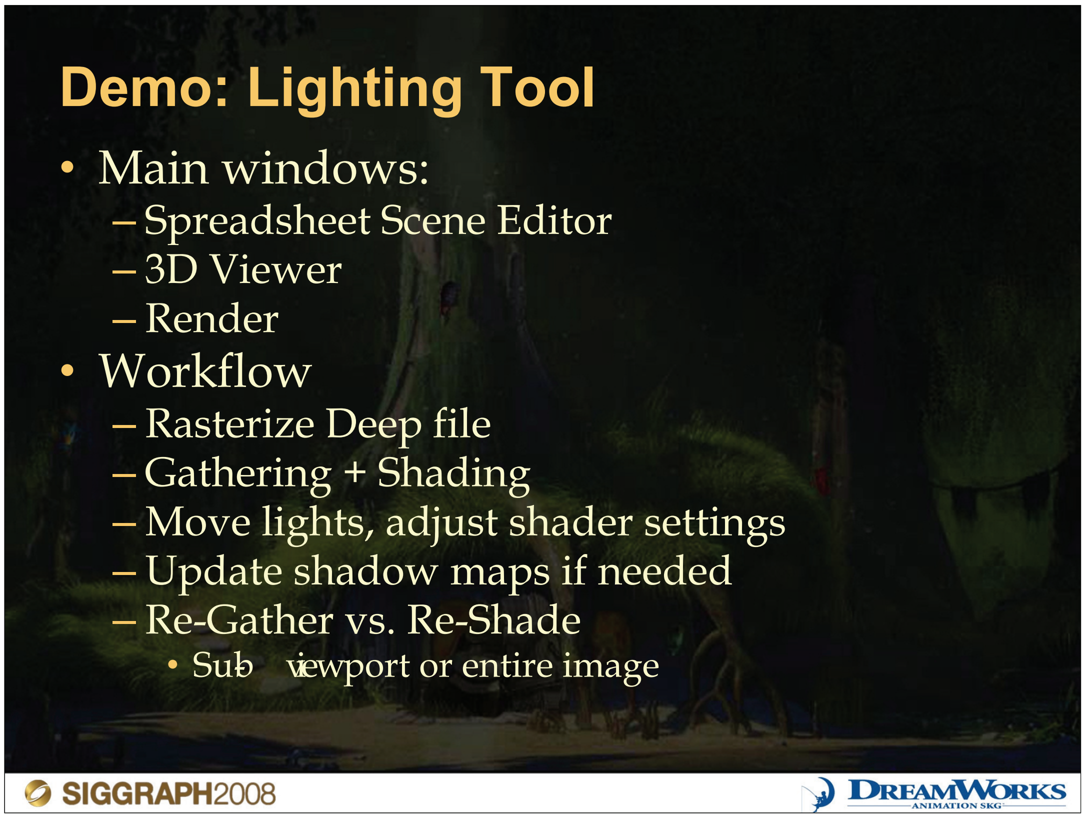
When we use irradiance caching, the expensive irradiance calculations are only performed at the "white dots" in this image and are cached for reuse. In between the white dots, we use the irradiance and its gradients from several neighbor cache records to smoothly interpolate the irradiance.

Notice the low density of the cache record distribution on surfaces that are smooth (belly, arms, hands, etc.). Also notice how dense the cache distribution can get in creases and on surfaces that are displacement-mapped in a way that produces high-frequency geometric detail (vest). We will talk in later slides how to remedy this problem, as well as what solution we use to deal with fur (eyebrows, medium-to-short fur and grass in general).

Irradiance Caching Integration

SIGGRAPH2008

DreamWorks Animation SKG



Demo: Lighting Tool

- Main windows:
  - Spreadsheet Scene Editor
  - 3D Viewer
  - Render
- Workflow
  - Rasterize Deep file
  - Gathering + Shading
  - Move lights, adjust shader settings
  - Update shadow maps if needed
  - Re-Gather vs. Re-Shade
    - Sub viewport or entire image

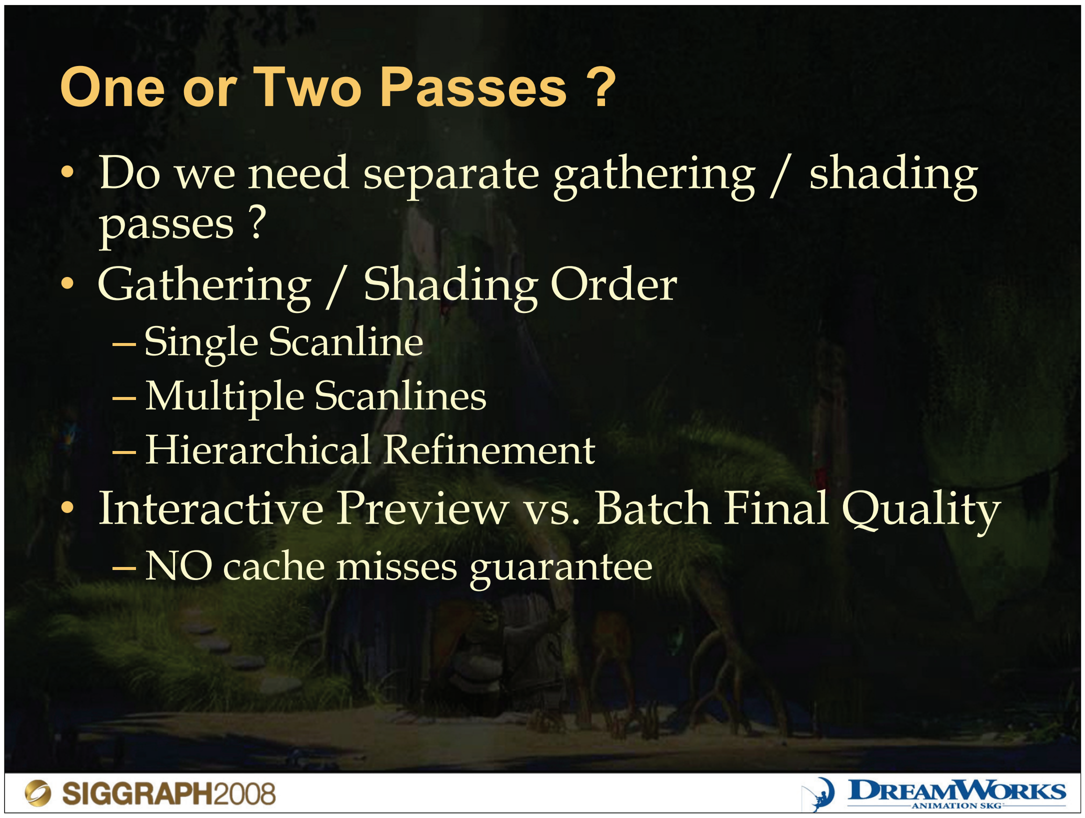
SIGGRAPH2008 DreamWorks Animation SKG

During the course we will show a video of our interactive lighting tool being used through a typical user workflow session.

The lighting tool allows the user to edit scene and shader parameters through a spreadsheet interface. A 3D viewer window lets the user handle light and camera placement as well as inspect geometry. A render window displays the rendered images and allows the user to do rendering tasks, such as rasterizing a deep file, updating shadow maps, and starting and stopping (re)gathering and/or (re)shading.

Irradiance caching works fits quite well within this workflow, specifically when the user wants to gather and re-shade many times specific sub-viewports and then finally update the entire image.

Explicitly requesting gathering triggers the irradiance cache to be flushed before the gathering/shading passes start. If the user only wants to re-shade, the pre-existing irradiance cache (if any) is used. Gathering may still happen if the user is shading a viewport that does not yet contain any/enough irradiance cache records.



One or Two Passes ?

- Do we need separate gathering / shading passes ?
- Gathering / Shading Order
  - Single Scanline
  - Multiple Scanlines
  - Hierarchical Refinement
- Interactive Preview vs. Batch Final Quality
  - NO cache misses guarantee

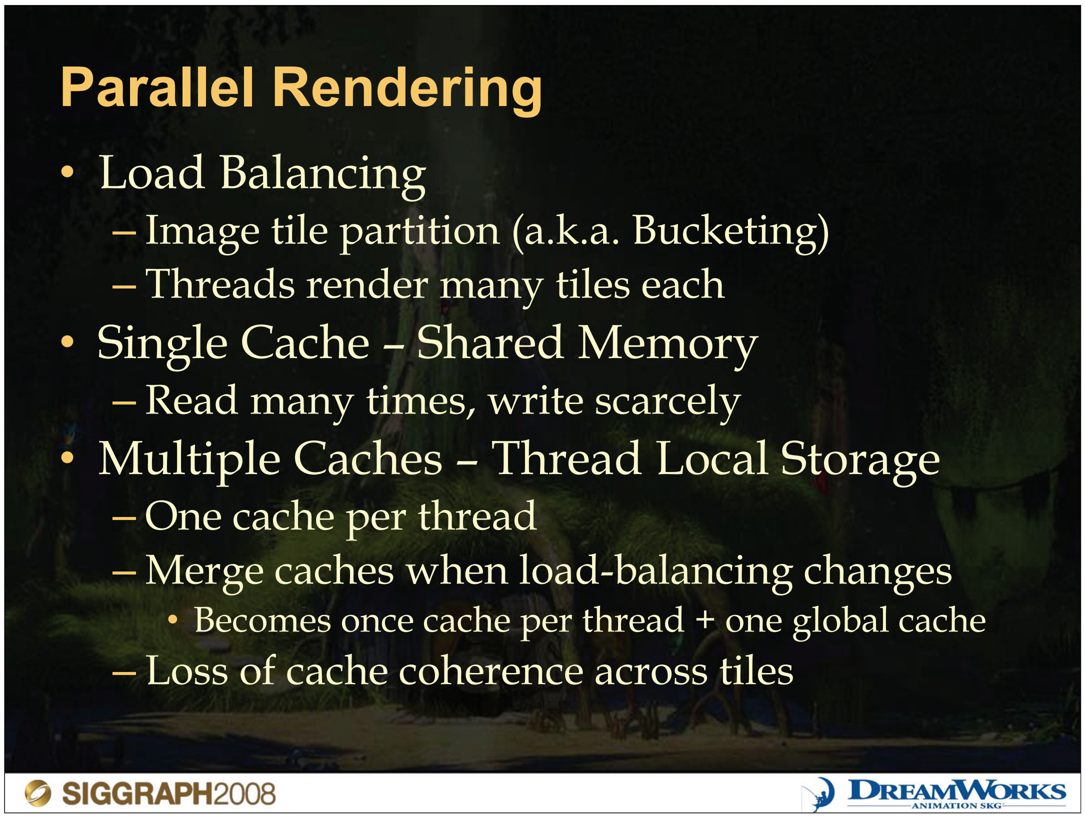
SIGGRAPH2008
DreamWorks Animation SKG

One question that arises with using Irradiance Caching, is whether a separate shading pass is needed after the gathering pass, or if both gathering and final shading can happen in the same pass. The answer mainly depends if the user is doing interactive preview work or if final quality rendering is required.

Another important factor is the gathering/shading order. When the gathering/shading calculations happen along a scanline (or multiple scanlines when rendering on a multi-core CPU), irradiance cache records are created as the scanline progresses. In this scheme, notice that cache records only affect interpolated irradiance for subsequent scanlines (and not scanlines that preceded their creation). This usually causes noticeable irradiance discontinuity artifacts, and impacts the artist's workflow. When rendering with scanlines, we found that a separate final shading pass is needed, even for preview work.

When rendering using hierarchical refinement however, the image is scanned multiple times across several sub-passes. The first sub-pass shades only every few pixels and every few scanlines yet displays the whole image with reduced detail in a pixelated fashion. Each subsequent sub-pass shades more and more pixels in between the subset of pixels shaded during previous sub-passes, thereby reducing pixelation and revealing more and more image detail. In this scheme, the same discontinuity problem happens within each sub-pass, but the visual artifacts are greatly reduced by the fact that we use many sub-passes. This makes it acceptable to use a single gathering/shading pass (made of many sub-passes) for preview work.

For final quality rendering, we have an important requirement: no cache misses and no cache record creation can happen during final shading. This would otherwise introduce irradiance discontinuities in the image plane as described above, which then translates into temporal flickering and popping in animations. We therefore use a separate gathering pass before the final shading pass.



Parallel Rendering

- Load Balancing
  - Image tile partition (a.k.a. Bucketing)
  - Threads render many tiles each
- Single Cache – Shared Memory
  - Read many times, write scarcely
- Multiple Caches – Thread Local Storage
  - One cache per thread
  - Merge caches when load-balancing changes
    - Becomes once cache per thread + one global cache
  - Loss of cache coherence across tiles

SIGGRAPH2008 DREAMWORKS ANIMATION SKG

Our renderer can render images in parallel, thereby leveraging the power of multi-core CPUs. We use multiple threads of execution and a divide and conquer strategy. We actually use multi-processing and selective explicit shared memory (for several practical reasons) in a way that is very much similar to multi-threading. For the clarity of the explanation we will refer to multi-threading since most people are familiar with this parallel programming model.

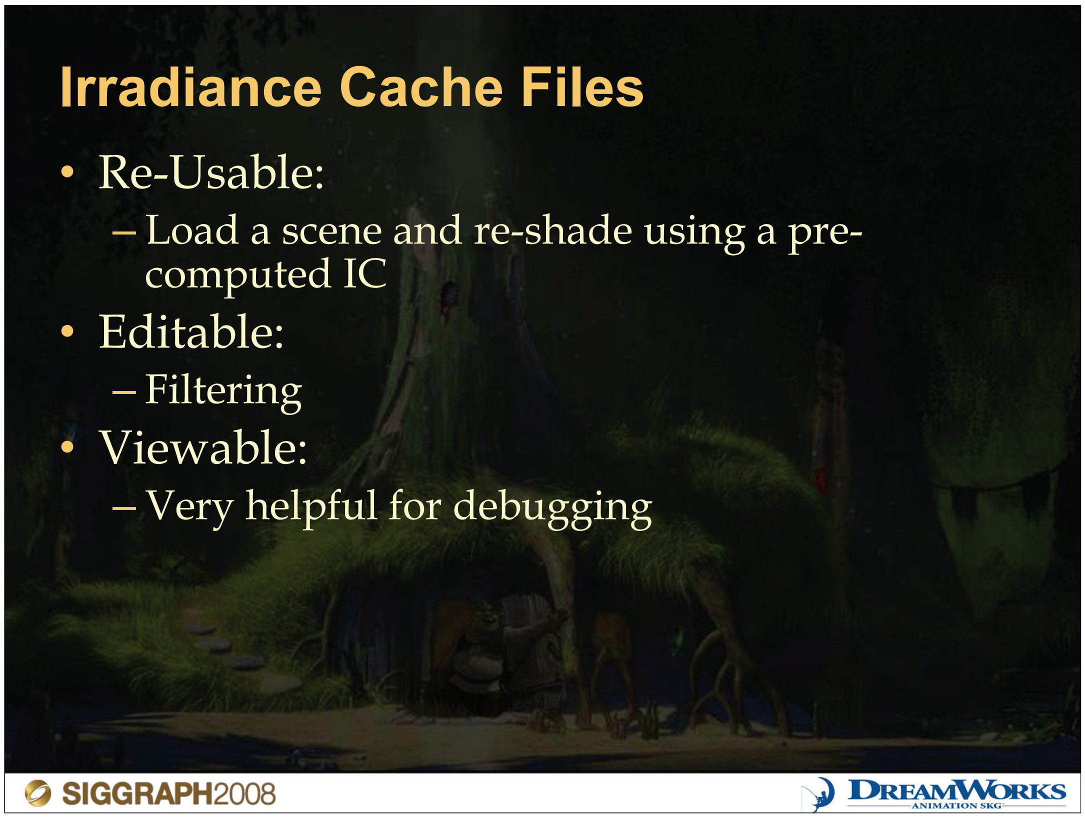
As is common practice, we partition the image into several shading tiles (or buckets), and each thread is responsible for rendering as many buckets as it can in a greedy fashion. The same parallel rendering main-loop is used when using irradiance caching. Notice that the irradiance cache becomes a read-write shared resource across threads, since irradiance cache records can contribute to shading multiple adjacent tiles handled by different threads. This resource sharing can be dealt with in several ways, using classic parallelization techniques.

The simplest approach is to use a single irradiance cache in shared memory and rely on a read-write lock resource sharing mechanism, since the cache will be read from many times and written to scarcely. This works well but places a small contention point and may impact scalability with growing number of CPU-cores and threads.

Another alternative is to use multiple irradiance caches. This may be desirable if we want to remove the contention point altogether, or necessary in cases where the parallel execution environment doesn't provide an efficient shared memory system. In this case each thread can manage its own separate irradiance cache, which ends up being populated with cache records that cover the tiles shaded by the thread.

When considering a multiple cache approach in the context of an interactive lighting tool, some difficulties arise. In such a tool, we might be asking our threads to shade different sub-viewports, and therefore the load-balancing might assign each thread with a different set of image tiles across successive reshades. When the load balancing changes, it becomes necessary to merge all of the threads' individual caches together into a global irradiance cache, which then becomes available to the threads for subsequent gathering/shading. Care must also be taken when merging thread local caches again and again, to avoid redundant duplication of cache records originating from a previous merge (as opposed to newly created cache records).

This approach doesn't come without its own inefficiencies. In a multiple cache scheme, cache records are only re-used within areas of the image shaded by a given thread. In essence, we do loose a bit of cache coherence across tile boundaries, and some extra work will be spent creating cache records on each side of tile boundaries. It is then crucial to choose a big-enough tile size in order to minimize this loss of cache coherence. Unfortunately this is contrary to using small enough tiles for the sake of good load balancing, and therefore also limits the scalability of the algorithm with growing number of CPU-cores.



Irradiance Cache Files

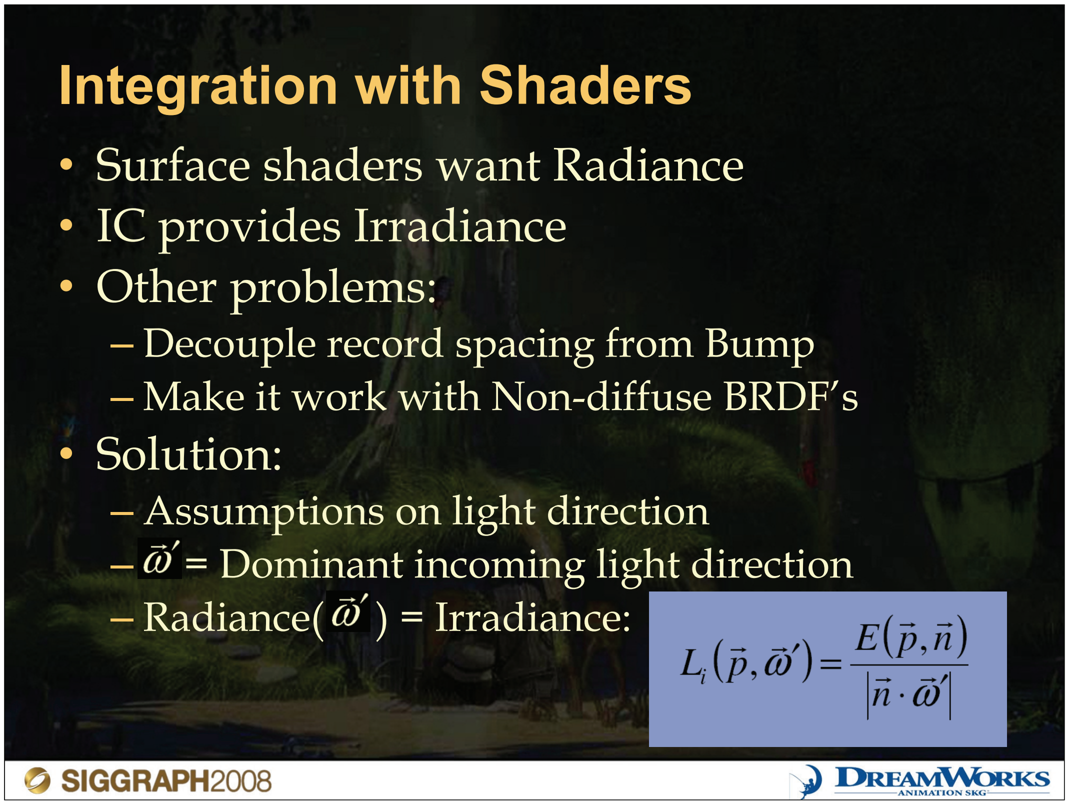
- Re-Usable:
  - Load a scene and re-shade using a pre-computed IC
- Editable:
  - Filtering
- Viewable:
  - Very helpful for debugging

SIGGRAPH2008

During the course we will show a video of our interactive tools used to edit, and inspect irradiance cache files using a 3D viewer.

Being able to read and write irradiance cache files has been very valuable for various reasons. The most obvious one is the ability to re-use an irradiance cache created in a previous interactive lighting session. The lighters often open a shot and shade an image multiple times using a previously computed irradiance cache.

Irradiance cache files also make the irradiance cache editable, viewable, and even playable over time, which has been very useful for debugging purposes.



Integration with Shaders

- Surface shaders want Radiance
- IC provides Irradiance
- Other problems:
  - Decouple record spacing from Bump
  - Make it work with Non-diffuse BRDF's
- Solution:
  - Assumptions on light direction
  - $\vec{\omega}'$ = Dominant incoming light direction
  - Radiance($\vec{\omega}'$) = Irradiance:

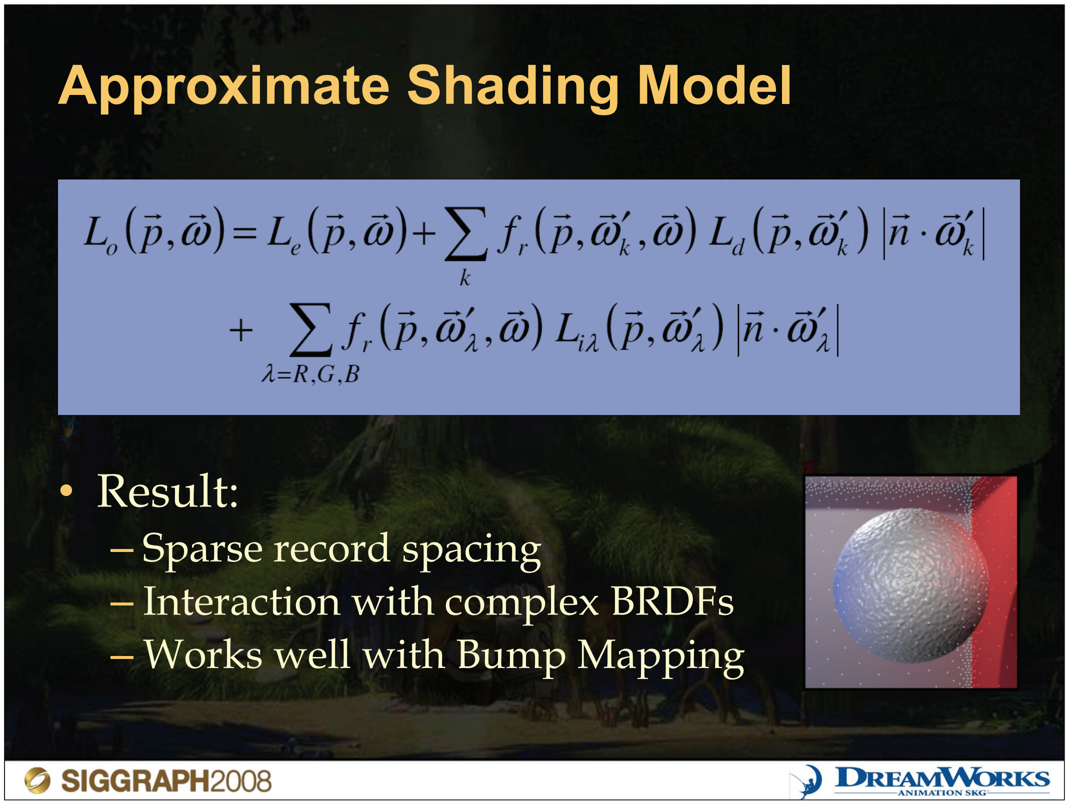
$$L_i(\vec{p},\vec{\omega}') = \frac{E(\vec{p},\vec{n})}{|\vec{n}\cdot\vec{\omega}'|}$$

SIGGRAPH2008
DREAMWORKS ANIMATION SKG

Integrating irradiance caching in a rendering system that uses programmable shaders needs to be considered. Most shading systems support surface shaders and light shaders. The latter are most often responsible for computing illumination, and the latter for applying a BRDF. Typically, light shaders provide surface shaders with sets of (radiance, direction) value pairs. Irradiance caching however provides irradiance quantities which are not associated to any specific incoming direction.

Furthermore we want irradiance caching record spacing to be independent of the bump-mapped normal variation, yet we want bump-mapped surfaces that are lit with irradiance caching to still look like they have some bump. Finally, even though irradiance caching assumes purely diffuse BRDF's in many ways, we also want glossy surfaces lit with irradiance caching to still look reasonably glossy.

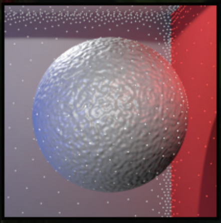
To remedy / alleviate the three problems above, we use a simple assumption which is far from fool-proof but works well in practice. During the gathering process, we keep track of the dominant incoming light direction (also called "bent normal" in the literature) by simply computing an average of ray directions over the hemisphere, weighted by their contribution to the corresponding irradiance value. We also store this direction in the irradiance cache, and interpolate it similarly to how we interpolate the irradiance. Our assumption is then to assume that all of the lighting contributing to the cached irradiance comes from this direction. With this assumption it is then trivial to compute the corresponding radiance and feed this information to surface shaders, which then use the bumped normal in their BRDF evaluations.



Approximate Shading Model

$$L_o(\vec{p},\vec{\omega}) = L_e(\vec{p},\vec{\omega}) + \sum_k f_r(\vec{p},\vec{\omega}'_k,\vec{\omega})\, L_d(\vec{p},\vec{\omega}'_k)\, \left|\vec{n}\cdot\vec{\omega}'_k\right|$$
$$+ \sum_{\lambda=R,G,B} f_r(\vec{p},\vec{\omega}'_\lambda,\vec{\omega})\, L_{i\lambda}(\vec{p},\vec{\omega}'_\lambda)\, \left|\vec{n}\cdot\vec{\omega}'_\lambda\right|$$

- Result:
  - Sparse record spacing
  - Interaction with complex BRDFs
  - Works well with Bump Mapping



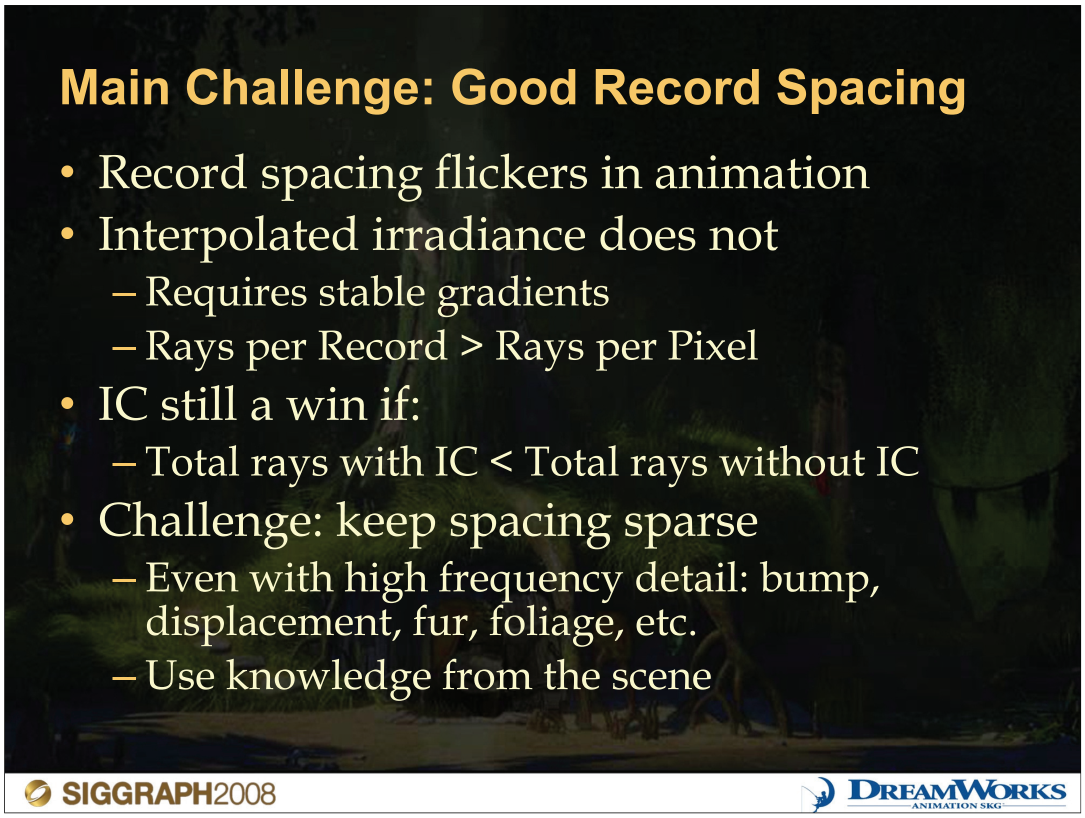
SIGGRAPH2008 DREAMWORKS ANIMATION SKG

Formally speaking, the formula above defines our approximate shading model based on the description of the previous slide. In this formula, we also decompose the incoming light direction calculations over three “wavelength” (the three RGB channels in fact), which provides three (radiance, direction) value pairs for each pixel (one for each wavelength). This color separation trick tends to increase chromaticity effects on bumped glossy surfaces, but may also cause color separation which is undesirable. It is not always used and can be dialed in/out in our system.

The result of this model is shown on a bumped glossy sphere within a cornell box. Notice the sparcity of the record spacing and the glossiness of the sphere appearance. The sphere here is only lit with indirect illumination using irradiance caching and the shading model we described.

Implementation Details

SIGGRAPH2008
DreamWorks Animation SKG



Main Challenge: Good Record Spacing

- Record spacing flickers in animation
- Interpolated irradiance does not
  - Requires stable gradients
  - Rays per Record > Rays per Pixel
- IC still a win if:
  - Total rays with IC < Total rays without IC
- Challenge: keep spacing sparse
  - Even with high frequency detail: bump, displacement, fur, foliage, etc.
  - Use knowledge from the scene

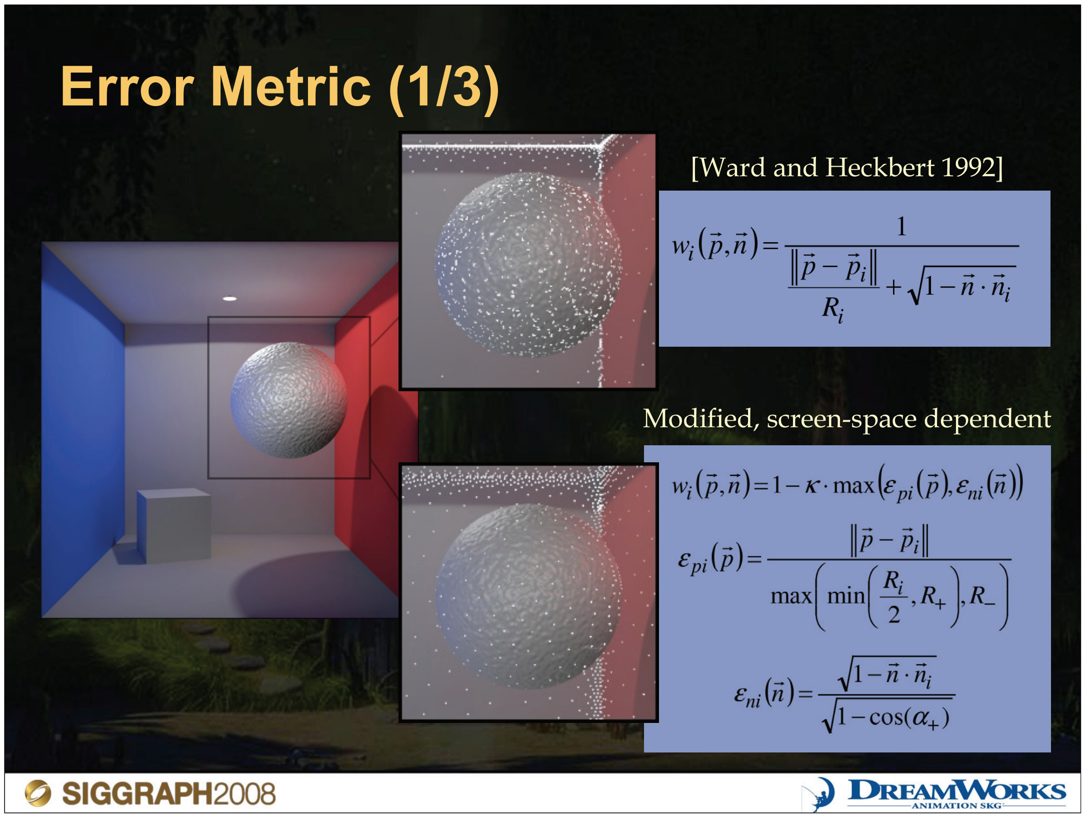
SIGGRAPH2008 DreamWorks Animation SKG

In this section we will talk about the main challenge in implementing irradiance caching: providing a good record spacing quality.

As we've seen, it is possible to render stable non-flickering animations using irradiance caching, even though the underlying cache record distribution changes as can be seen in playbacks of images rendered with irradiance cache dots. At first, it is in fact hard to imagine that a stable result is possible with such a flickering distribution.

Stable results require accurate irradiance values and stable gradients, and in turn may require more rays per cache record, than might be needed per pixel if we were not to use irradiance caching. Even though, using irradiance caching is still a win as long as the record spacing remains sparse enough to offset this cost.

In the next few slides we will talk about the few extra tricks we use to control the record spacing distribution, and show how we can use scene-knowledge to exploit underlying geometry smoothness even in cases with high frequency displacement maps or fur.



Error Metric (1/3)

[Ward and Heckbert 1992]

$$w_i(\vec{p},\vec{n}) = \frac{1}{\frac{\|\vec{p}-\vec{p}_i\|}{R_i} + \sqrt{1-\vec{n}\cdot\vec{n}_i}}$$

Modified, screen-space dependent

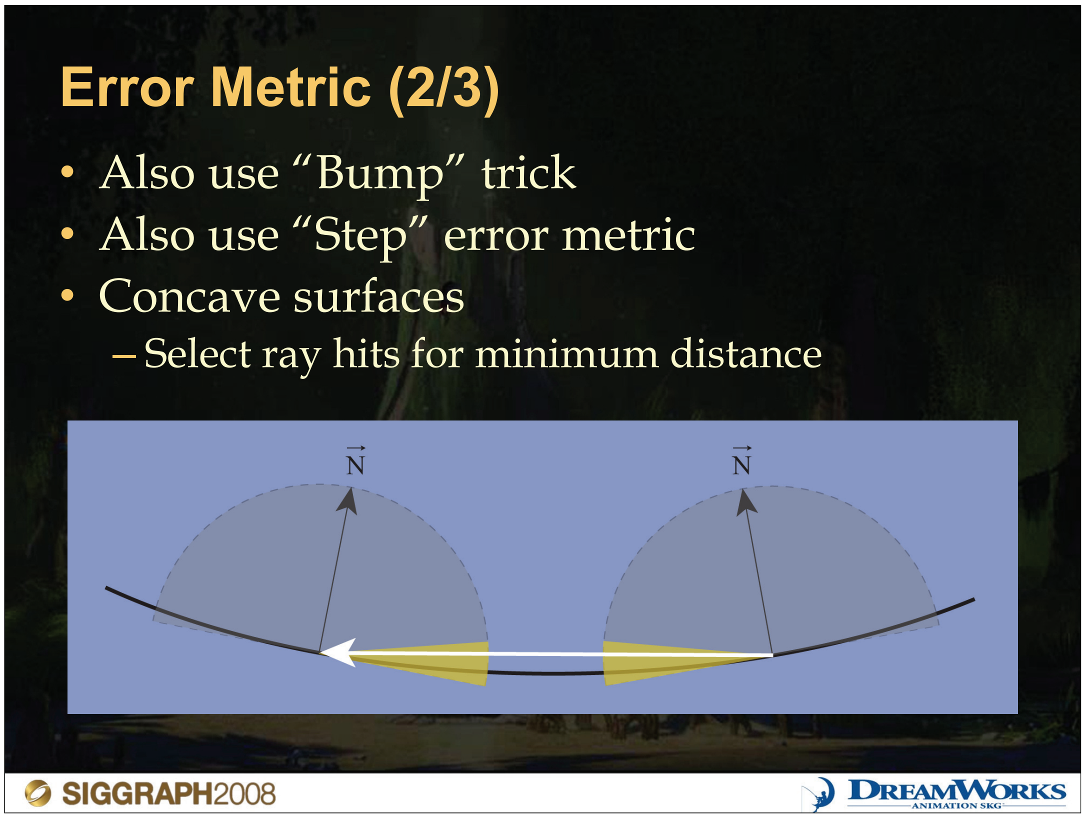
$$w_i(\vec{p},\vec{n}) = 1 - \kappa \cdot \max\left(\varepsilon_{pi}(\vec{p}), \varepsilon_{ni}(\vec{n})\right)$$

$$\varepsilon_{pi}(\vec{p}) = \frac{\|\vec{p}-\vec{p}_i\|}{\max\left(\min\left(\frac{R_i}{2}, R_+\right), R_-\right)}$$

$$\varepsilon_{ni}(\vec{n}) = \frac{\sqrt{1-\vec{n}\cdot\vec{n}_i}}{\sqrt{1-\cos(\alpha_+)}}$$

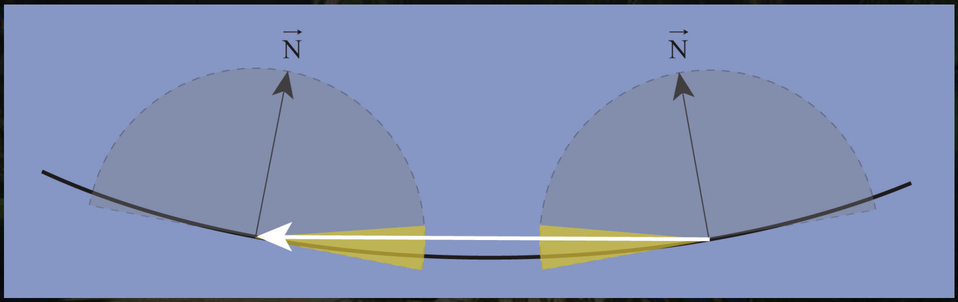
SIGGRAPH2008 DreamWorks ANIMATION SKG

As described in our paper [TL04], we use a different error metric than the one proposed by [Ward and Heckbert 1992]. It allows us to control the record spacing with a screen-space metric which allows us to bound the minimum and maximum screen-space distance between cache records. As emphasized by previous speakers in this course, this has many advantages especially in scenes with wide depth range or when considering the world-space scale invariance of the metric. We also use a slightly more intuitive normal error metric, normalized once the normal deviation reaches a specific maximum angle.



Error Metric (2/3)

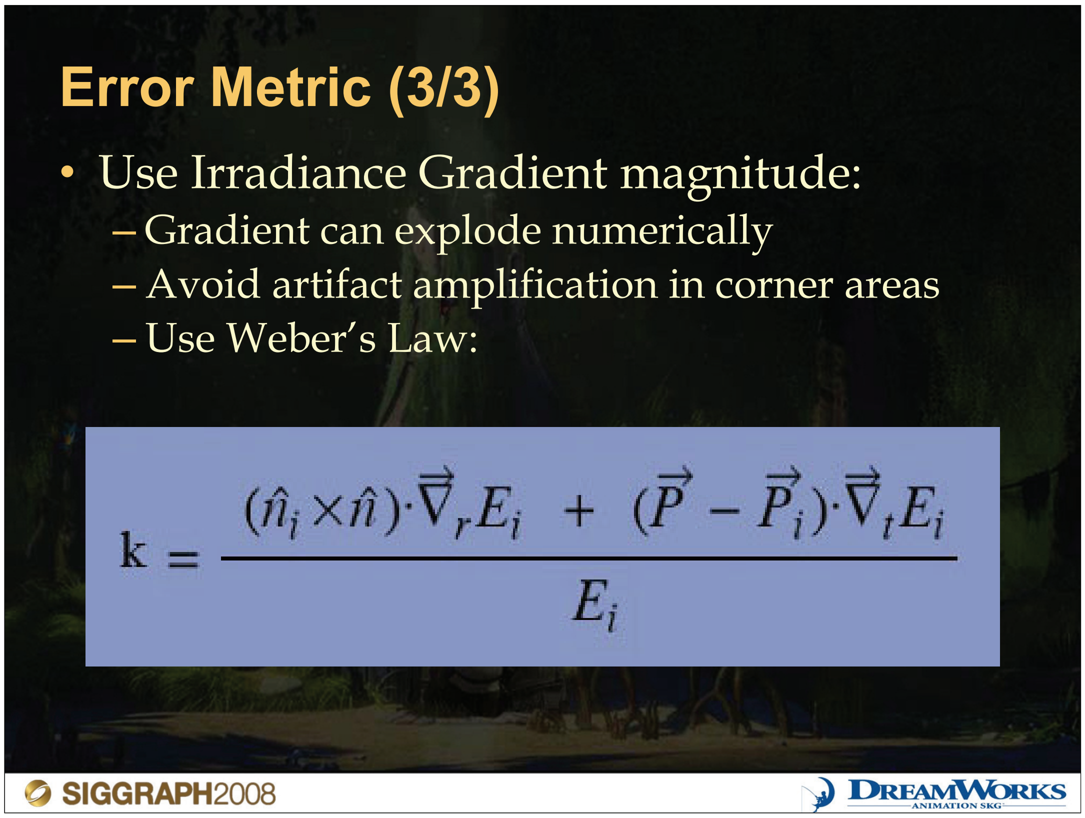
- Also use “Bump” trick
- Also use “Step” error metric
- Concave surfaces
  - Select ray hits for minimum distance



SIGGRAPH2008 DreamWorks Animation SKG

Like other implementations, we also use the bump mapping trick and the “step” error metric described previously in this course.

Additionally we carefully deal with concave surfaces to prevent unnecessarily dense record spacing. We add an extra test to select rays when computing the minimum distance Ri. We simply use a threshold on the angle between the ray direction and the surface normal both at the gathering point and at the ray intersection point. In the case we fall above the threshold for both angles, we do not discard the corresponding radiance value, but we discard the ray’s intersection distance from contributing to Ri. This helps achieve a coarse record spacing distribution on “slightly” concave surfaces. How slightly depends on the value chosen for the angle threshold.



Error Metric (3/3)

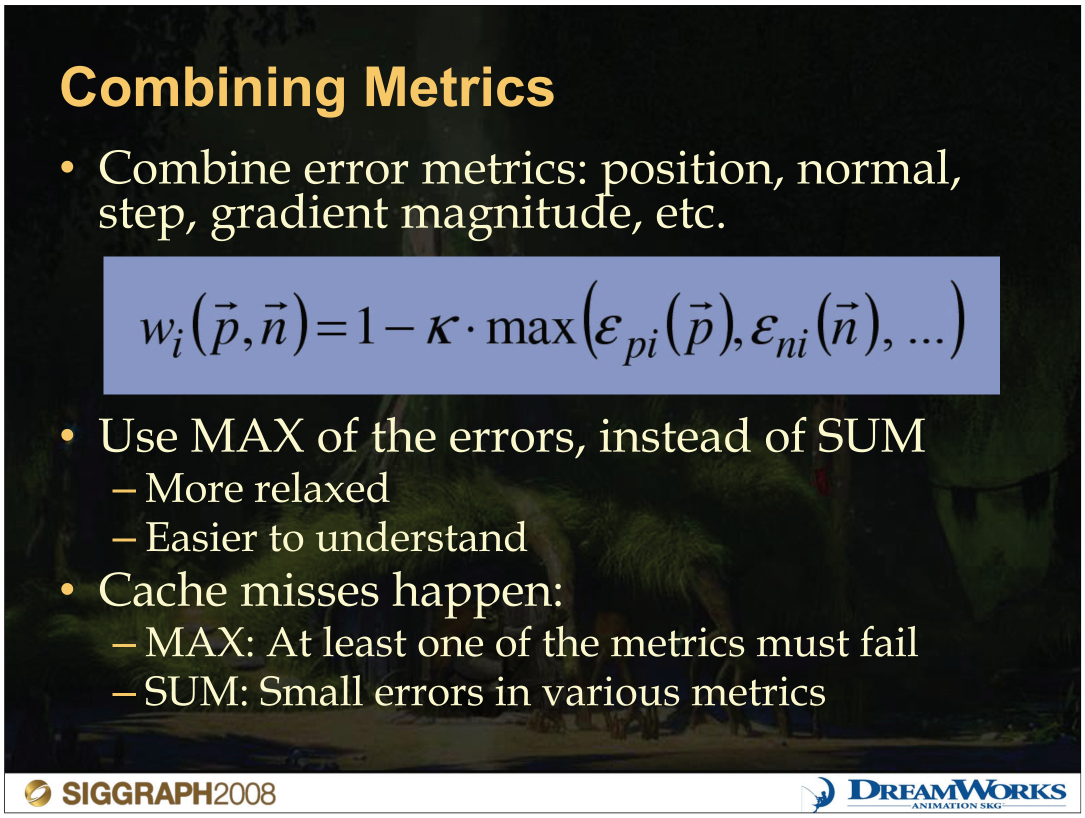
- Use Irradiance Gradient magnitude:
  - Gradient can explode numerically
  - Avoid artifact amplification in corner areas
  - Use Weber's Law:

$$k = \frac{(\hat{n}_i \times \hat{n}) \cdot \vec{\nabla}_r E_i + (\vec{P} - \vec{P}_i) \cdot \vec{\nabla}_t E_i}{E_i}$$

SIGGRAPH2008 DREAMWORKS ANIMATION SKG

Another problem that happens frequently with irradiance caching, is the numerical explosion of the translation gradient vector's magnitude. This is the case when ray intersection distances become really small as in corner areas, as the formulation of the gradient requires dividing by the ray intersection distance.

We tried arbitrarily capping the magnitude of the gradient vectors, but this results in under or over estimation of the irradiance in corners. What we found works quite well is to limit the interpolation from cache records based on the magnitude of the gradient. We add an extra term to our error metric formula, and limit how far a cache record can be reused based on Weber's law. We simply threshold on the ratio between the irradiance delta coming from applying the gradient at a specific position and the sampled irradiance stored in the cache record.



Combining Metrics

- Combine error metrics: position, normal, step, gradient magnitude, etc.

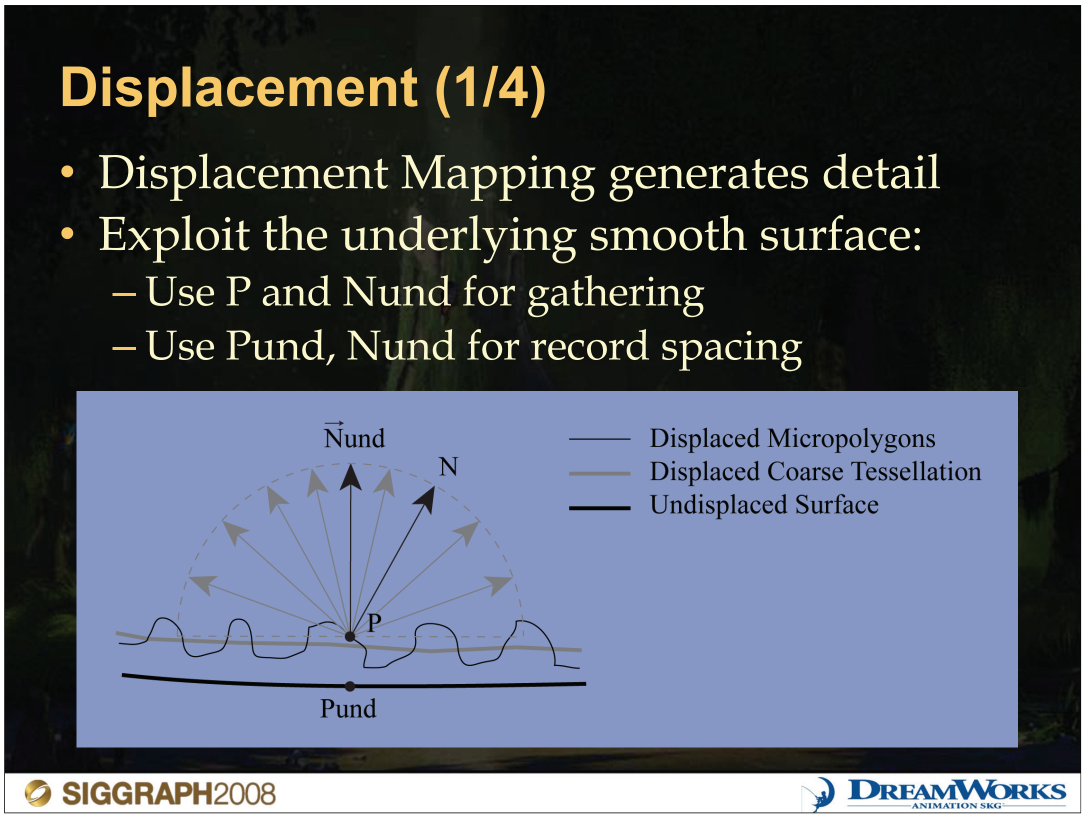
$$w_i(\vec{p},\vec{n}) = 1 - \kappa \cdot \max\left(\varepsilon_{pi}(\vec{p}), \varepsilon_{ni}(\vec{n}), \ldots\right)$$

- Use MAX of the errors, instead of SUM
  - More relaxed
  - Easier to understand
- Cache misses happen:
  - MAX: At least one of the metrics must fail
  - SUM: Small errors in various metrics

SIGGRAPH2008 DreamWorks Animation SKG

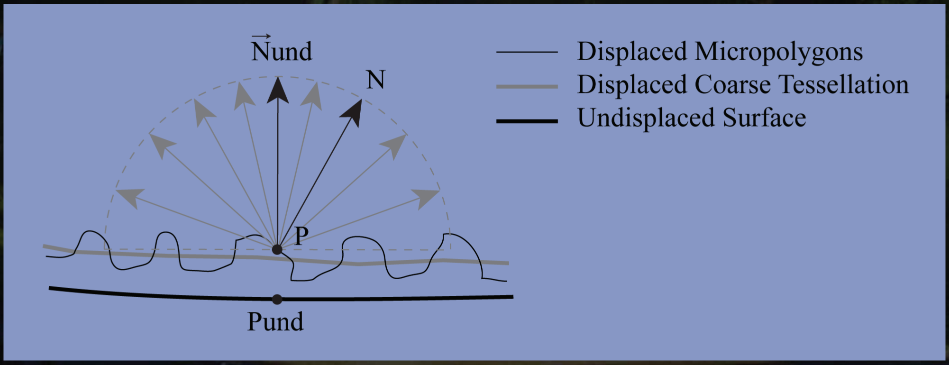
To make the model a bit easier to understand, we use a combination of error metrics that are all normalized between 0 and 1. When gathering / shading a particular pixel, we consider a set of candidate cache records to interpolate from. When one of the error metrics reaches 1, the corresponding cache record is discarded. If all candidate cache records are discarded a cache miss happens.

Combining our error metrics that way provides a more relaxed approach, where at least one of the error metrics must fully fail in order to discard a cache record.



Displacement (1/4)

- Displacement Mapping generates detail
- Exploit the underlying smooth surface:
  - Use P and Nund for gathering
  - Use Pund, Nund for record spacing



SIGGRAPH2008 DREAMWORKS ANIMATION SKG

We now describe how we can maintain a sparse record spacing distribution on surfaces that are displacement mapped.

Displacement mapping an produce surfaces with high frequency geometric detail and rapidly varying surface normal and even cause abrupt surface normal discontinuities. One possible solution is to exploit the smoothness of the underlying un-displaced surface:

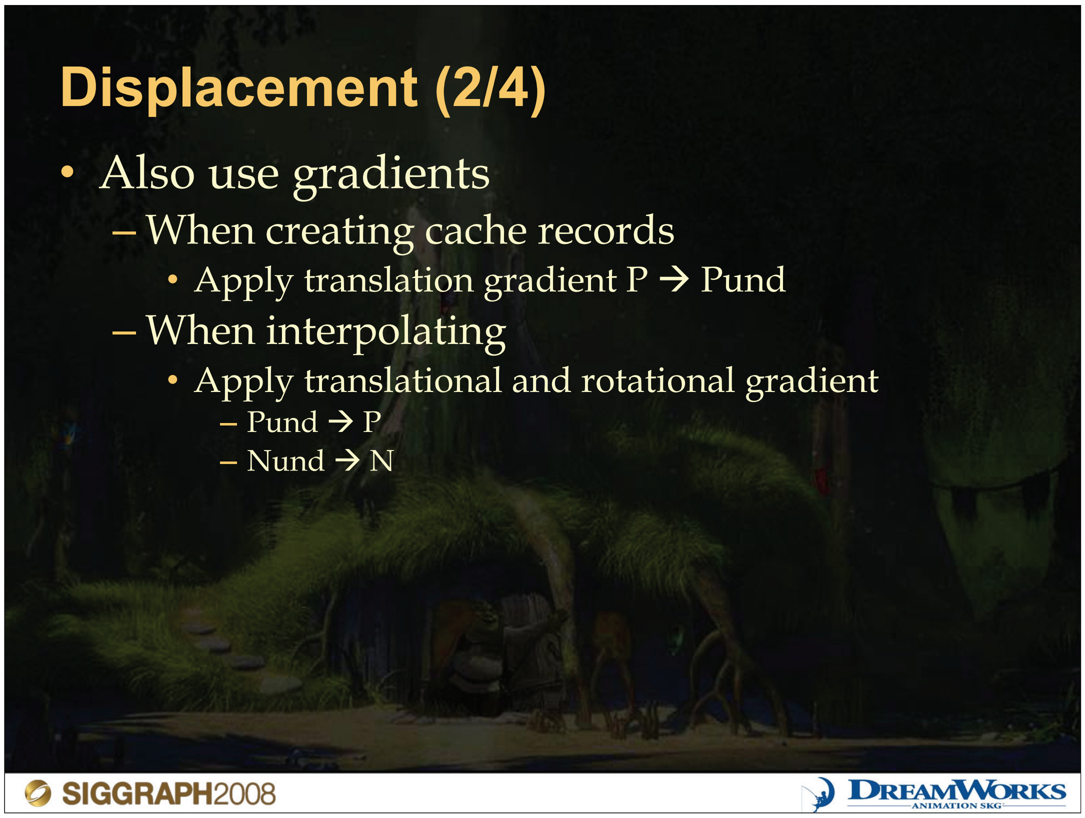
•Cache records are stored on the un-displaced surface,

•Cache lookups happen on the un-displaced surface,

•The irradiance interpolation should be careful to apply gradient vectors to extrapolate the irradiance to the displaced shading position and normal.

When sampling the irradiance however, we want to use the displaced position as ray origins to avoid obvious self-intersection problems. We also want the hemisphere of ray directions to be aligned on the un-displaced surface normal. This is so our irradiance gradient calculations align with the un-displaced smooth surface.

For this to be possible, we simply store in our deep file the un-displaced surface position and normal for each displaced micropolygon vertex.

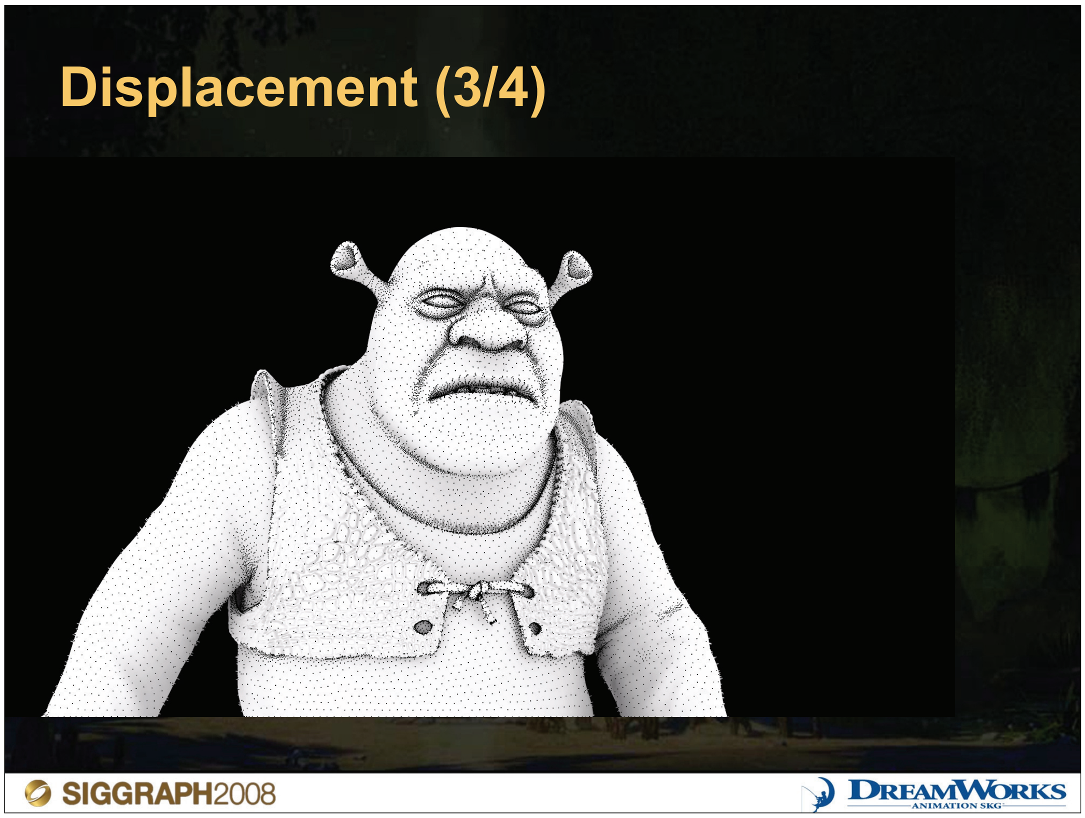


Displacement (2/4)

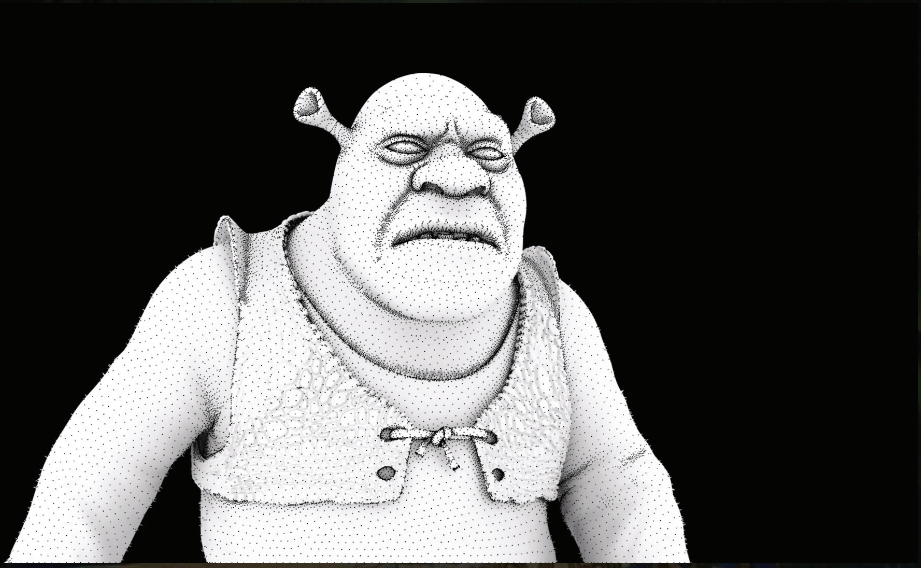
- Also use gradients
  - When creating cache records
    - Apply translation gradient P → Pund
  - When interpolating
    - Apply translational and rotational gradient
      - Pund → P
      - Nund → N

SIGGRAPH2008 DreamWorks Animation SKG

In the case where displacement mapping can move surface vertices along any direction (not just along the surface normal), it may be beneficial for each new cache record to estimate the irradiance value at the cache record location on the un-displaced surface. For this purpose we can use the sampled irradiance value on the displaced surface and the translation irradiance gradients.



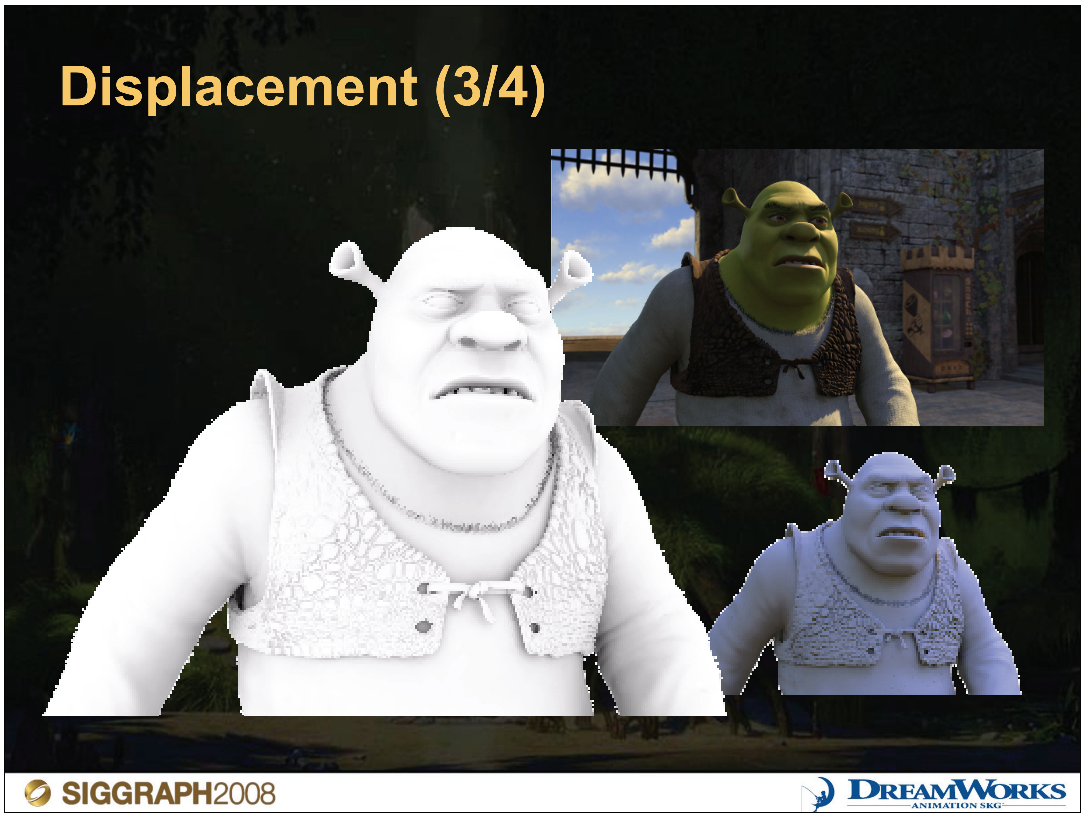
Displacement (3/4)



SIGGRAPH2008

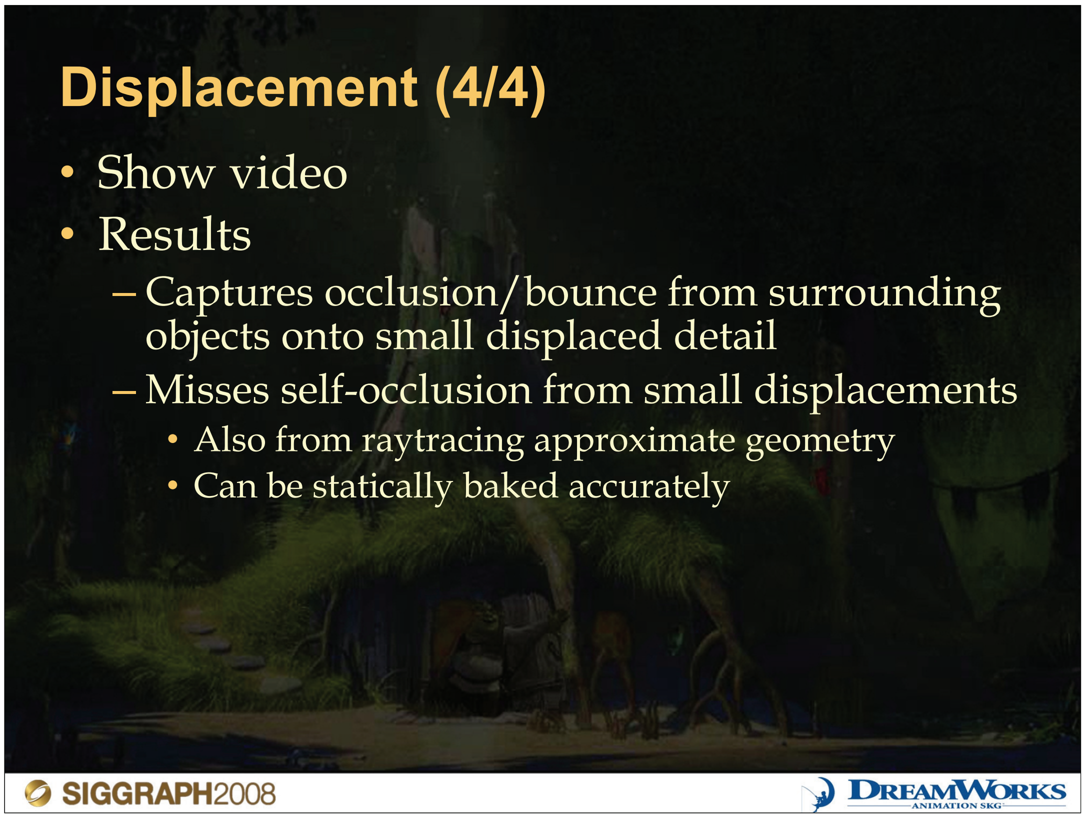
DREAMWORKS ANIMATION SKG

And here is what the record spacing looks like.



Displacement (3/4)

Here is the image you get…



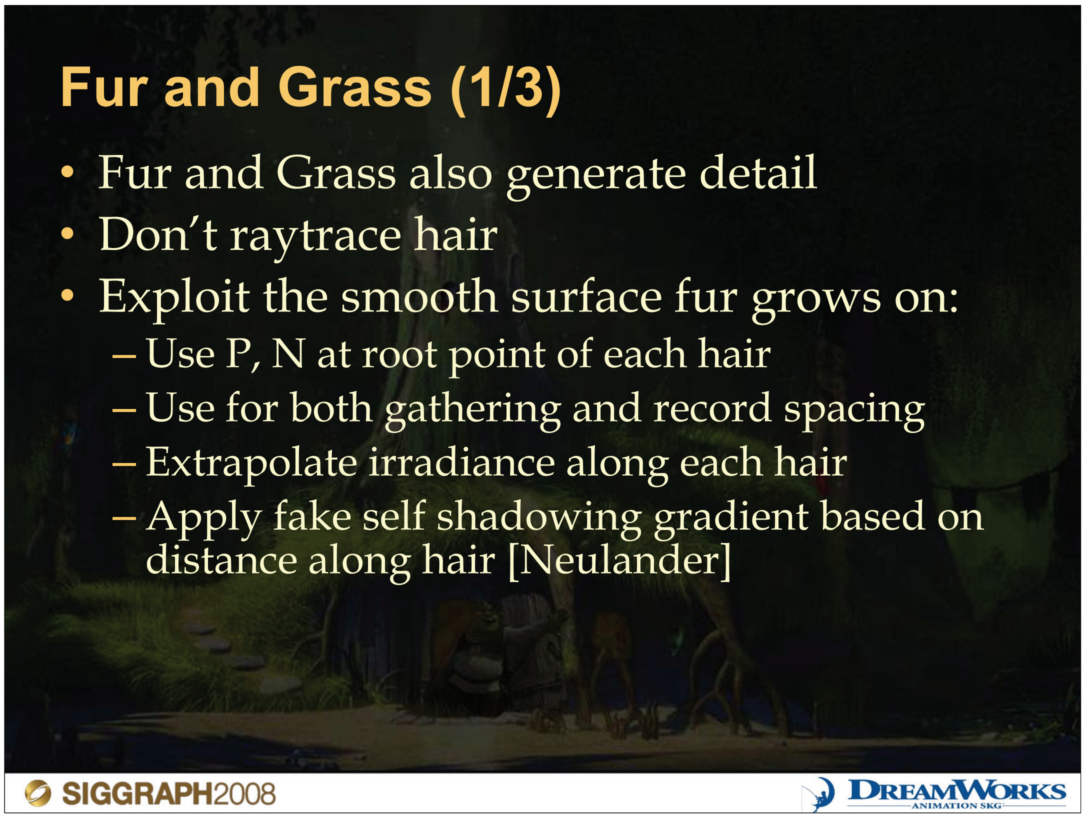
Displacement (4/4)

- Show video
- Results
  - Captures occlusion/bounce from surrounding objects onto small displaced detail
  - Misses self-occlusion from small displacements
    - Also from raytracing approximate geometry
    - Can be statically baked accurately

SIGGRAPH2008
DreamWorks Animation SKG

During the course we will show an animation with some results. As we can see in the images and video, we can achieve a sparse record spacing on displaced surfaces, and still capture occlusion / bounce from surrounding objects onto small displaced detail. When shrek's arm occludes its vest, it reveals small displaced surface detail.

We do however miss self-occlusion or self-bounce from small displacements with this technique. This is also partially the fact that we raytrace approximate geometry. In the case of ambient occlusion, this effect is mostly static and can be pre-computed accurately, stored in uv texture maps and easily combined to the final result.



Fur and Grass (1/3)

- Fur and Grass also generate detail
- Don't raytrace hair
- Exploit the smooth surface fur grows on:
  - Use P, N at root point of each hair
  - Use for both gathering and record spacing
  - Extrapolate irradiance along each hair
  - Apply fake self shadowing gradient based on distance along hair [Neulander]

SIGGRAPH2008 DREAMWORKS ANIMATION SKG

Characters and objects covered in fur and grassy ground planes are more and more common and pose challenging rendering problems. Raytracing hair is one of them, and in our system we typically avoid explicitly doing it: hair rarely directly contributes to occlusion or bouncing light onto itself or onto other surrounding surfaces and is sometimes replaced by lower resolution proxy geometry to achieve the desired effect.

However, we definitely want hair to receive bounce lighting or occlusion from neighboring surfaces. We therefore need to perform gathering calculations from hair fragments and we can accelerate this process using irradiance caching for medium-to-short hair and grass. Here again we can exploit the smoothness of the underlying surface that fur grows on (called the "growing surface" below):

•Cache records are stored on the growing surface,

•Cache lookups happen on the growing surface for each hair fragment at the root of the corresponding hair,

•Irradiance is sampled at the root point of hairs when cache misses happen,

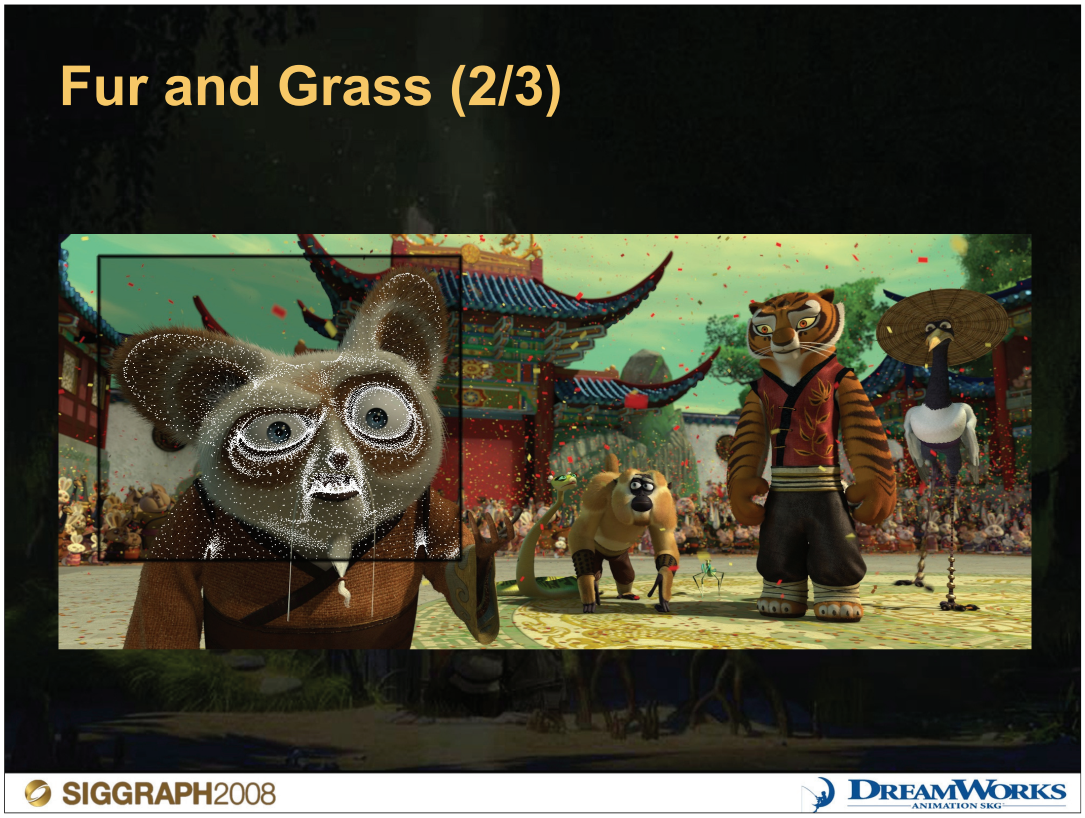
•Hair fragments don't define a surface themselves and don't provide a very meaningful normal. Therefore we don't apply any gradient when extrapolating the irradiance along each hair. However we can apply fake self-shadowing gradient based on the distance along the hair. There are many ways to do this and a good reference can be found here [Neulander].

For this to be possible, we simply store in our deep file the growing surface position and normal for each hair.

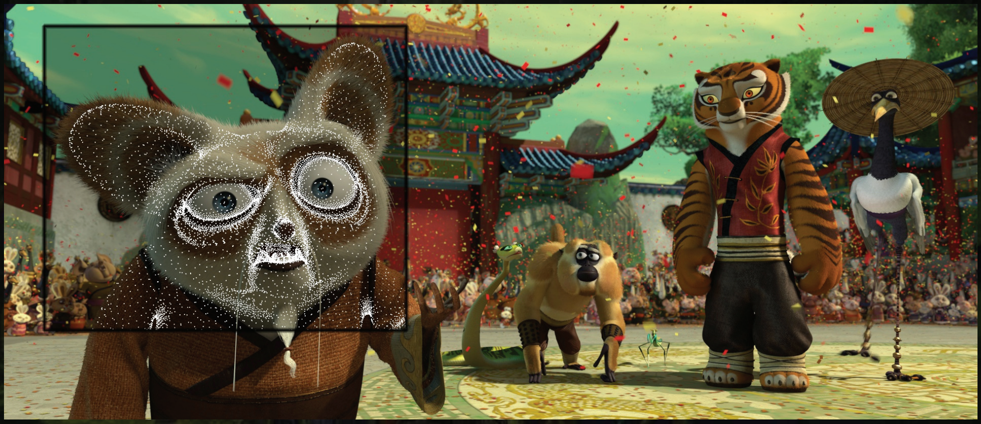
Fur and Grass (2/3)

SIGGRAPH2008 DreamWorks Animation SKG

This technique works quite well in practice for medium to short fur and captures occlusion / bounce from surrounding objects onto the hair. Unfortunately, there doesn't seem to be a generic coherence we can exploit similarly with longer hair. Also this technique doesn't capture fur self-occlusion although alternate techniques exist and are widely used for this purpose.

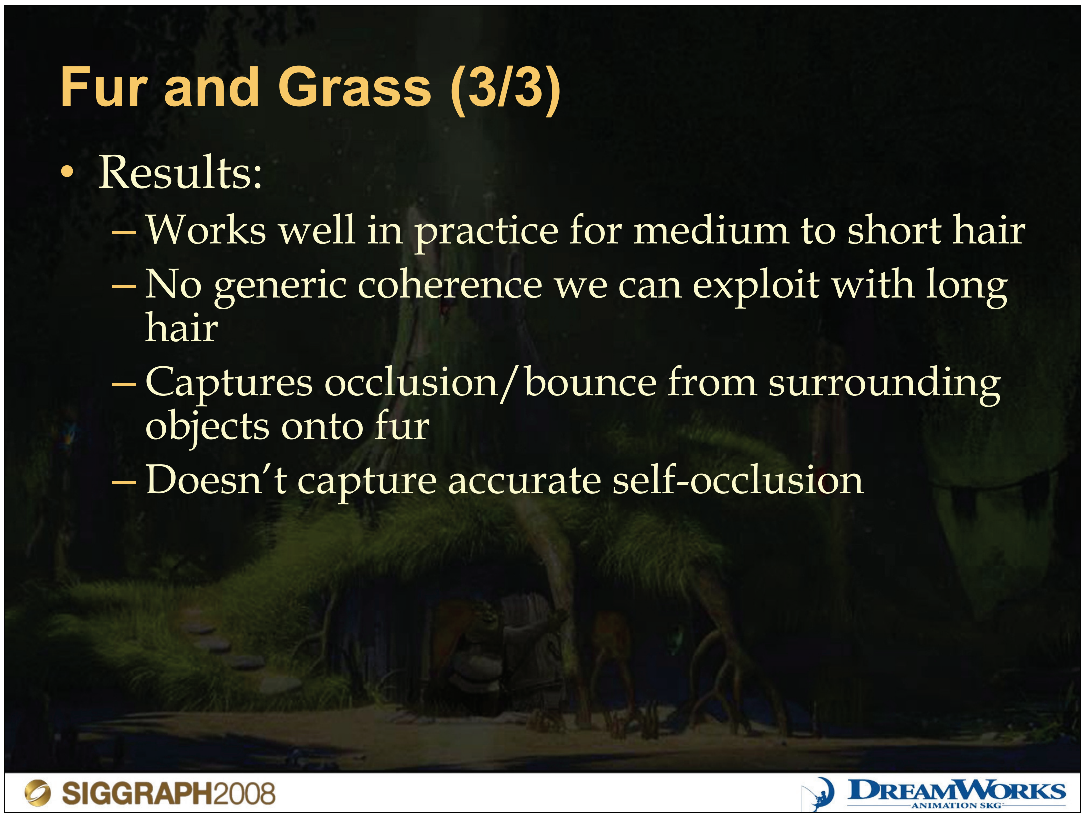


Fur and Grass (2/3)



SIGGRAPH2008 DREAMWORKS ANIMATION SKG

This technique works quite well in practice for medium to short fur and captures occlusion / bounce from surrounding objects onto the hair. Unfortunately, there doesn't seem to be a generic coherence we can exploit similarly with longer hair. Also this technique doesn't capture fur self-occlusion although alternate techniques exist and are widely used for this purpose.

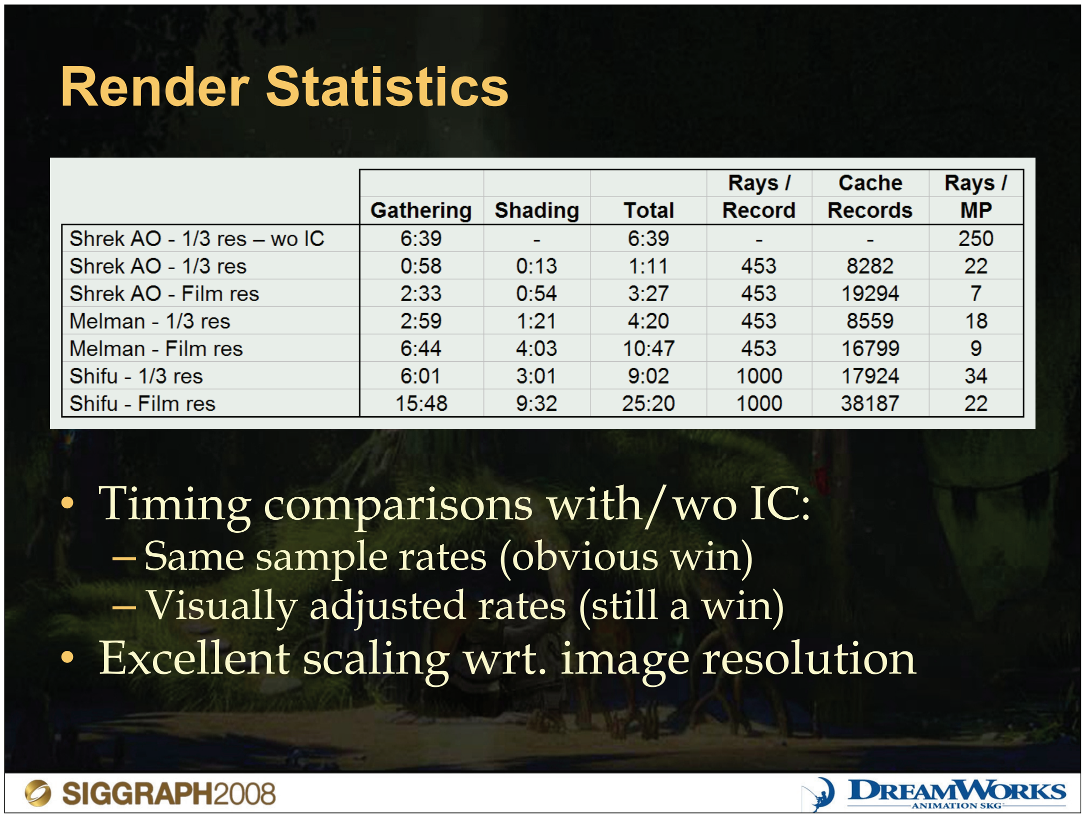


Fur and Grass (3/3)

- Results:
  - Works well in practice for medium to short hair
  - No generic coherence we can exploit with long hair
  - Captures occlusion/bounce from surrounding objects onto fur
  - Doesn't capture accurate self-occlusion

SIGGRAPH2008 DREAMWORKS ANIMATION SKG

This technique works quite well in practice for medium to short fur and captures occlusion / bounce from surrounding objects onto the hair. Unfortunately, there doesn't seem to be a generic coherence we can exploit similarly with longer hair. Also this technique doesn't capture fur self-occlusion although alternate techniques exist and are widely used for this purpose.



Render Statistics

	Gathering	Shading	Total	Rays / Record	Cache Records	Rays / MP
Shrek AO - 1/3 res – wo IC	6:39	-	6:39	-	-	250
Shrek AO - 1/3 res	0:58	0:13	1:11	453	8282	22
Shrek AO - Film res	2:33	0:54	3:27	453	19294	7
Melman - 1/3 res	2:59	1:21	4:20	453	8559	18
Melman - Film res	6:44	4:03	10:47	453	16799	9
Shifu - 1/3 res	6:01	3:01	9:02	1000	17924	34
Shifu - Film res	15:48	9:32	25:20	1000	38187	22

- Timing comparisons with/wo IC:
  - Same sample rates (obvious win)
  - Visually adjusted rates (still a win)
- Excellent scaling wrt. image resolution

SIGGRAPH2008 DREAMWORKS ANIMATION SKG

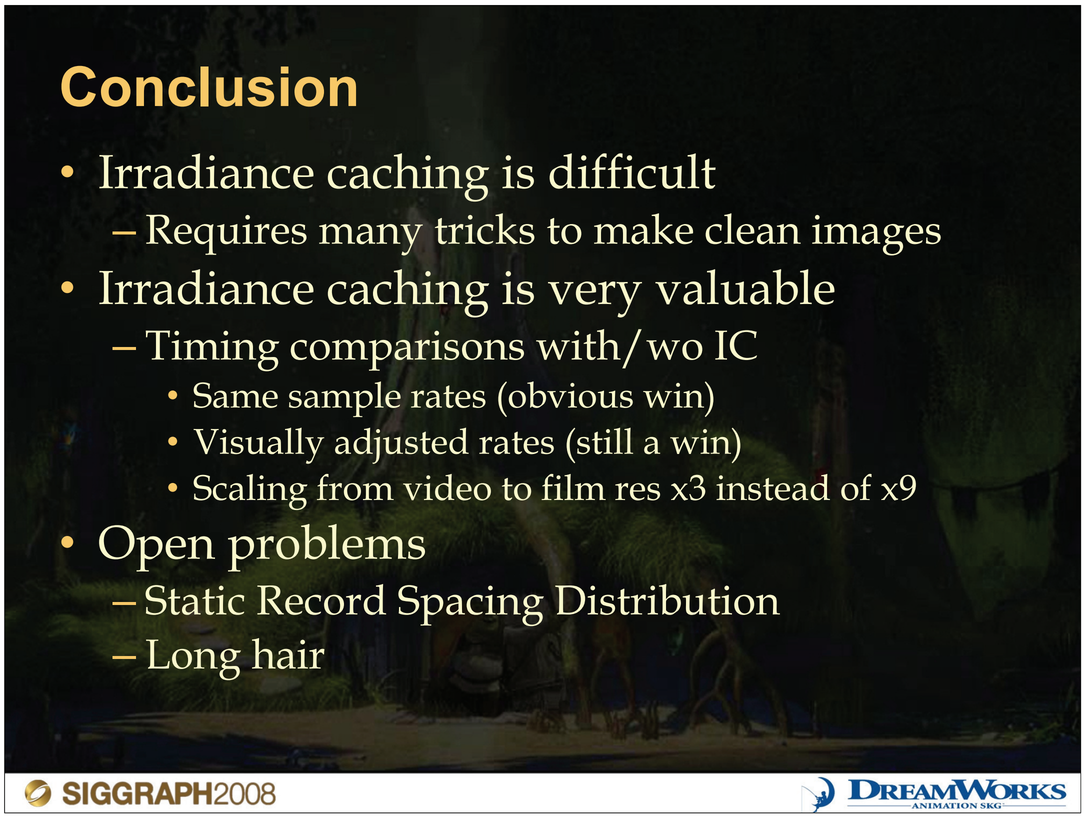
Here are some render times for the various image examples shown. They were all achieved on a AMD Opteron 2.14 GHz workstation running on a single core.

We also provide a comparison with/without using irradiance caching. In this comparison, we visually adjusted the number of rays per pixel to achieve a similarly noise-free image than when using irradiance caching.

Video

SIGGRAPH2008 DreamWorks Animation SKG

SHOW VIDEO KFP SQ ?



Conclusion

- Irradiance caching is difficult
  - Requires many tricks to make clean images
- Irradiance caching is very valuable
  - Timing comparisons with/wo IC
    - Same sample rates (obvious win)
    - Visually adjusted rates (still a win)
    - Scaling from video to film res x3 instead of x9
- Open problems
  - Static Record Spacing Distribution
  - Long hair

SIGGRAPH2008 DREAMWORKS ANIMATION SKG

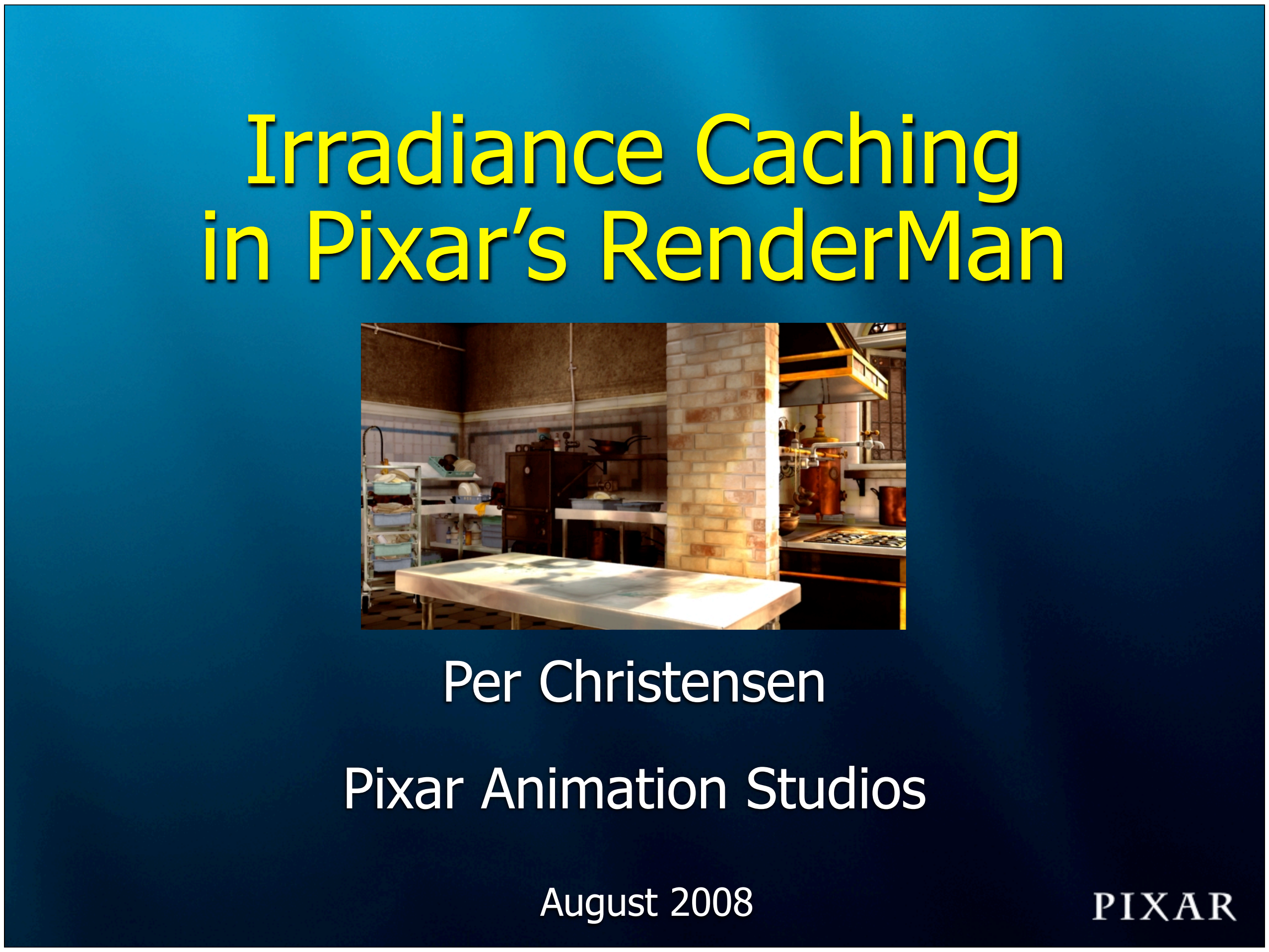
As we have seen throughout this course, irradiance caching is not an easy topic and requires many tricks to work well and produce clean images. It is however a very valuable technique, as it provides many benefits in the context of an interactive lighting tool and pipeline. It also accelerates gathering calculations significantly as can be shown in our render time results above.

Direction for future work includes exploring static record spacing distributions on animated deforming characters, as well as finding ways to extract some form of coherence from long hair.

Acknowledgements

- Course Organisers
- Janne Kontkanen
- Jin Liou
- Mark Edwards
- Everyone at DreamWorks

SIGGRAPH2008
DREAMWORKS
ANIMATION SKG



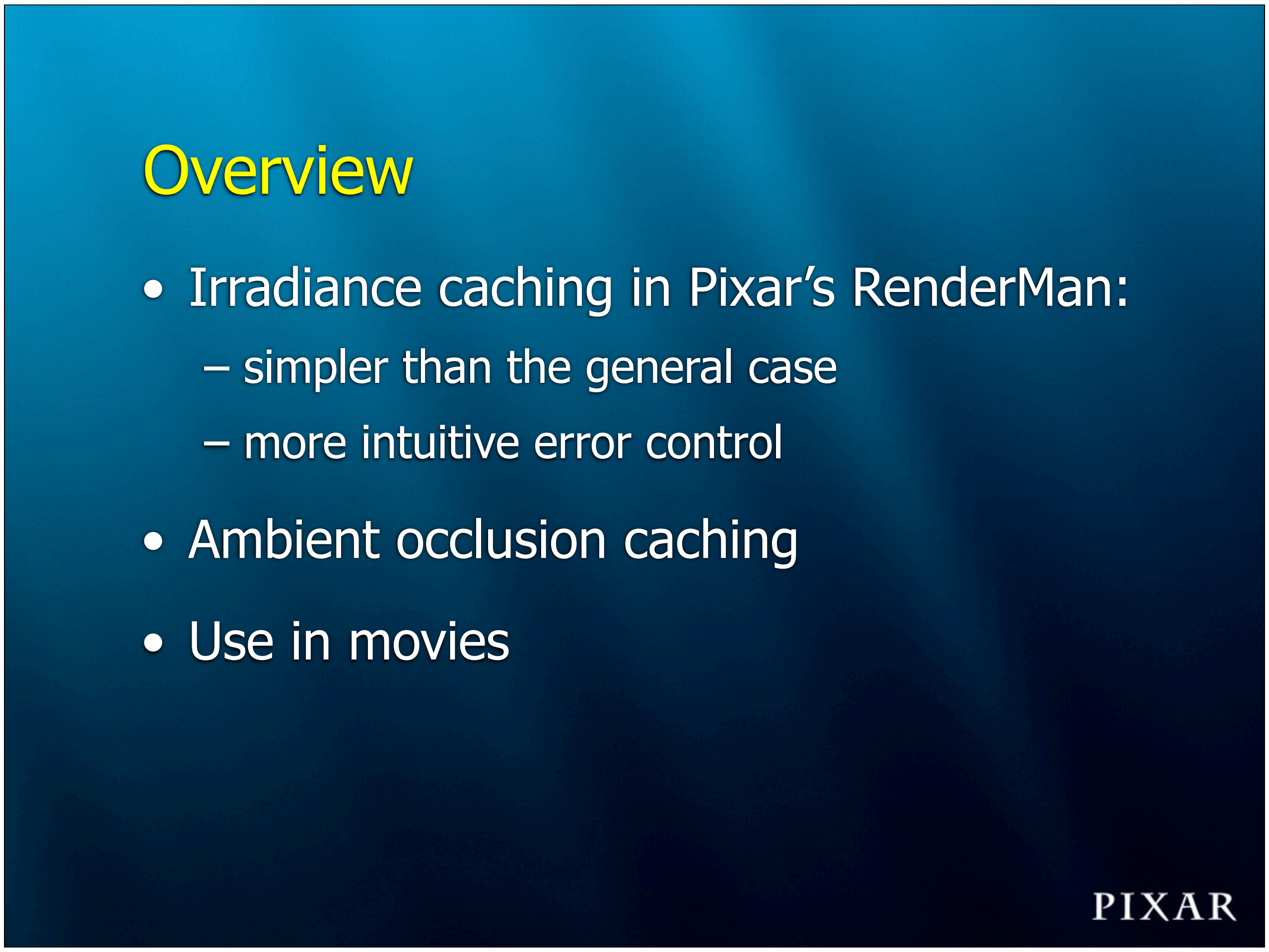
Irradiance Caching in Pixar's RenderMan

Per Christensen

Pixar Animation Studios

August 2008

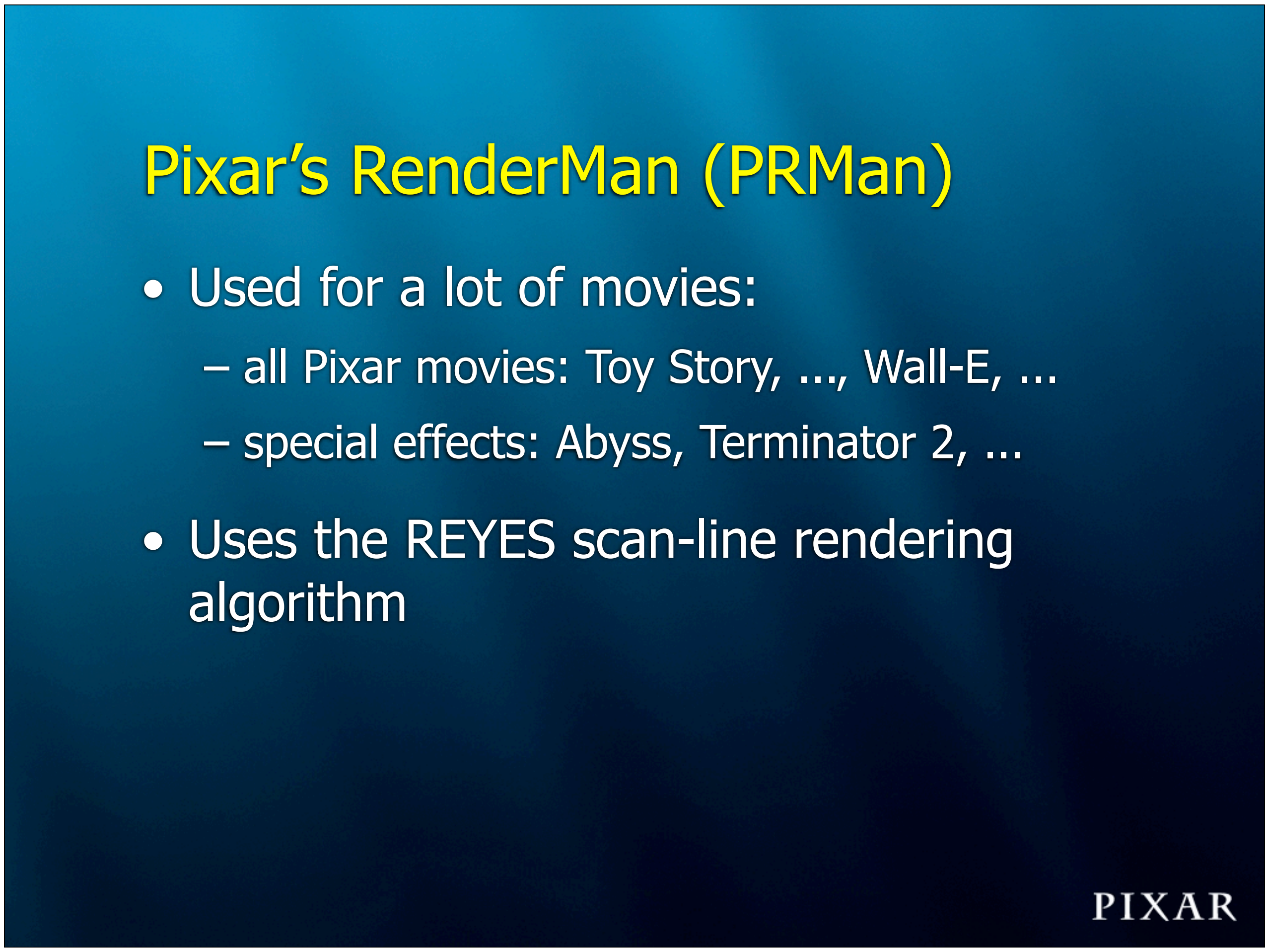
PIXAR



Overview

- Irradiance caching in Pixar's RenderMan:
  - simpler than the general case
  - more intuitive error control
- Ambient occlusion caching
- Use in movies

PIXAR



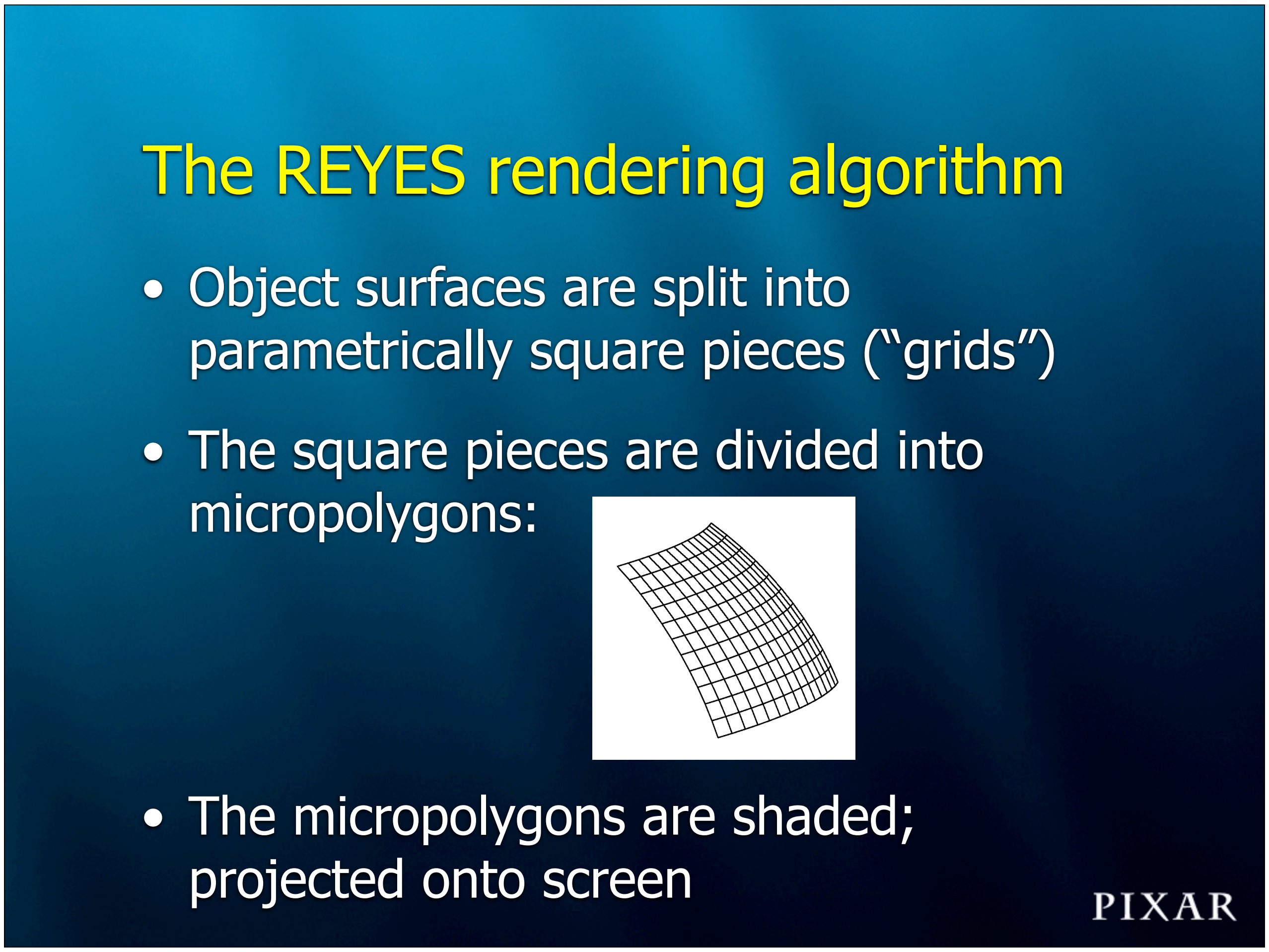
Pixar's RenderMan (PRMan)

- Used for a lot of movies:
  - all Pixar movies: Toy Story, ..., Wall-E, ...
  - special effects: Abyss, Terminator 2, ...
- Uses the REYES scan-line rendering algorithm

PIXAR

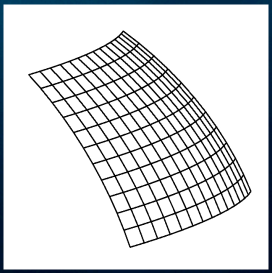
PRMan was also used for Star Wars episodes 1-3, Lord of the Rings 1-3, the Harry Potter movies, etc.
Every Visual Effects Oscar Winner of the past 15 years used PRMan.

A list of movies produced with PRMan can be found at https://renderman.pixar.com/products/whatsrenderman/movies.html



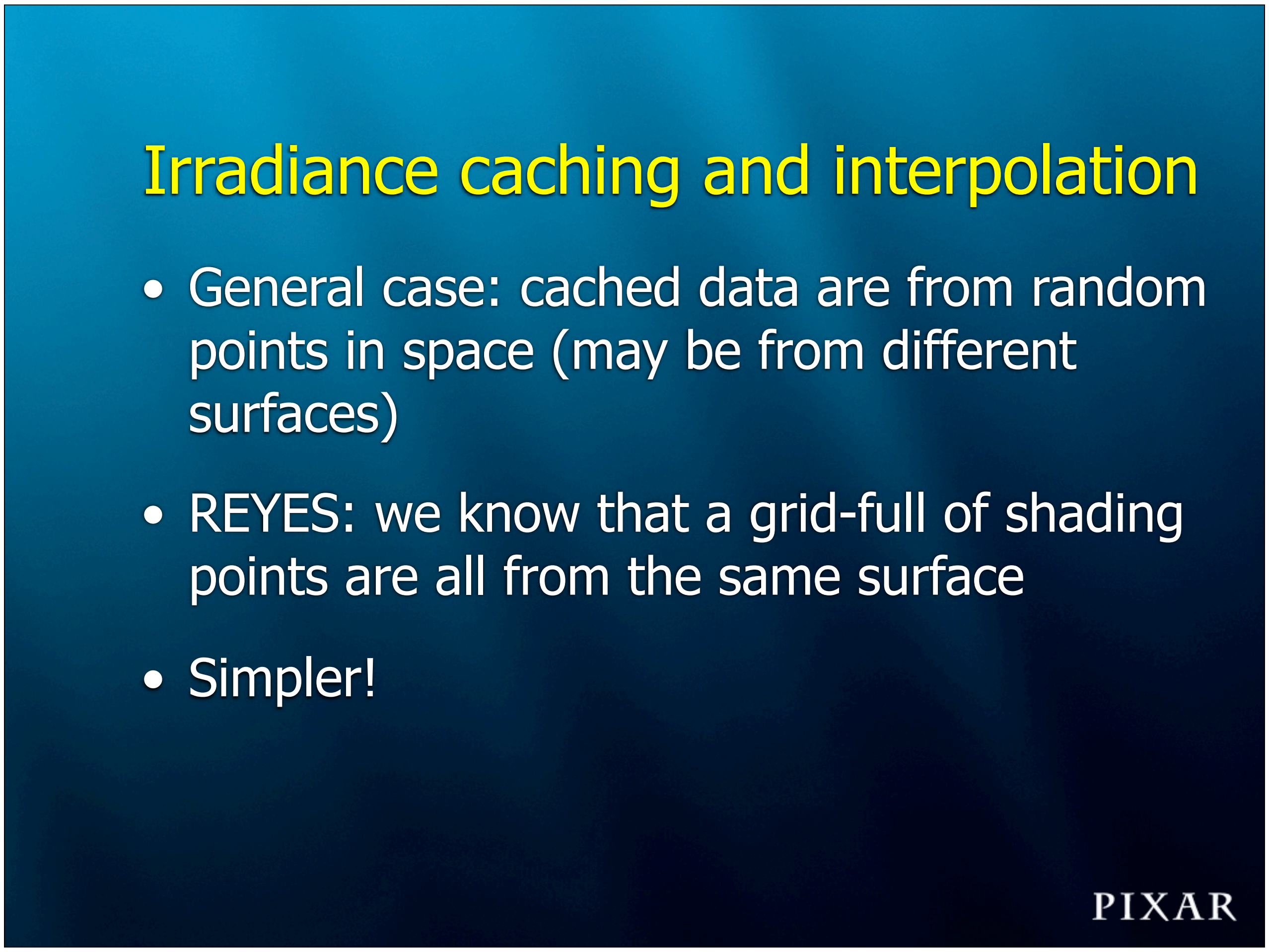
The REYES rendering algorithm

- Object surfaces are split into parametrically square pieces ("grids")
- The square pieces are divided into micropolygons:



- The micropolygons are shaded; projected onto screen

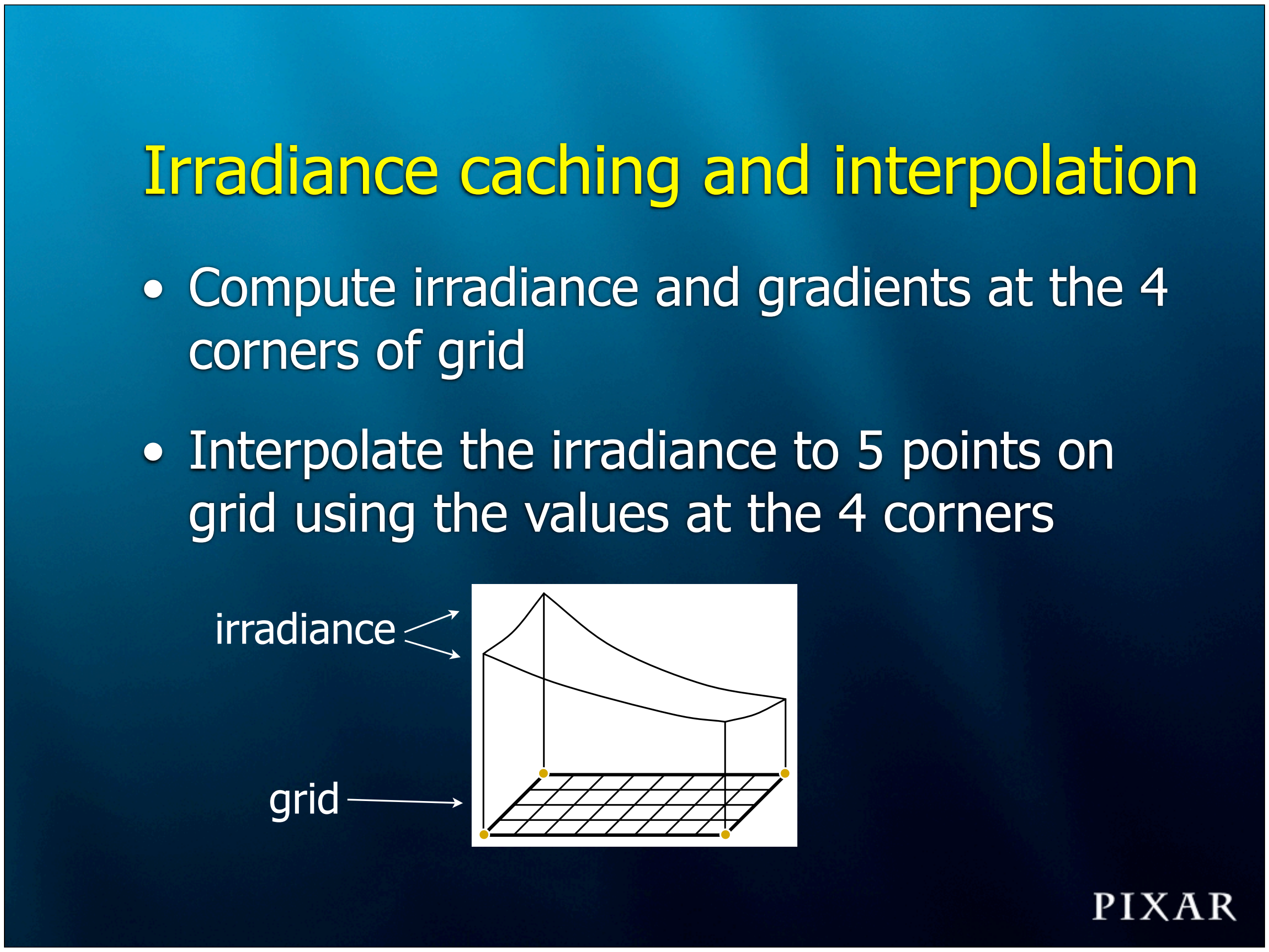
PIXAR



Irradiance caching and interpolation

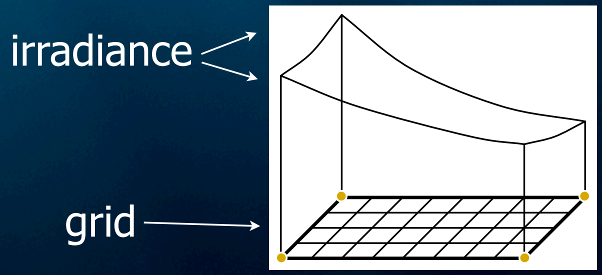
- General case: cached data are from random points in space (may be from different surfaces)
- REYES: we know that a grid-full of shading points are all from the same surface
- Simpler!

PIXAR

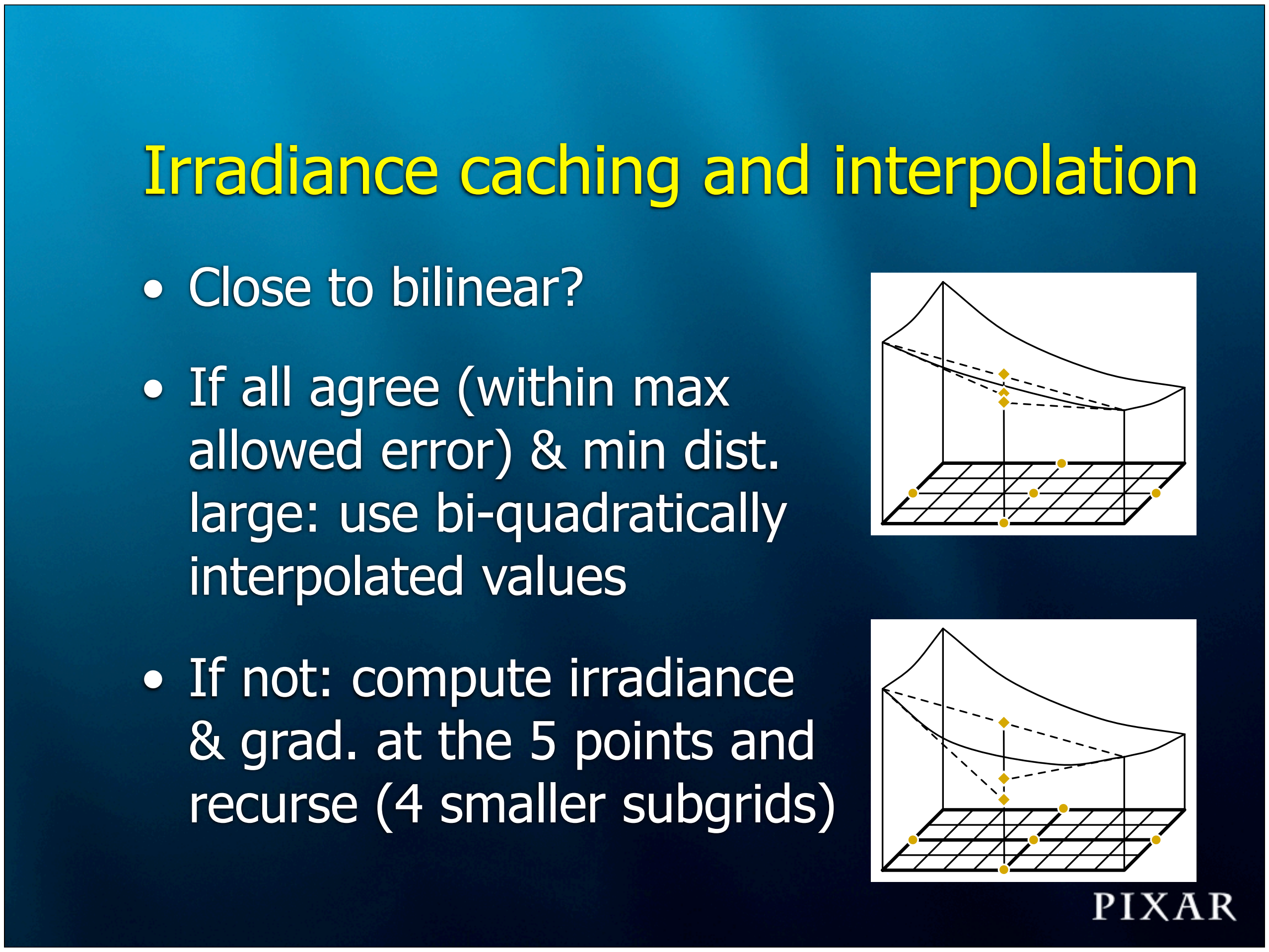


Irradiance caching and interpolation

- Compute irradiance and gradients at the 4 corners of grid
- Interpolate the irradiance to 5 points on grid using the values at the 4 corners

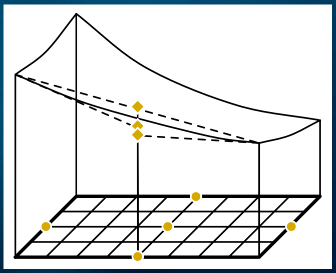


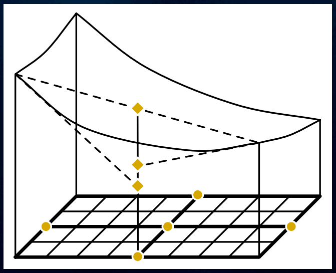
PIXAR



Irradiance caching and interpolation

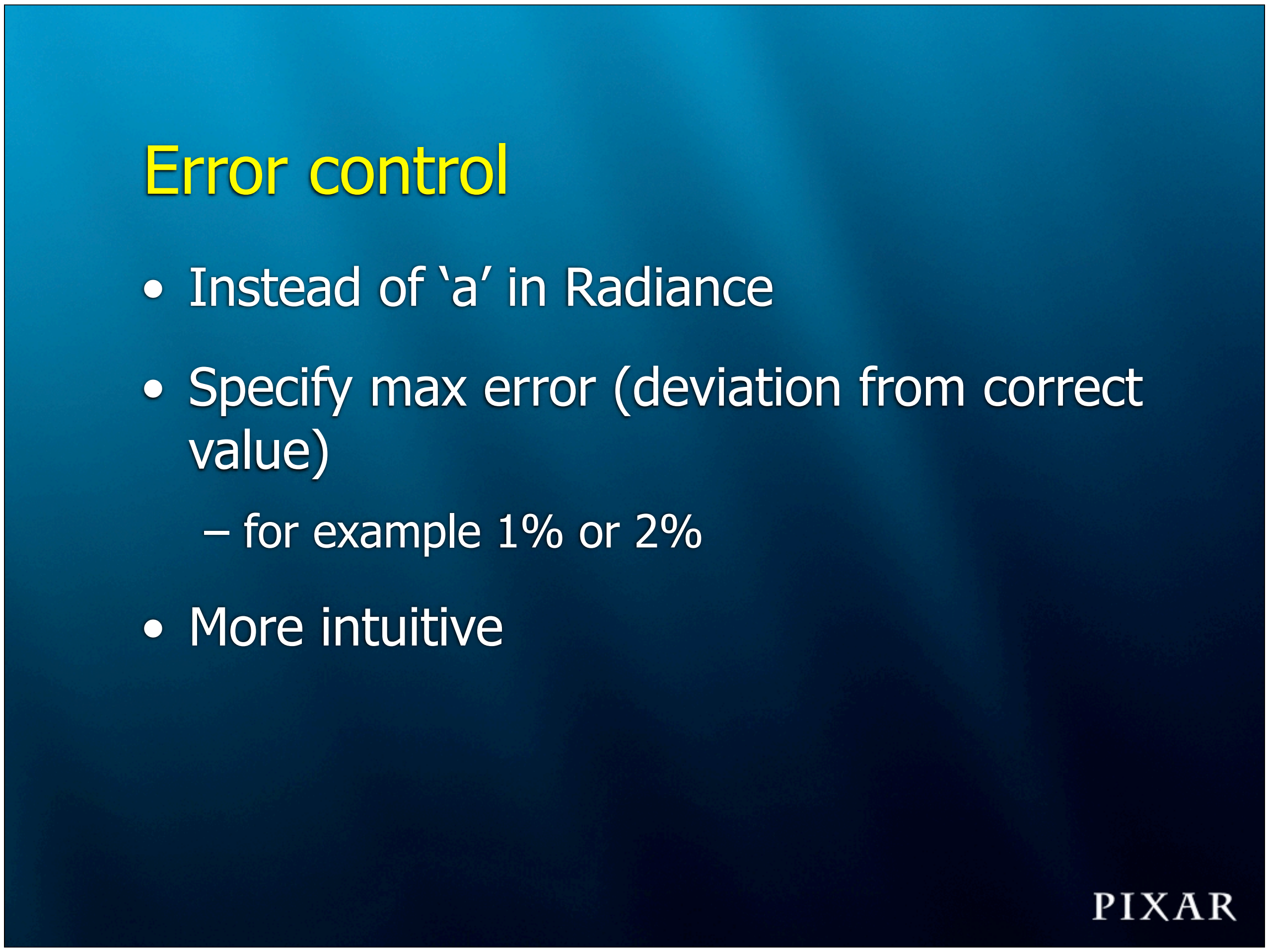
- Close to bilinear?
- If all agree (within max allowed error) & min dist. large: use bi-quadratically interpolated values
- If not: compute irradiance & grad. at the 5 points and recurse (4 smaller subgrids)





PIXAR

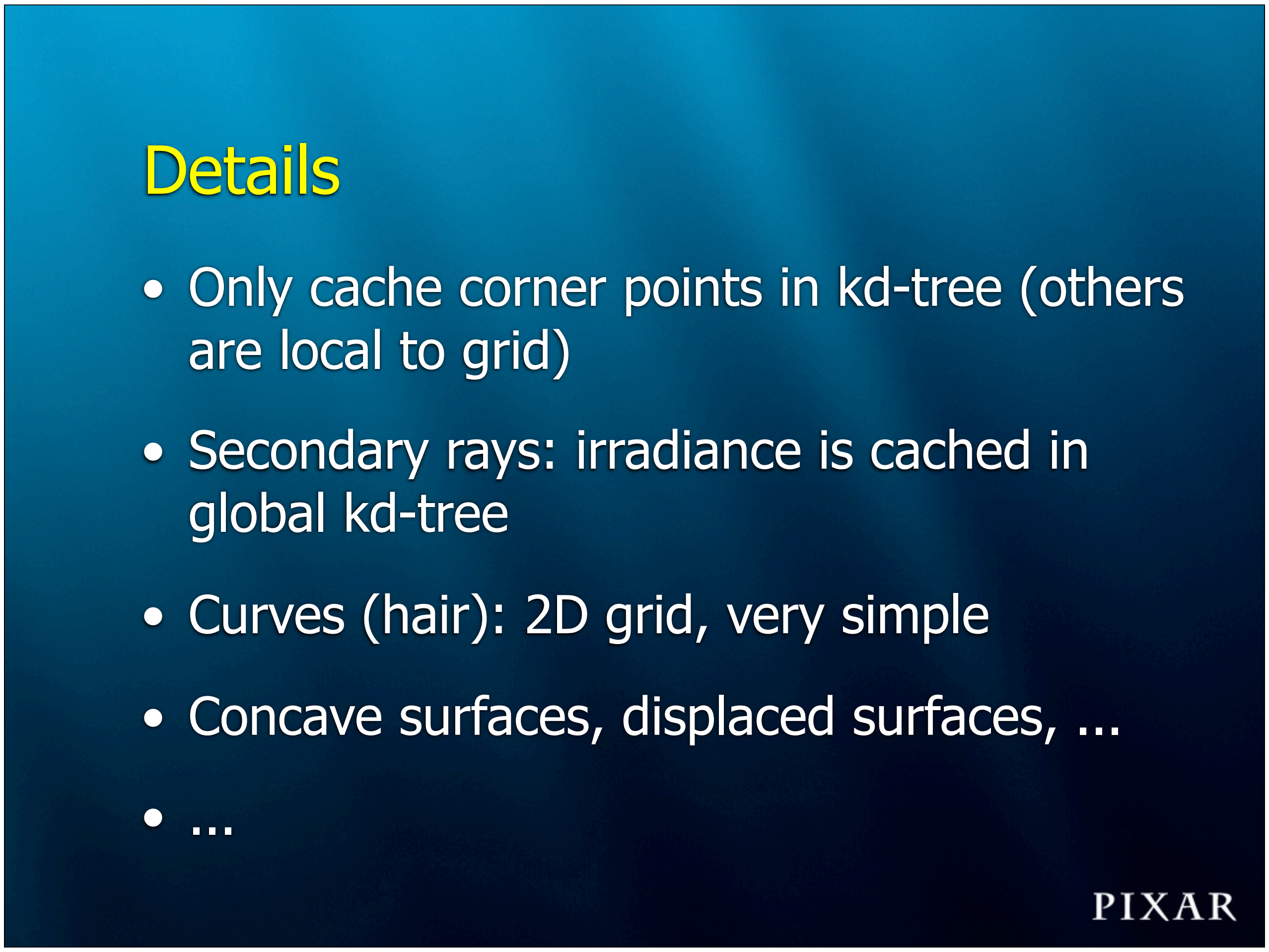
We assume that the irradiance on the patch has a bilinear variation. If not, we split the patch.



Error control

- Instead of 'a' in Radiance
- Specify max error (deviation from correct value)
  - for example 1% or 2%
- More intuitive

PIXAR

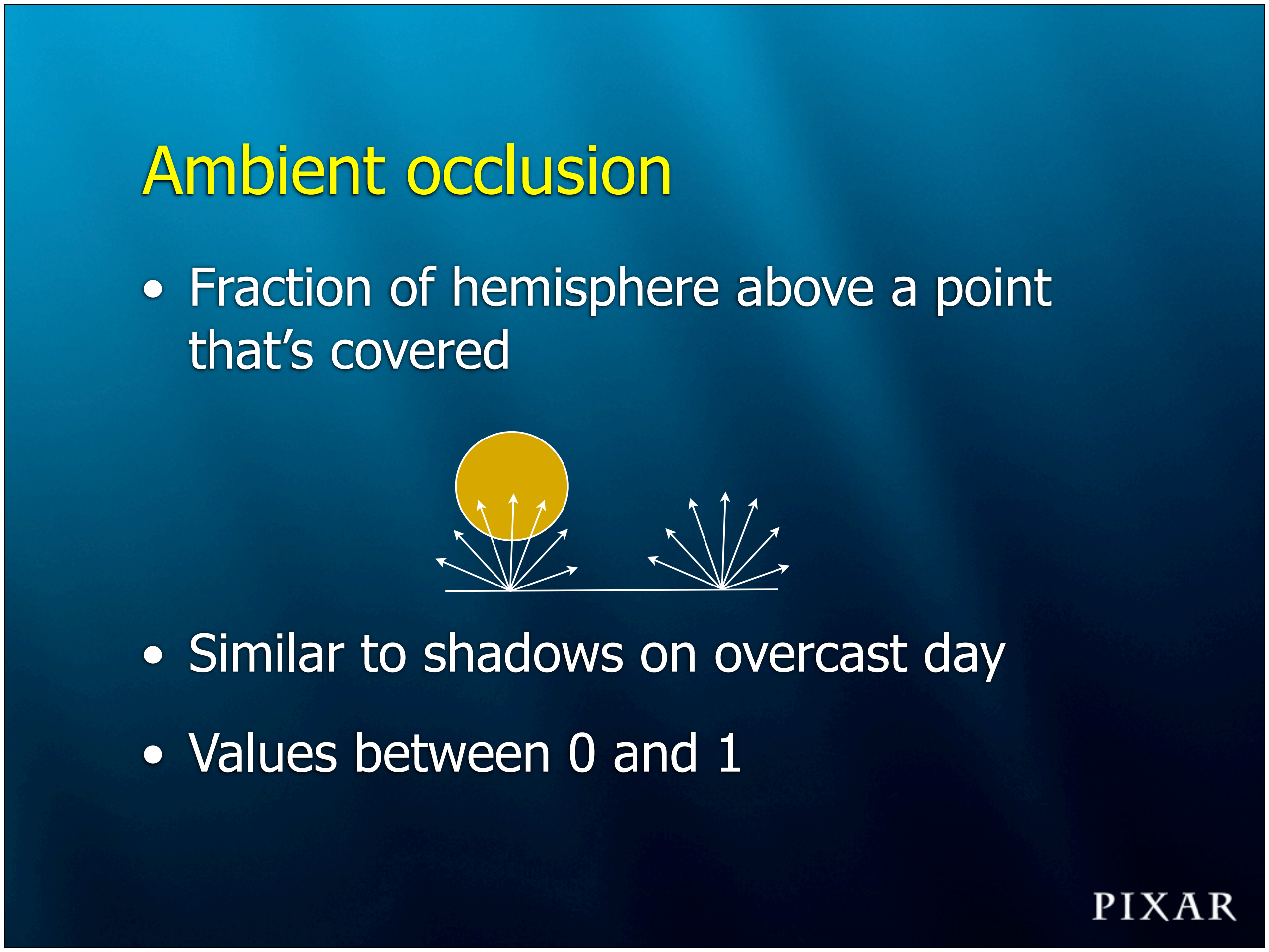


Details

- Only cache corner points in kd-tree (others are local to grid)
- Secondary rays: irradiance is cached in global kd-tree
- Curves (hair): 2D grid, very simple
- Concave surfaces, displaced surfaces, ...
- ...

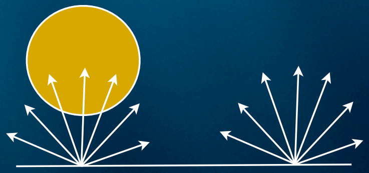
PIXAR

How to deal with concave and displaced surfaces has already been covered by the previous speaker



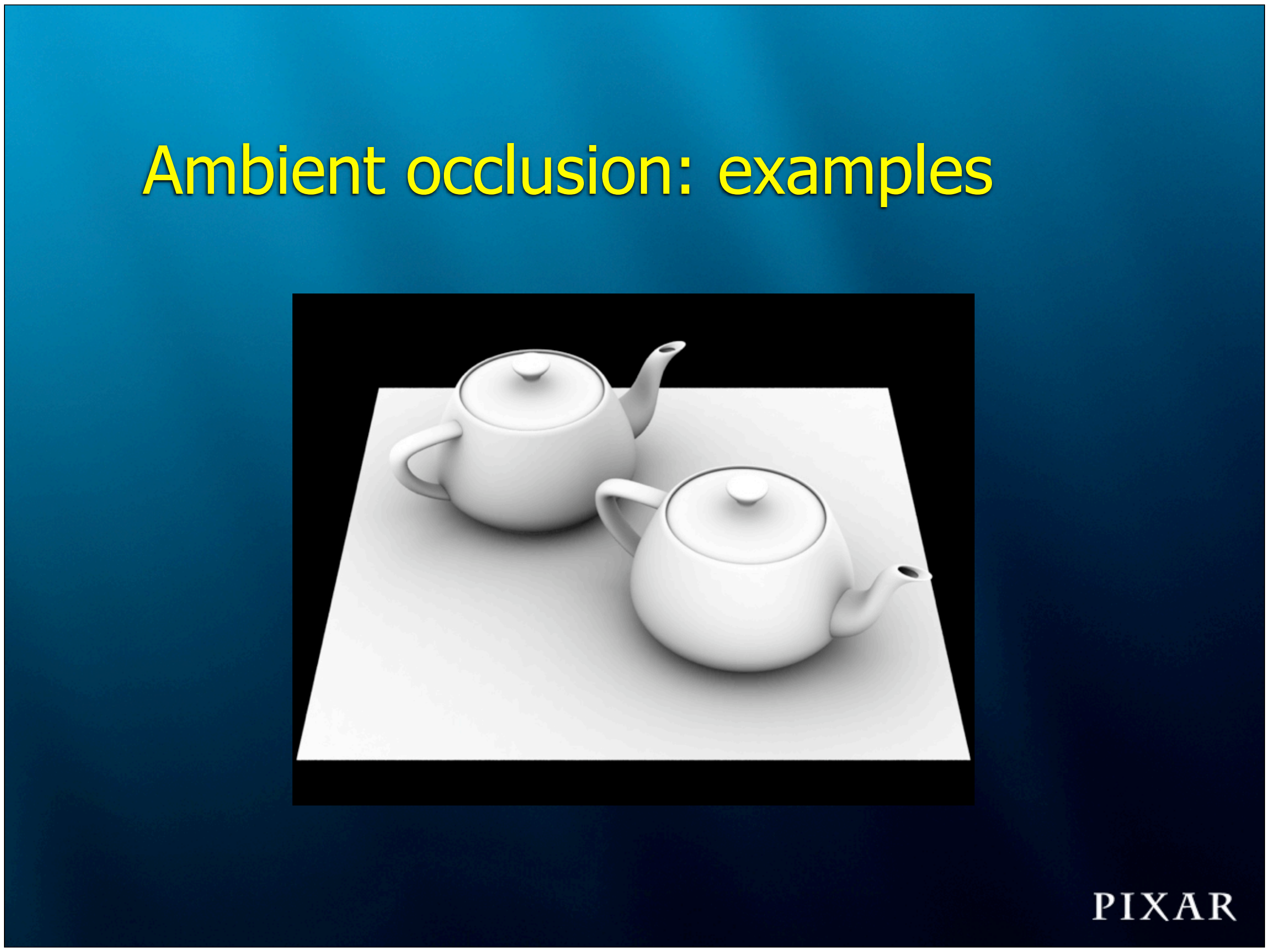
Ambient occlusion

- Fraction of hemisphere above a point that's covered

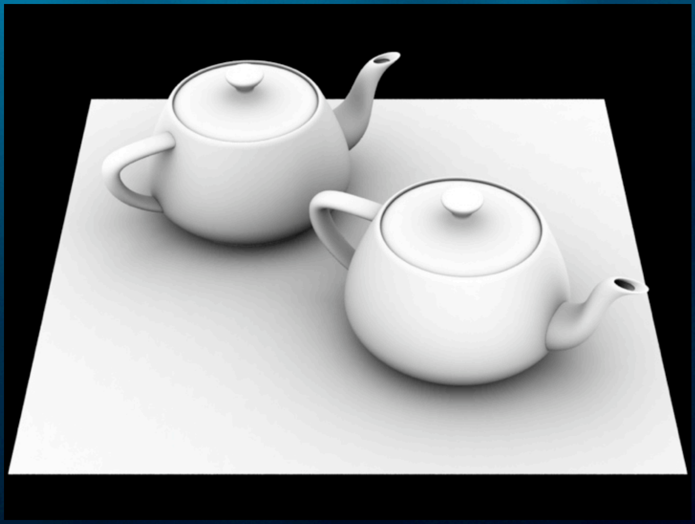


- Similar to shadows on overcast day
- Values between 0 and 1

PIXAR



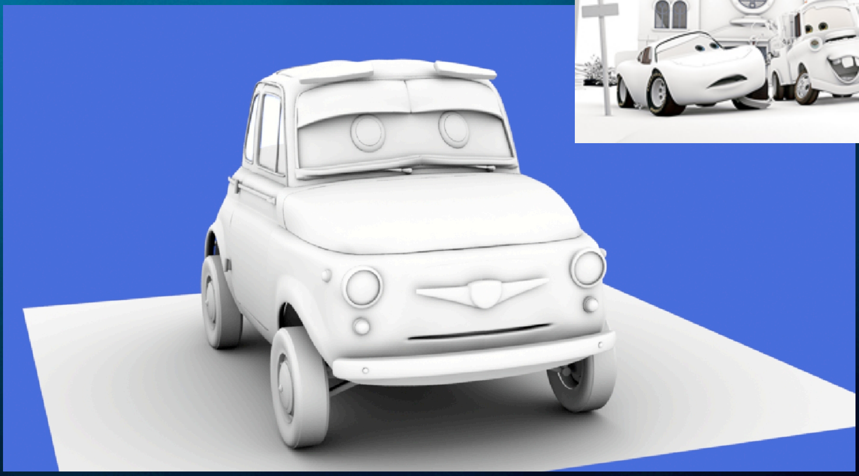
Ambient occlusion: examples



PIXAR

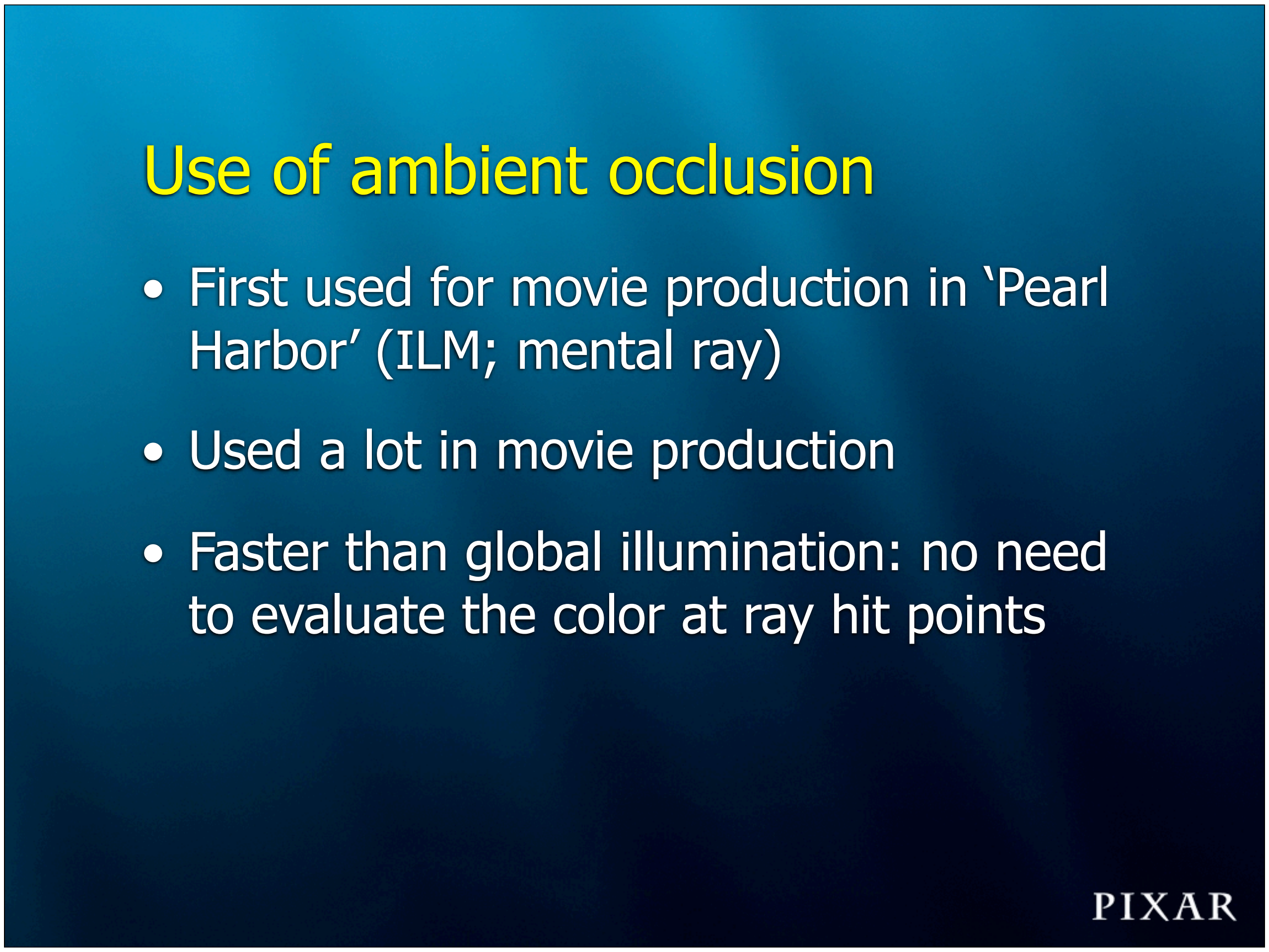
Ambient occlusion: examples

... more cars



Luigi from `Cars'

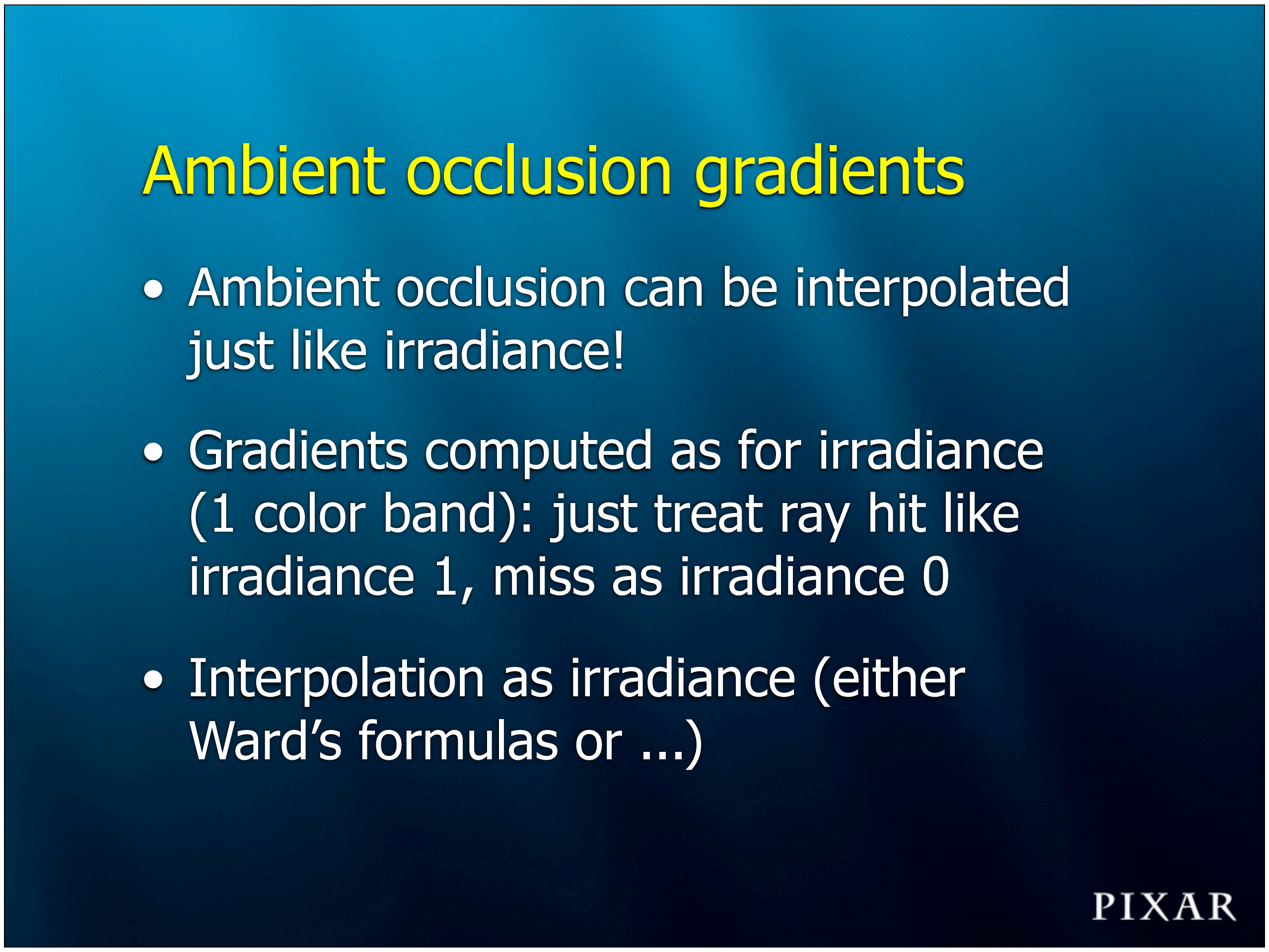
PIXAR



Use of ambient occlusion

- First used for movie production in ‘Pearl Harbor’ (ILM; mental ray)
- Used a lot in movie production
- Faster than global illumination: no need to evaluate the color at ray hit points

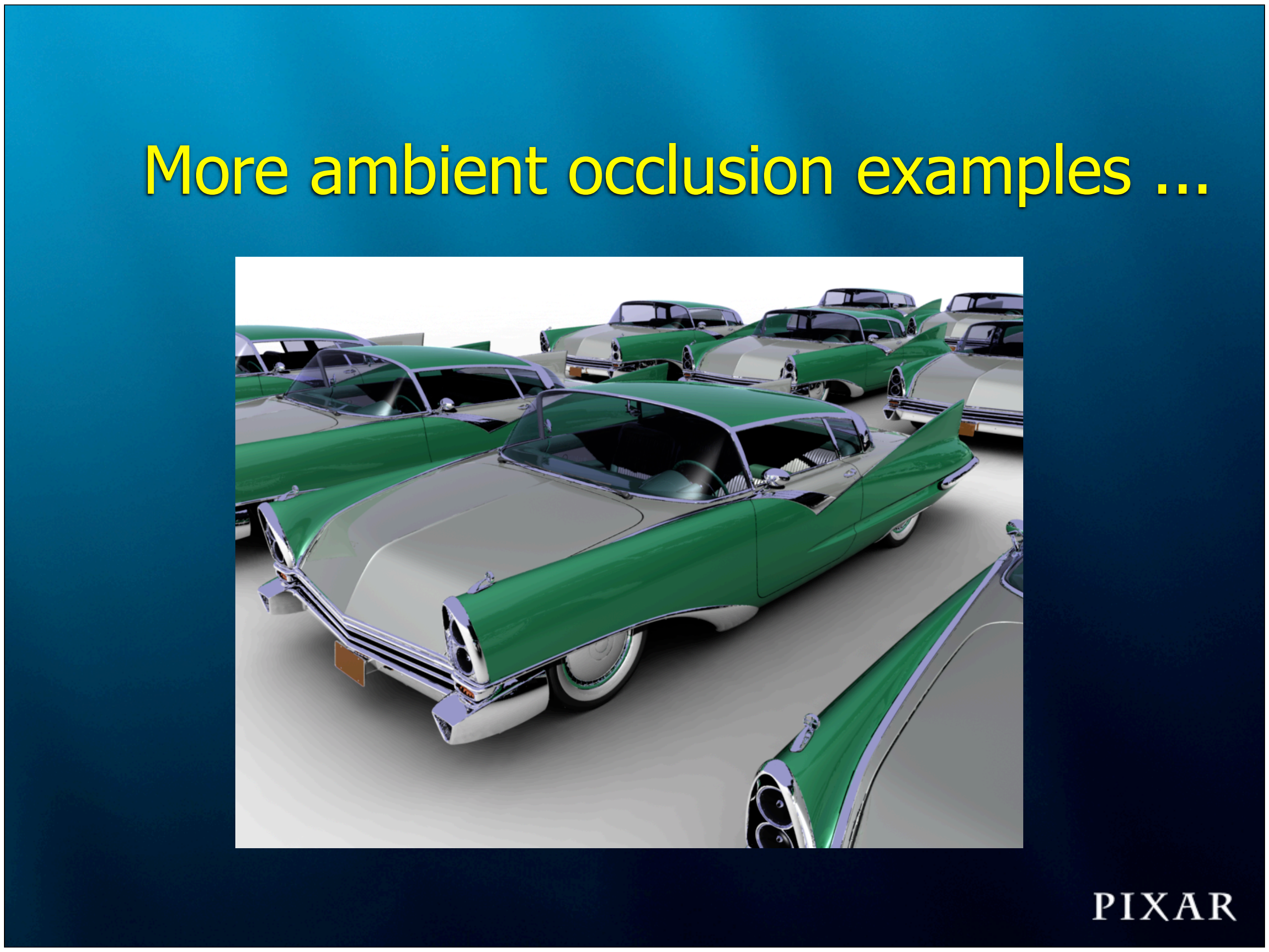
PIXAR



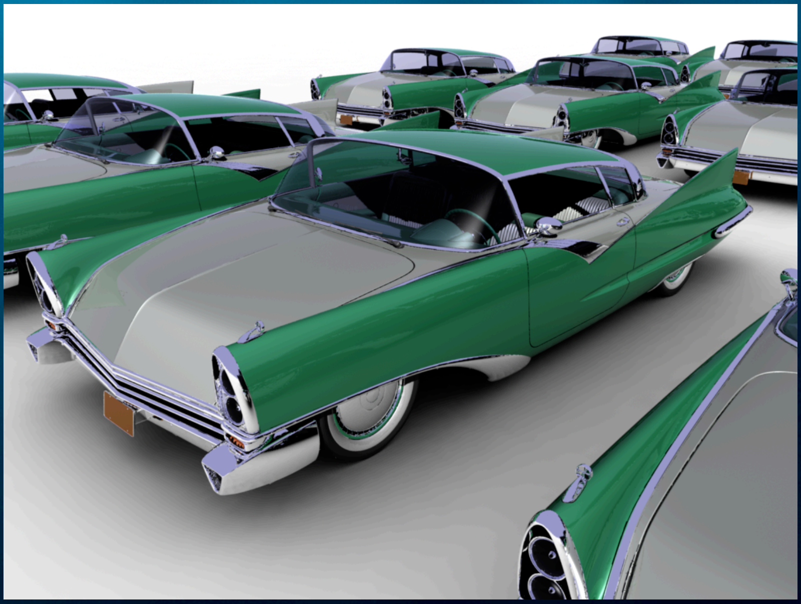
Ambient occlusion gradients

- Ambient occlusion can be interpolated just like irradiance!
- Gradients computed as for irradiance (1 color band): just treat ray hit like irradiance 1, miss as irradiance 0
- Interpolation as irradiance (either Ward's formulas or ...)

PIXAR

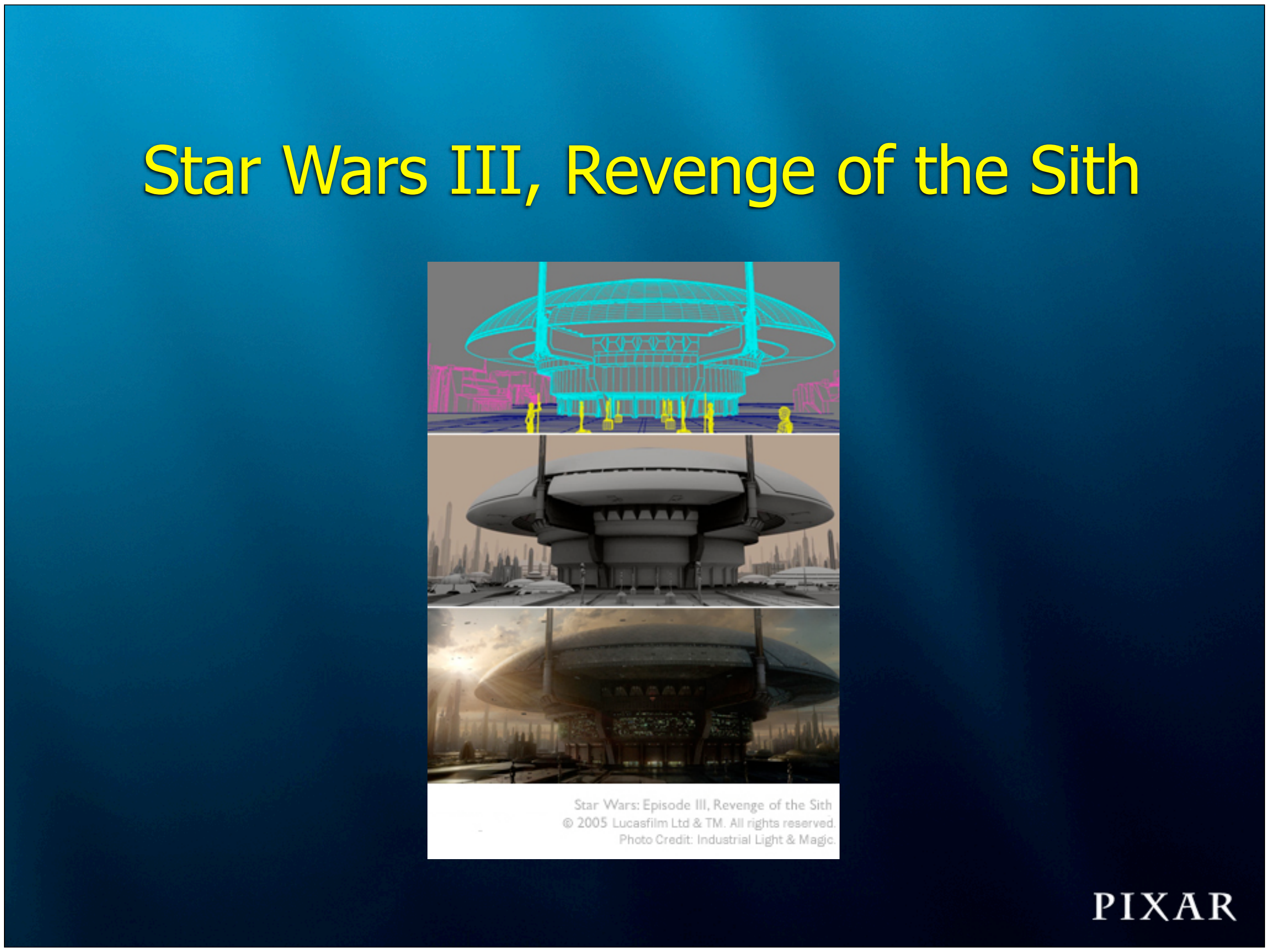


More ambient occlusion examples ...

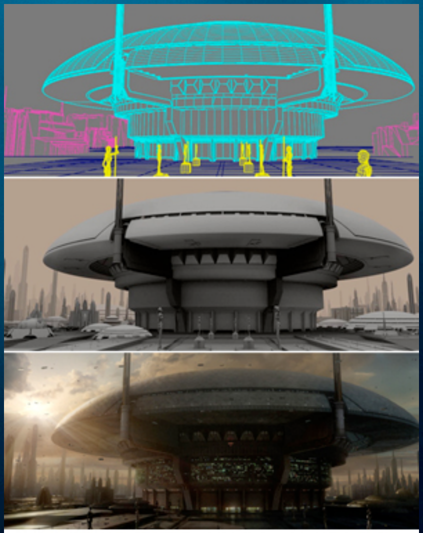


PIXAR

This is 15 copies of Al's car from Toy Story 2. The floor has ambient occlusion computed using irradiance gradients.



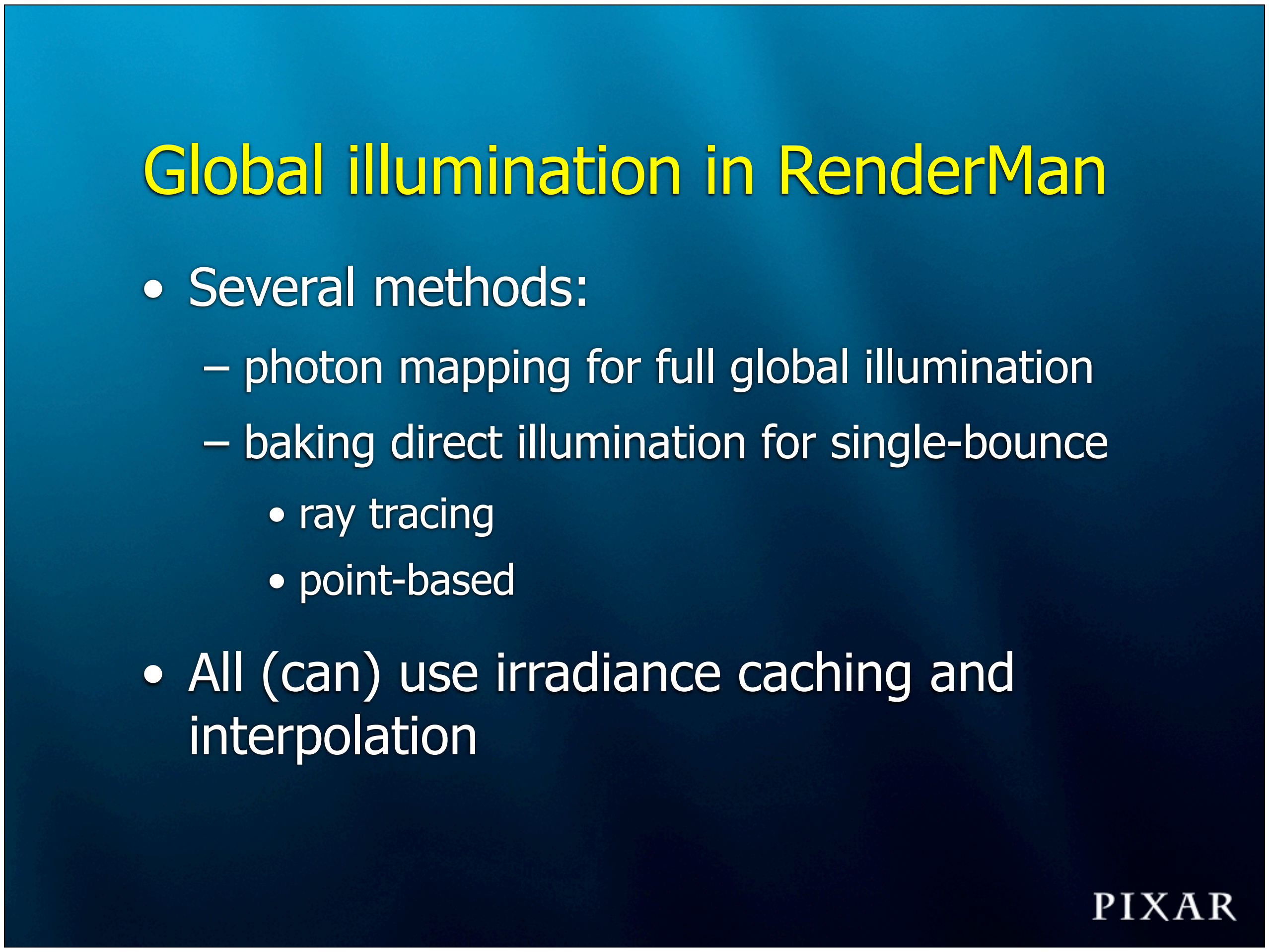
Star Wars III, Revenge of the Sith



Star Wars: Episode III, Revenge of the Sith
© 2005 Lucasfilm Ltd & TM. All rights reserved.
Photo Credit: Industrial Light & Magic.

PIXAR

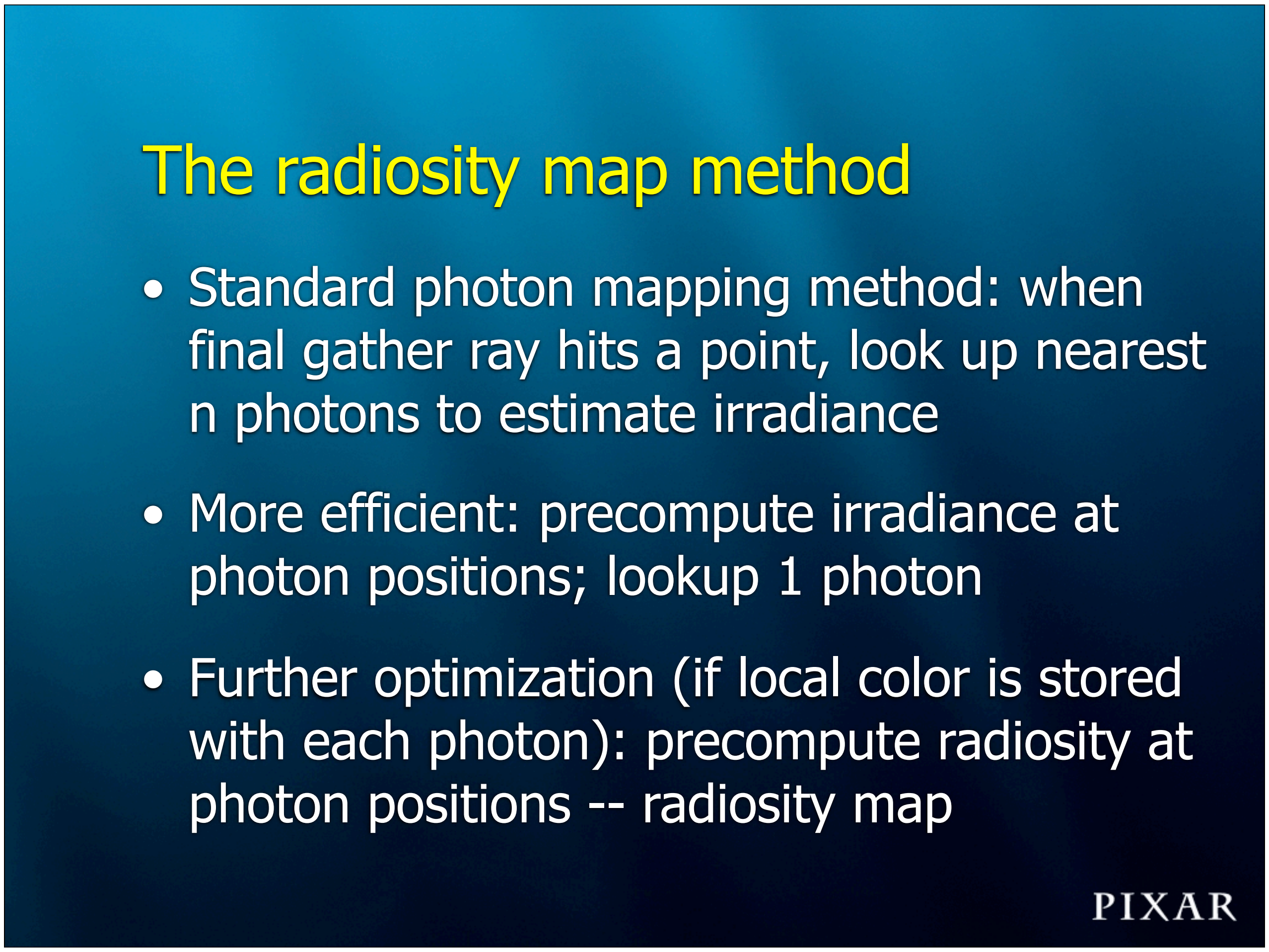
More ambient occlusion ....



Global illumination in RenderMan

- Several methods:
  - photon mapping for full global illumination
  - baking direct illumination for single-bounce
    - ray tracing
    - point-based
- All (can) use irradiance caching and interpolation

PIXAR

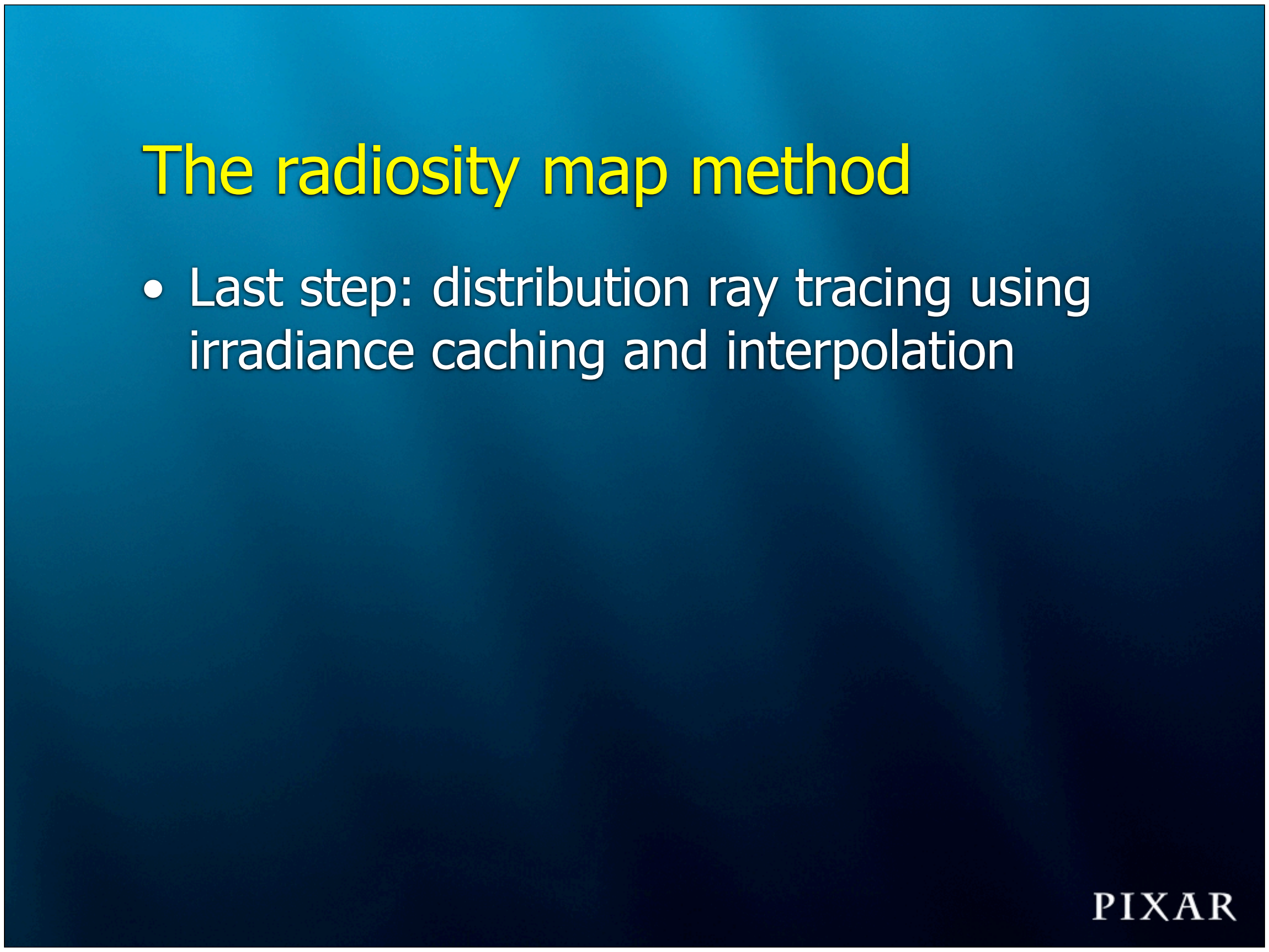


The radiosity map method

- Standard photon mapping method: when final gather ray hits a point, look up nearest n photons to estimate irradiance
- More efficient: precompute irradiance at photon positions; lookup 1 photon
- Further optimization (if local color is stored with each photon): precompute radiosity at photon positions -- radiosity map

PIXAR

More information about the radiosity map method can be found in the SIGGRAPH 2007 Course Notes “High-quality rendering using ray tracing and photon mapping” (Henrik Wann Jensen and Per Christensen).

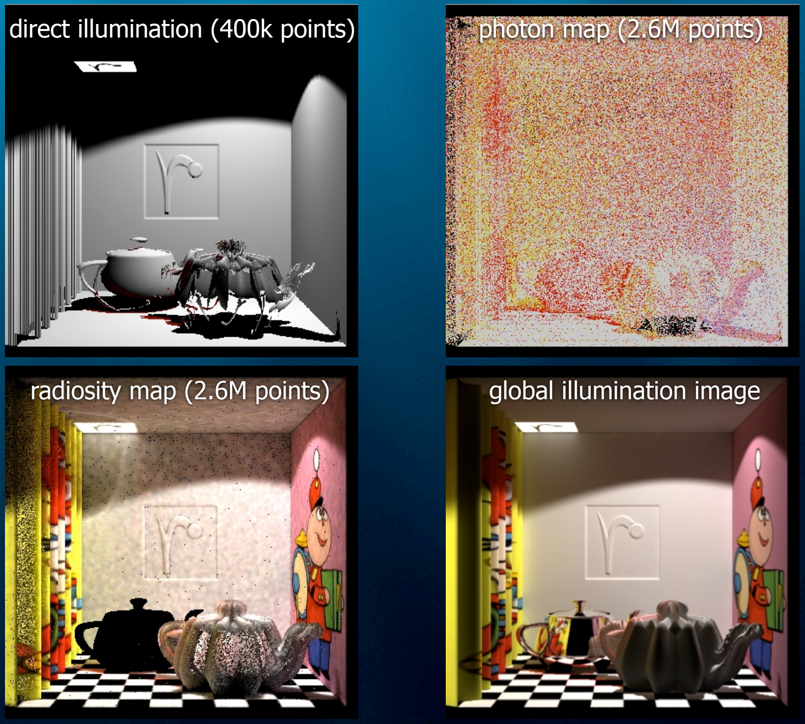


The radiosity map method

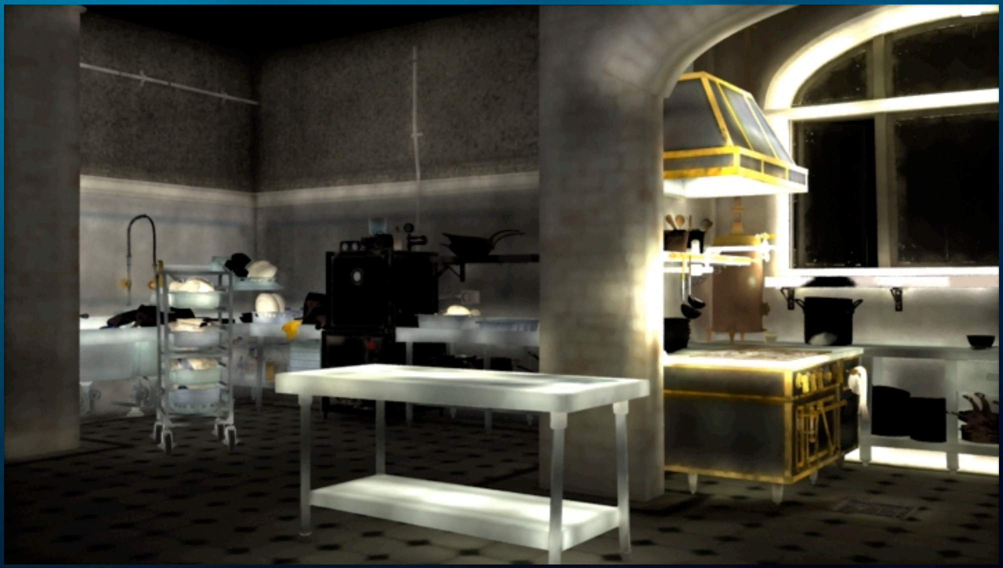
- Last step: distribution ray tracing using irradiance caching and interpolation

PIXAR

Example: box with teapots

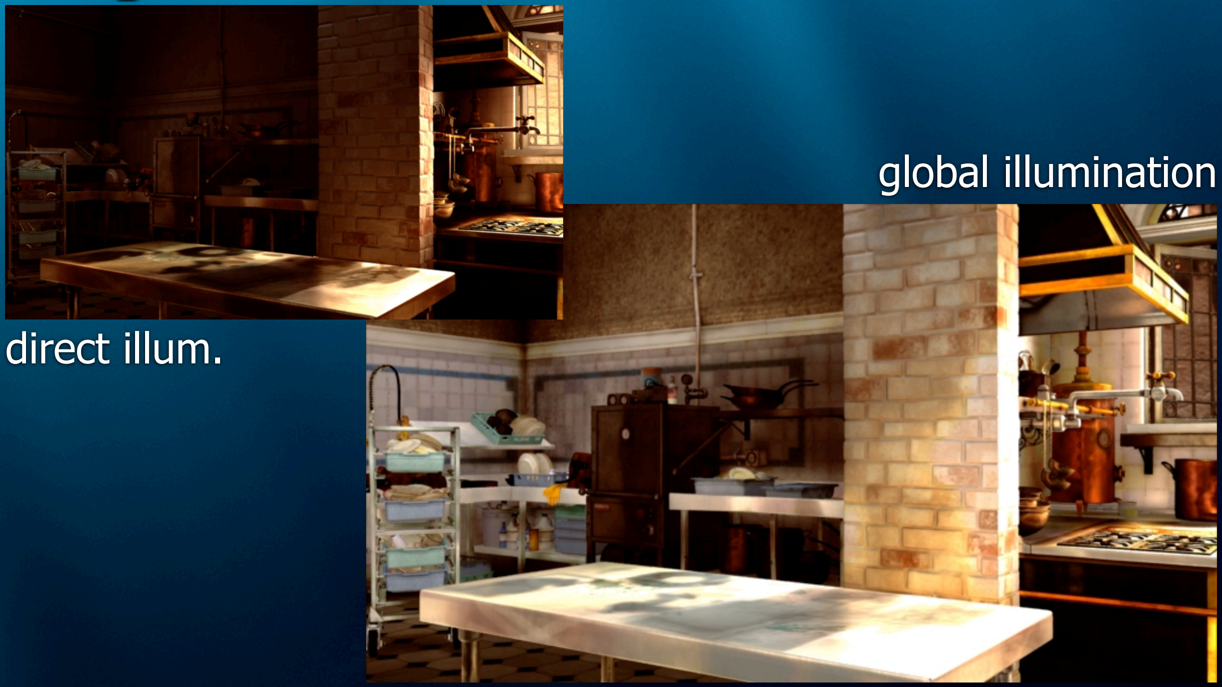

direct illumination (400k points)
photon map (2.6M points)
radiosity map (2.6M points)
global illumination image

‘Ratatouille’ kitchen radiosity map

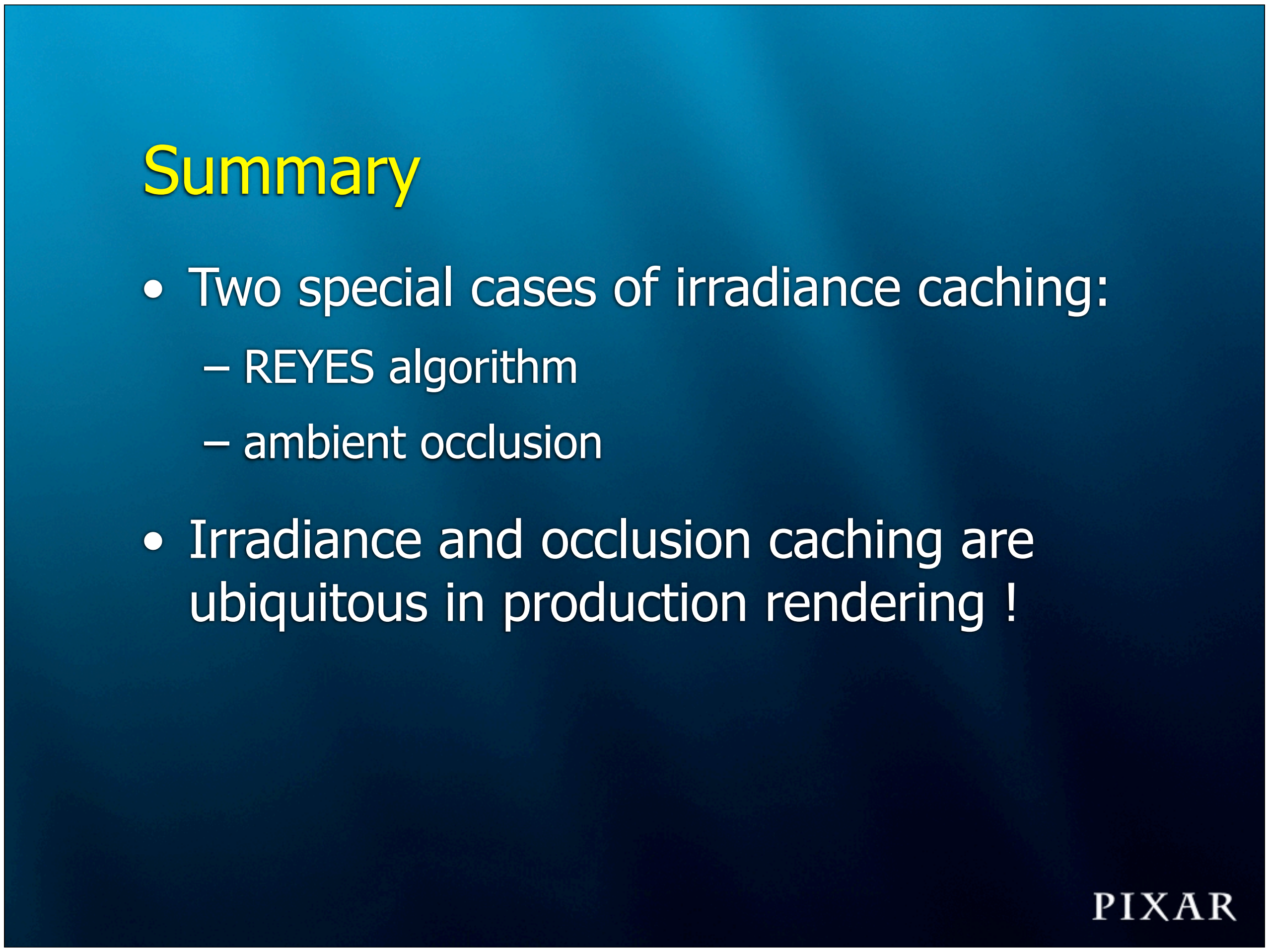


PIXAR

‘Ratatouille’ kitchen: Direct vs. global illumination



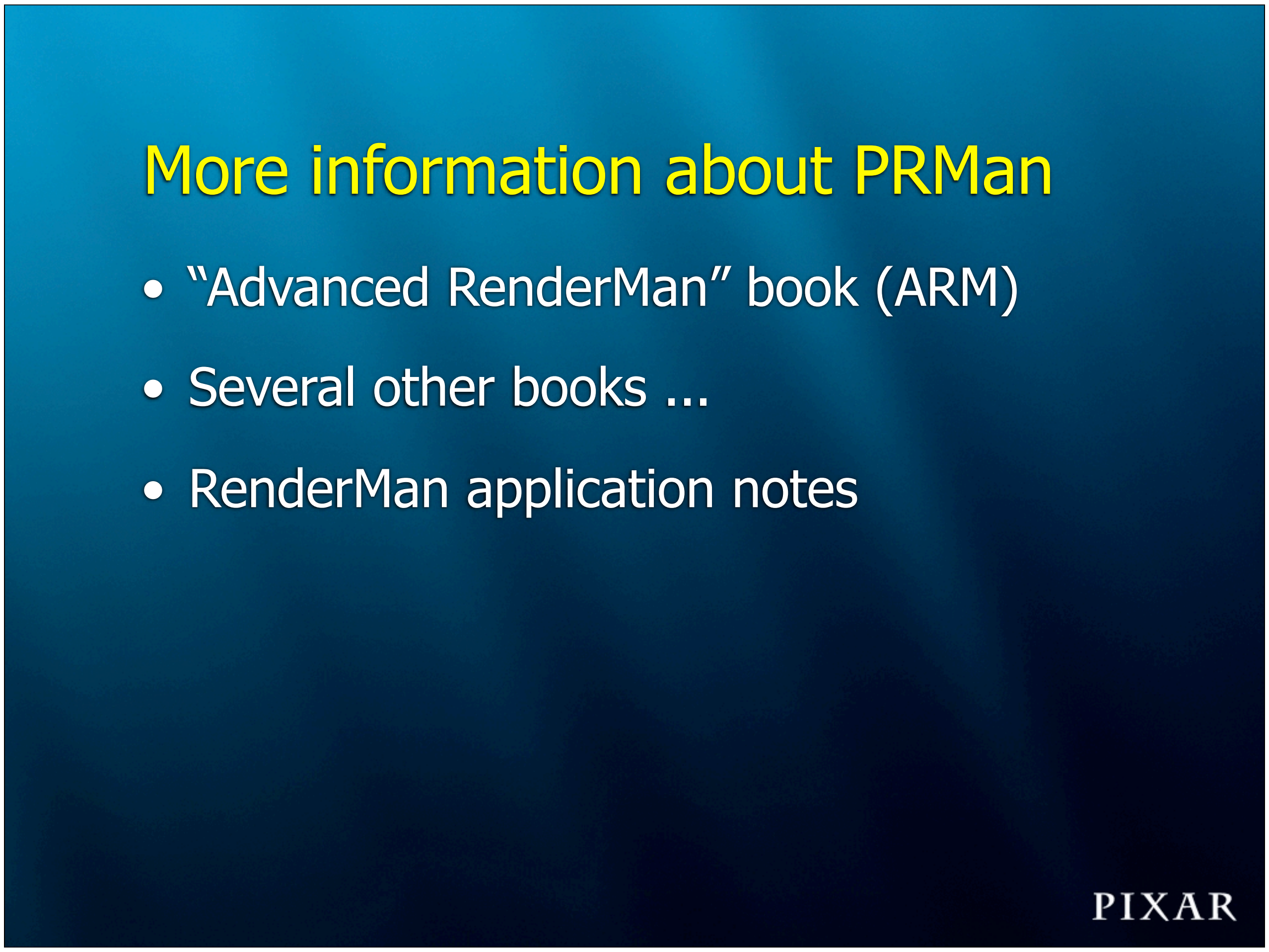
Global illumination was actually not used on the Rat movie, but ambient occlusion was used a lot!
This is a global illumination example I rendered using the kitchen geometry and shaders from the Rat movie.
Notice the complex direct illumination and shading: ~200 light sources, ~150 surface shaders, ~1000 textures.



Summary

- Two special cases of irradiance caching:
  - REYES algorithm
  - ambient occlusion
- Irradiance and occlusion caching are ubiquitous in production rendering !

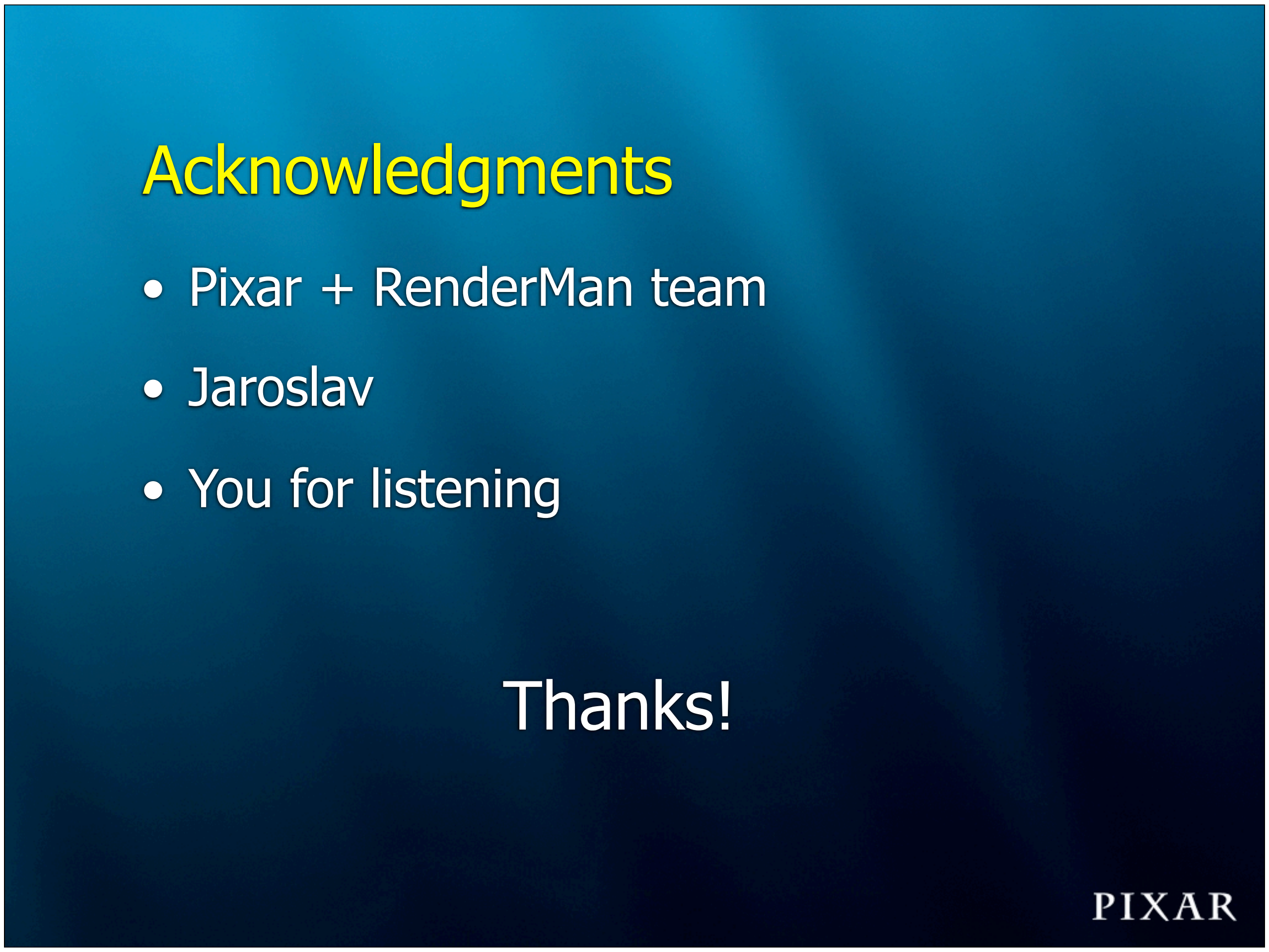
PIXAR



More information about PRMan

- "Advanced RenderMan" book (ARM)
- Several other books ...
- RenderMan application notes

PIXAR



Acknowledgments

- Pixar + RenderMan team
- Jaroslav
- You for listening

Thanks!

PIXAR

Questions?

PIXAR