

Optimizing Realistic Rendering with Many-Light Methods

SIGGRAPH 2012 Course

Course Notes

This is a preliminary version. The final version is available from
<http://cgg.mff.cuni.cz/~jaroslav/papers/mlcourse2012>.

Organizers

Jaroslav Křivánek
Charles University, Prague

Miloš Hašan
UC Berkeley

Lecturers

Adam Arbree
Autodesk, Inc.

Carsten Dachsbacher
Karlsruhe Institute of Technology

Alexander Keller
NVIDIA ARC GmbH

Bruce Walter
Cornell University

Abstract

With the recent improvements in hardware performance, there has been an increased demand in various industries, including game development, film production, or architectural visualization, for realistic image rendering with global illumination. However, the inability of the state of the art algorithms to meet the strict speed and quality requirements fosters more research in this area. Many-light rendering, a class of methods derived from the Instant Radiosity algorithm proposed by Keller [1997], has received much attention in recent years. By reducing light transport simulation to rendering the scene with many light sources, the many-light formulation offers a unified view of the global illumination problem. Unlike with other GI algorithms, the quality-speed trade-off in the many-light methods (in terms of the number of lights) is able to produce artifact-free images in a fraction of a second while converging to the full GI solution over time. This formulation is therefore potentially able to cater to all of the aforementioned industries.

The primary goal of this course is to present a coherent summary of the state of the art in many-light rendering. The course covers the basic many-light formulation, as well as the recent work on its use for computing global illumination in real-time, on improving scalability with a large number of lights, on using many-lights as a basis for a full GI solution, and also on rendering participating media. The course will focus on the clarity of the underlying mathematical concepts as well as on the practical aspects of the individual algorithms. An important part of the course will be devoted to the practical considerations necessary for the use of many-light methods in the Autodesk 360 Rendering service.

Intended audience

Industry professionals and researchers interested in recent advances in realistic rendering with global illumination. Software developers and managers looking for the right global illumination solution for their application will also benefit from the course.

Prerequisites

Familiarity with rendering and with concepts of global illumination computation is expected.

Level of difficulty

Intermediate

Syllabus

1. Introduction (*Křivánek*)
(5 min)
2. Instant Radiosity - principles and practice (*Keller*)
(30 min)
 - Light transport simulation using many point light sources
 - Path space partitioning
 - Consistent generation of light paths
3. Handling difficult paths (*Hašan, Křivánek*)
(30 min)
 - Kollig-Keller method for dealing with singularities and its limitations
 - Virtual spherical lights
 - Improved VPL distribution and local VPLs
4. Scalability with many lights I (*Walter*)
(25 min)
 - Lightcuts and Multidimensional Lightcuts
- Break
(15 min)
4. Scalability with many lights II (*Hašan*)
(20 min)
 - Matrix row-column sampling, Matrix Slice Sampling
 - Visibility clustering
5. Real-time many-light rendering (*Dachsbacher*)
(35 min)
 - Many-light generation with Reflective Shadow Maps
 - Fast rendering with many lights via Imperfect Shadow Maps and Micro-Rendering
 - Approximate bias compensation for high-quality rendering of surface and volume lighting
6. Many-lights methods in Autodesk 360 Rendering (*Arbree*)
(30 min)
 - What is Autodesk Cloud Rendering?
 - Our many-lights rendering algorithm
 - Advantages of a many-light solution
 - Discussion of results
7. Conclusions / Q & A (*all*)
(5 min)

Course presenter information

Jaroslav Krivánek *Charles University, Prague*
jaroslav.krivanek@mff.cuni.cz

Jaroslav is an assistant professor at Charles University in Prague. Prior to this appointment, he was a Marie Curie post-doctoral research fellow at the Cornell University Program of Computer Graphics, and a junior researcher and assistant professor at Czech Technical University in Prague. Jaroslav received his Ph.D. from IRISA/INRIA Rennes and the Czech Technical University (joint degree) in 2005. In 2003 and 2004 he was a research associate at the University of Central Florida. His primary research interest is realistic rendering and global illumination.

Miloš Hašan *UC Berkeley*
milos.hasan@gmail.com

Miloš Hašan is a post-doctoral researcher at the University of California, Berkeley. He received his Ph.D. from Cornell University in 2009, after which he spent one year as a post-doctoral fellow at Harvard University. His primary research interest is in the area of light transport, including efficient global illumination algorithms, precomputed techniques, GPU-oriented algorithms, and applications to fabrication and visualization.

Adam Arbree *Autodesk, Inc.*
adam.arbree@autodesk.com

Adam Arbree is a principle engineer developing rendering applications for Autodesk. He received his Ph.D. from Cornell University in 2009 and, in his dissertation, he created scalable many-lights rendering algorithm for subsurface scattering. For the last two years, he has been designing and building a physically accurate, many-lights rendering system for architectural visualization for Autodesk. The resulting product, Autodesk 360 Rendering, premiered in September 2011 as part Autodesk's global launch of cloud services.

Carsten Dachsbacher *Karlsruhe Institute of Technology*
dachsbacher@kit.edu

Carsten Dachsbacher is Full Professor for computer graphics at the Karlsruhe Institute of Technology, Germany. Prior to joining KIT, he has been Assistant Professor at the University Stuttgart, and post-doctoral fellow at REVES/INRIA Sophia-Antipolis, France. He received a PhD from the University of Erlangen, Germany. His research includes real-time computer graphics, (interactive) global illumination, GPU techniques, and visualization. He has published several articles at various conferences including SIGGRAPH and Eurographics. Carsten has been a tutorial speaker at SIGGRAPH, Eurographics, and the Game Developers Conference.

Alexander Keller *NVIDIA ARC GmbH*
keller.alexander@gmail.com

Alexander Keller is a member of NVIDIA Research and leads advanced rendering research at NVIDIA ARC GmbH, Berlin. Before, he had been the Chief Scientist of mental images and had been responsible for research and the conception of future products and strategies including the design of the iray renderer. Prior to industry, he worked as a full professor for computer graphics and scientific computing at Ulm University, where he co-founded the UZWR (Ulmer Zentrum für wissenschaftliches Rechnen). Alexander Keller holds a PhD in computer science, authored more than 21 patents, and published more than 40 papers mainly in the area of quasi-Monte Carlo methods and photorealistic image synthesis.

Bruce Walter *Cornell University*
bjw@graphics.cornell.edu

Bruce Walter is a Research Associate at the Cornell Program of Computer Graphics. His current research interests are expanding the capabilities of physically-based rendering and global illumination algorithms with respect to robustness, scalability, and generality. He has published many related research papers at SIGGRAPH and elsewhere. He also served a project lead for trueSpace product at Caligari, and was a post-doc in iMAGIS laboratory in Grenoble, France, and earned a Ph.D. from Cornell and a B.A. from Williams college.

Introduction

Jaroslav Křivánek

(Optimizing) Realistic Rendering with Many-Light Methods

An ACM SIGGRAPH 2012 course presented by

Jaroslav Křivánek

Charles University in Prague

Miloš Hašan

UC Berkeley

Adam Arbree

Autodesk

Carsten Dachsbacher

*Karlsruhe Institute of
Technology*

Alexander Keller

NVidia

Bruce Walter

Cornell University

This course covers a group of global illumination algorithms known as “many-light methods”, or “VPL-rendering methods”. (VPL = virtual point light)

Realistic Rendering with Many-Light Methods

Introduction

Jaroslav Křivánek
Charles University in Prague

Global Illumination



No global illumination
(direct illumination only)



Global illumination
(direct + indirect illum.)

3

Global illumination (GI) is indeed one of the most important aspects of realistic rendering as you can see in this example. Global illumination fills shadows with natural light and adds illumination gradients around geometry features, that the human eye is used to seeing in reality.

Global Illumination



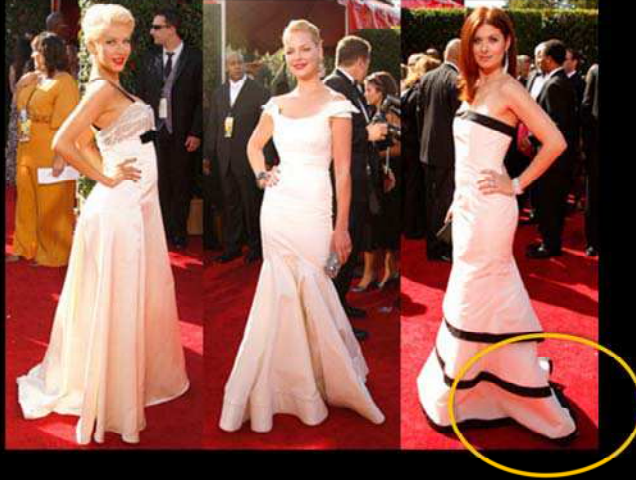
© nikki Candelerio, vray gallery



© Jonas Balzer

GI is extensively used in architectural visualization, product design, and in the movie industry.

Inter-reflection



Slide credit: Michael Bunnell

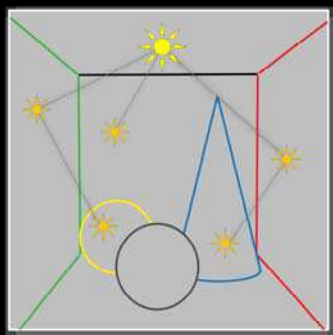
Global illumination is due to multiple inter-reflections of lights, as shown in the example above.

Rendering of realistic images with global illumination then involves simulating these inter-reflections.

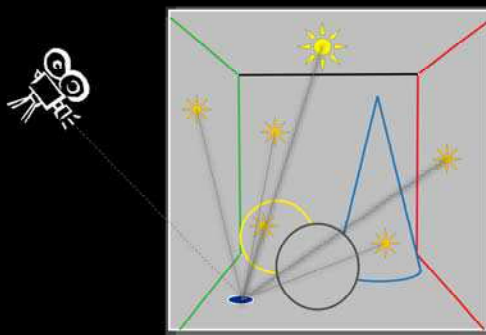
Many-light rendering

- Based on Instant radiosity [Keller 1997]
- Approximate indirect illumination by **Virtual Point Lights (VPLs)**

1. Generate VPLs



2. Render with VPLs



6

The many-light formulation, covered in our course, provides a particularly efficient approach to simulating the light inter-reflections.

These methods originate from the Instant Radiosity algorithm proposed by Alexander Keller. The main idea is to approximate indirect illumination by a number so called Virtual Point Lights, or VPLs.

A basic VPL rendering algorithm works as follows: In the first step, the VPLs are distributed on scene surfaces by tracing particles from light sources. In the second step, the color of a pixel is computed by summing the contributions from all the VPLs to the surface point(s) visible through that pixel.

In other words, the problem of computing indirect illumination has been reduced to the computation of direct illumination from many-lights, hence the name of the methods.

Many-lights: Advantages

- Unified approach
 - Everything represented by virtual lights
 - Area light, environment maps, indirect illumination

7

One of the most important advantages of the many-light formulation is that it unifies the rendering problem to computing direct illumination from a (potentially large) number of lights, the VPLs.

Indeed, large area lights, environment maps, and indirect illumination are seamlessly handled in the same way. All that is needed is to convert the illumination to a set of VPLs.

Many-lights: Advantages

- Spans a wide range of quality/cost ratios

Interactive rendering

16k VPLs, 5.5 fps



5.5 fps, 256², 16k

from [Ritchel et al., SigAsia 2008]

High-fidelity rendering

17M VPLs, 5 min



from [Davidovič et al., SigAsia 2010]

8

The popularity of many-light methods is also due to their wide applicability, from approximate interactive rendering to high-fidelity offline rendering. This is due to the fact that even with a limited number of VPLs, the generated images, though incorrect, provide a visually plausible approximation of the correct solution.

Main technical issues

- Making it fast (as always)
- Making it "asymptotically fast", i.e. scalable
- Making it accurate

9

The main technical issues that need to be addressed when using many-light methods in practice are listed on the slide.

First, as usual, we want to make the rendering fast. This involves especially accelerating the visibility tests required to compute illumination from VPLs. Such improvements lead to a constant factor speed-up.

However, the many-light methods lend themselves to asymptotic speed improvements, i.e. the rendering time can grow much more slowly than the number of VPLs. A prime example of this approach is the Lightcuts algorithm.

Finally, in the basic form, many-light methods suffer from some approximations. Special care is needed to make these methods applicable in high-fidelity rendering applications. The goal of our course is to cover all of these issues.

Course lecturers

- (in the order of appearance)
- Jaroslav Křivánek, *Charles University in Prague*
- Alexander Keller, *NVidia*
- Miloš Hašan, *UC Berkeley*
- Bruce Walter, *Cornell University*
- Carsten Dachsbacher, *KIT*
- Adam Arbree, *Autodesk*

10

The material will be presented by researchers who were originally involved with the development of the various many-light methods.

Course Overview

- 2:00 (05 min) ... **Introduction & Welcome** (*Křivánek*)
- 2:05 (30 min) ... **Instant Radiosity** (*Keller*)
- 2:35 (30 min) ... **Handling difficult light paths** (*Hašan, Křivánek*)
- 3:05 (25 min) ... **Scalability with many lights I** (*Walter*)

- 3:30 (15 min) ... **Break**

- 3:45 (20 min) ... **Scalability with many lights II** (*Hašan*)
- 4:05 (35 min) ... **Real-time many-light rendering** (*Dachsbacher*)
- 4:40 (30 min) ... **ML in Autodesk® 360 Rendering** (*Arbree*)
- 5:10 (05 min) ... **Conclusion - Q & A** (*All*)

Instant Radiosity - principles and practice

Alexander Keller

Slides for this part of the course are provided in the final version available from
<http://cgg.mff.cuni.cz/~jaroslav/papers/mlcourse2012>.

Handling difficult paths

Miloš Hašan, Jaroslav Křivánek

Realistic Rendering with Many-Light Methods

Handling Difficult Light Paths

(virtual spherical lights, improved VPL distribution)

Miloš Hašan
UC Berkeley

Jaroslav Křivánek
Charles University in Prague

Realistic Rendering with Many-Light Methods

Virtual Spherical Lights

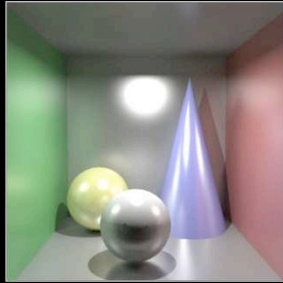
In this part of the course, I will discuss virtual spherical lights, a technique that can reduce the energy loss problems encountered with standard formulations of virtual point lights.

Glossy Inter-reflections

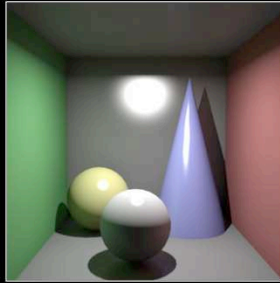


One important application is architectural or industrial previews. They often contain a significant fractions of glossy materials, which create interreflections that cannot be neglected. These are somewhat extreme examples: their appearance is completely dominated by glossy inter-reflections. But they do present nice examples of scenes that will bring classic many-light algorithms to their knees.

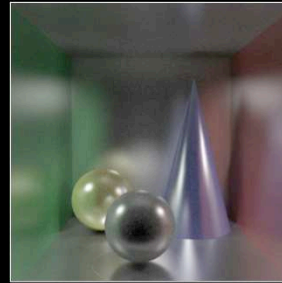
Problem



Path tracer



Instant radiosity



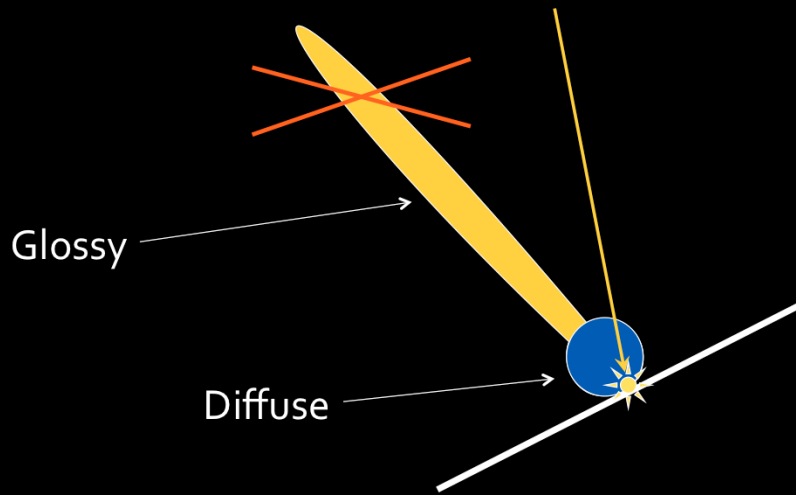
Difference image

4

So let's say we run standard instant radiosity on this glossy Cornell box. If we do it naively, we quickly find that we have to play tricks which will remove most of the interesting interreflection effects. These tricks are: consider only diffuse VPLs and apply plenty of clamping. Let's look at these in more detail.

Emission Distribution of a VPL

- Cosine-weighted BRDF lobe at the VPL location

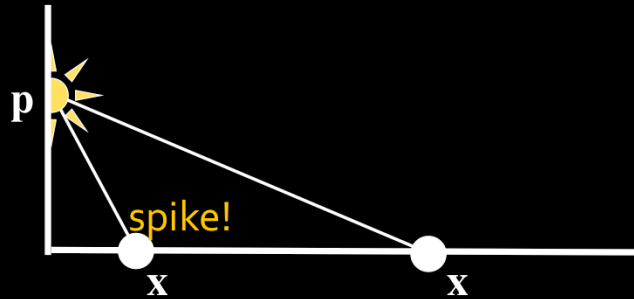


Here is what a VPL emission function should look like to give unbiased results: it is simply a combination of a diffuse and glossy BRDF lobe corresponding to the incoming direction of the VPL, multiplied by the cosine term.

We can easily see the issue – if the surface is highly glossy, the BRDF lobe will function as a „laser“ light that will create unacceptable spikes in random places around the scene. This problem is usually avoided by not including the glossy contribution of the VPL, which is clearly suboptimal.

Clamping the Remaining Spikes

As $\| \mathbf{p} - \mathbf{x} \| \rightarrow 0$, VPL contribution $\rightarrow \infty$



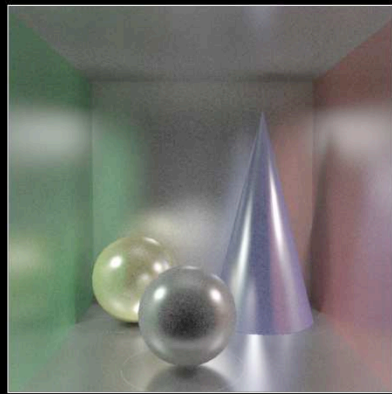
- Common solution: **Clamp** contribution

6

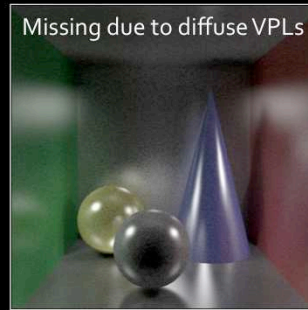
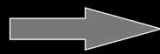
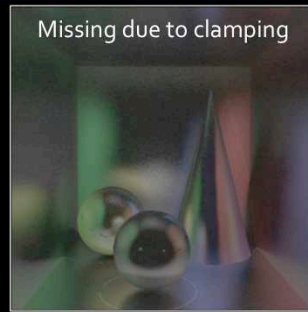
However, this will not be sufficient. As we move the VPL location \mathbf{p} and the shading point \mathbf{x} closer and closer to the corner, the VPL contribution will go to infinity. This problem will be even worse if the BRDF at \mathbf{x} is also glossy (remember, we only removed the glossy BRDF from the VPL location \mathbf{p}).

The common solution to this is to clamp the VPL contribution to a user-specified maximum value. This not only removes illumination, but introduces another tricky parameter to the user, which is difficult to set automatically.

The Missing Components



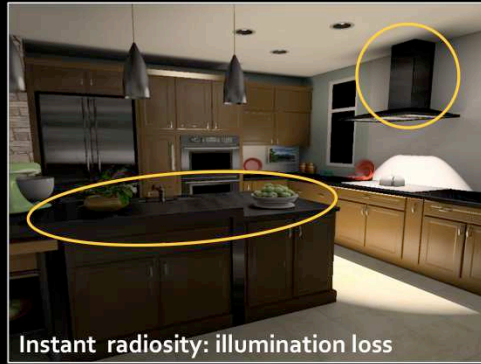
Missing energy



7

These two approximations cause a large chunk of the indirect illumination to be missing. Clearly, neither clamping and using diffuse-only VPLs can be neglected in our search for a better solution.

Real Scenes: Same Problem



8

One may think that this problem only occurs in a contrived scene such as the Cornell box, where we made all surfaces glossy. This is not the case: in this kitchen scene, instant radiosity some important parts of light transport, which results in serious illumination loss on glossy surfaces, as you can see on the range hood or the counter.

Clamping Compensation

- Compute the missing components by path tracing [Kollig and Keller 2004]



Path traced compensation: 3.5 hours



Reference

- Glossy scenes
 - As slow as path-tracing everything

9

The illumination loss problem hasn't been extensively discussed in previous work with the exception of the paper by Kollig and Keller, who propose to compensate for the missing energy by path tracing. Unfortunately, in glossy scenes, their compensation methods can be nearly as expensive as path tracing the entire image.

Our Method

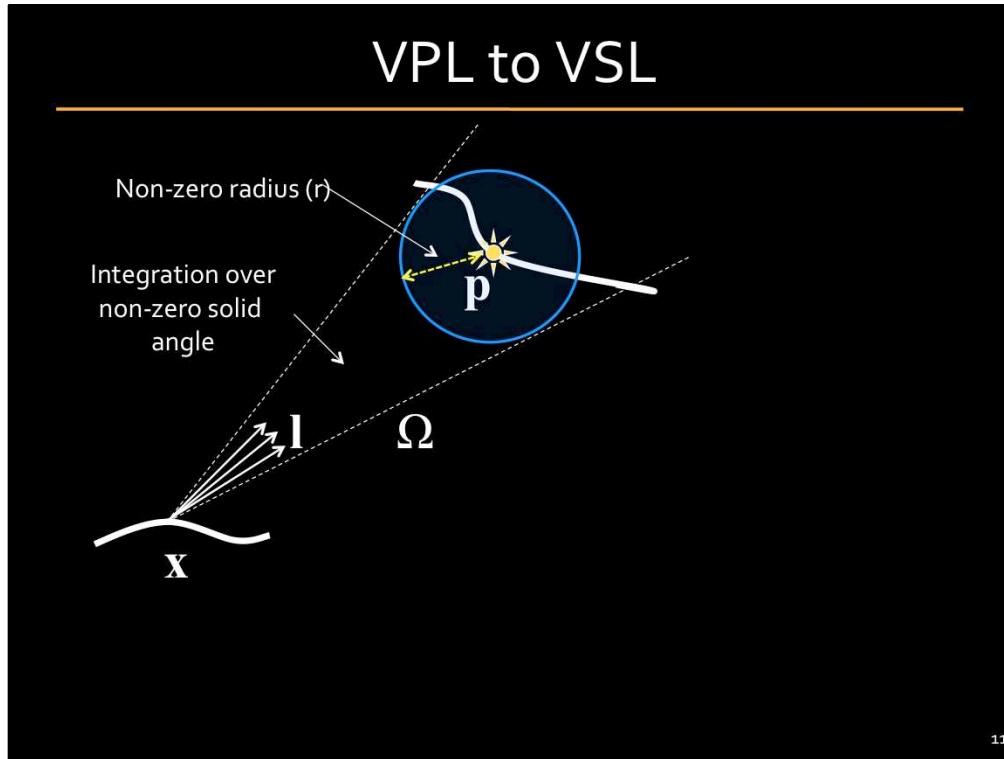
- New type of light: Virtual **Spherical** Light



Our idea, on the other hand, is to prevent the illumination loss in the first place, rather than trying to compensate for it. We achieve this by introducing a new type of light, the VSL, that overcomes some of the problems of VPLs.

With this new type of light, we are able to render images very similar to the reference yet in much shorter time.

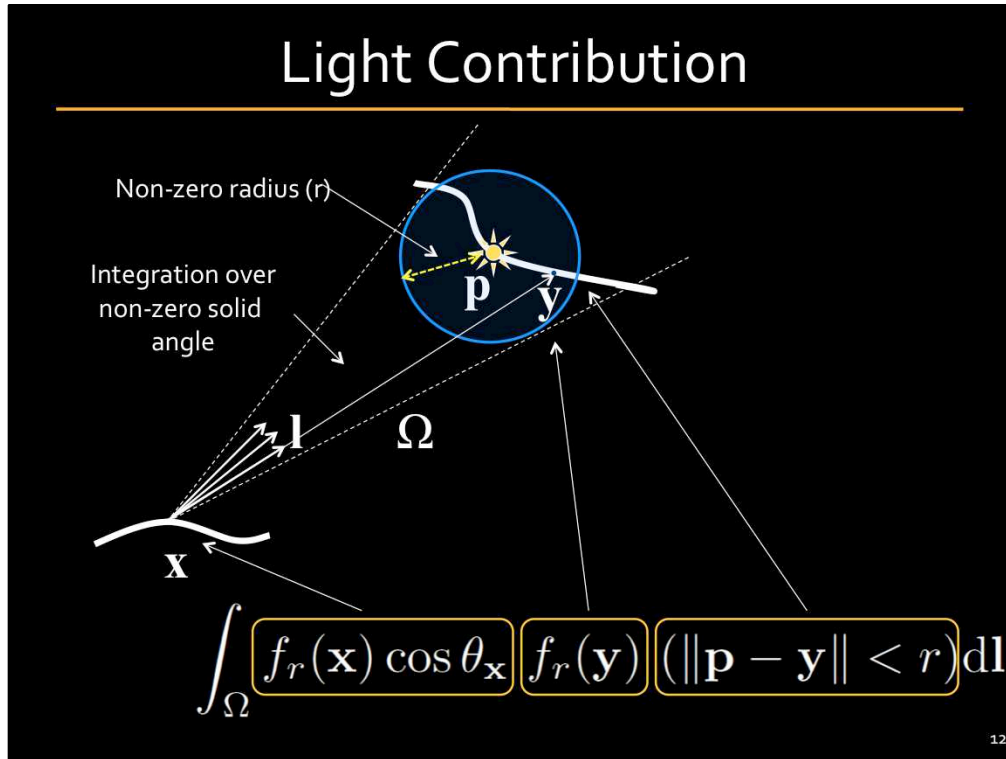
VPL to VSL



More specifically, we will spread the light energy over the surfaces inside the sphere of radius r centered at the light position p . And the contribution of the light will be computed as an integral over the solid angle subtended by the sphere.

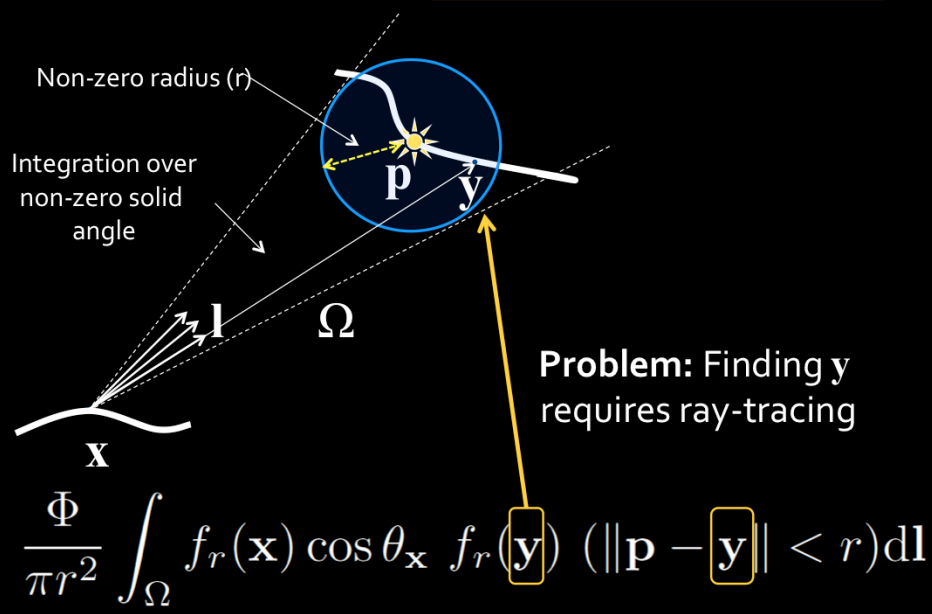
This can be seen as an analogy to photon mapping: A photon contributes to all surfaces within radius r , and an additional final gather operator uses the “splatted” illumination to light other surfaces.

Light Contribution



Let's write down the formula for the contribution of such a light to the surface point x . We have the integration over the solid angle. The integrand is a product of the following terms: the cosine weighted BRDF at the surface, next, the BRDF at the point y in the vicinity of the light location. Finally, we have an indicator term that is zero for all the directions that correspond to surface point y outside the sphere. We normalize the integration by the expected surface area inside the sphere, $\pi \cdot r^2$, and multiply by the light flux. To avoid this indicator term, we could define the light contribution as an integral over a disk area. Unfortunately, doing that re-introduces the infamous $1/\text{dist}^2$ term and produces bad results (we tried it).

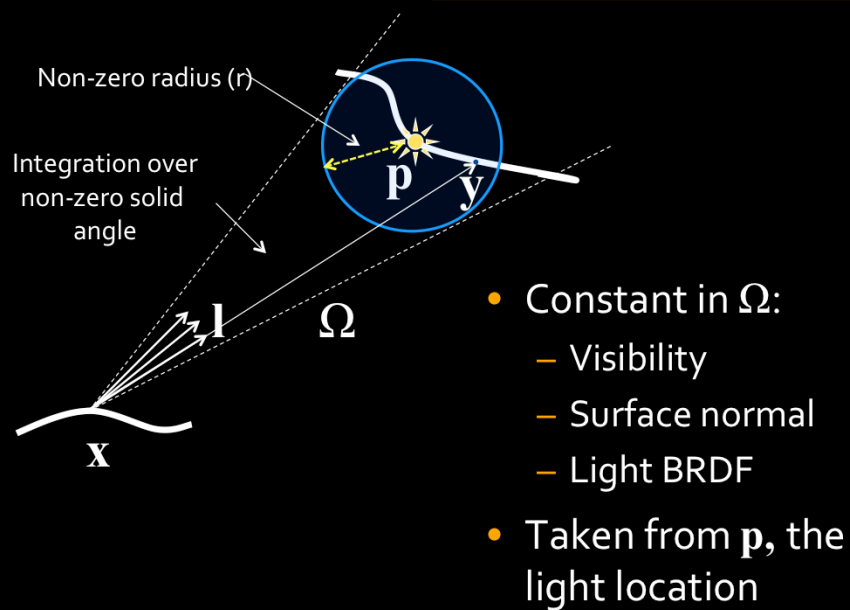
Light Contribution



13

Unfortunately, this formulation requires finding the point \mathbf{y} for all directions \mathbf{l} inside the cone, which requires ray tracing. This is clearly not feasible.

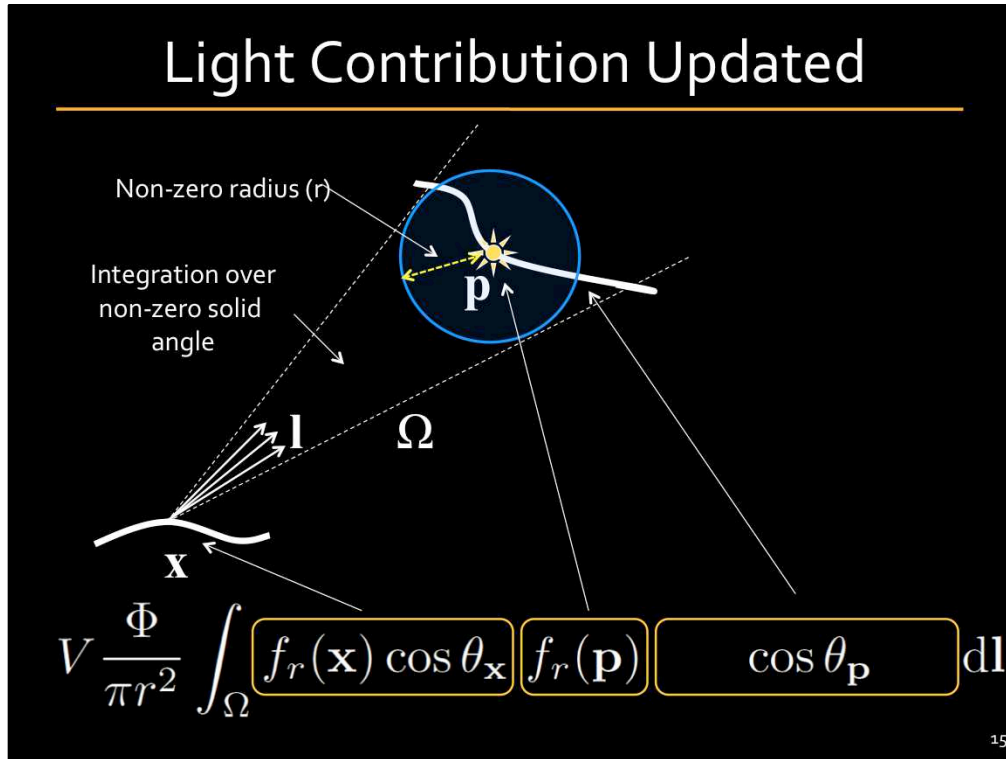
Simplifying Assumptions



14

To produce a computationally convenient approximation to the previous formula, we make the following simplifying assumptions. We assume that the visibility, the surface normal and the BRDF are constant inside the sphere. And we take them from the light location \mathbf{p} .

Light Contribution Updated



With these assumptions, we can write a formula for the contribution of a VSL.

The indicator function on the right will be approximated by a cosine term – this approximation tends to be correct as the distance between the light and the shading point increases.

Now we have arrived at a nice, clean formulation of the VSL contribution: it becomes an integral over the conical solid angle given by the sphere radius, and the integrand will be the product of two BRDFs and two cosine terms: one each for the receiver and the light location.

Virtual Spherical Light

- All inputs taken from \mathbf{x} and \mathbf{p}
 - Local computation
- Same interface as any other light
 - Can be implemented in a GPU shader
- Visibility factored from the integration
 - Can use shadow maps

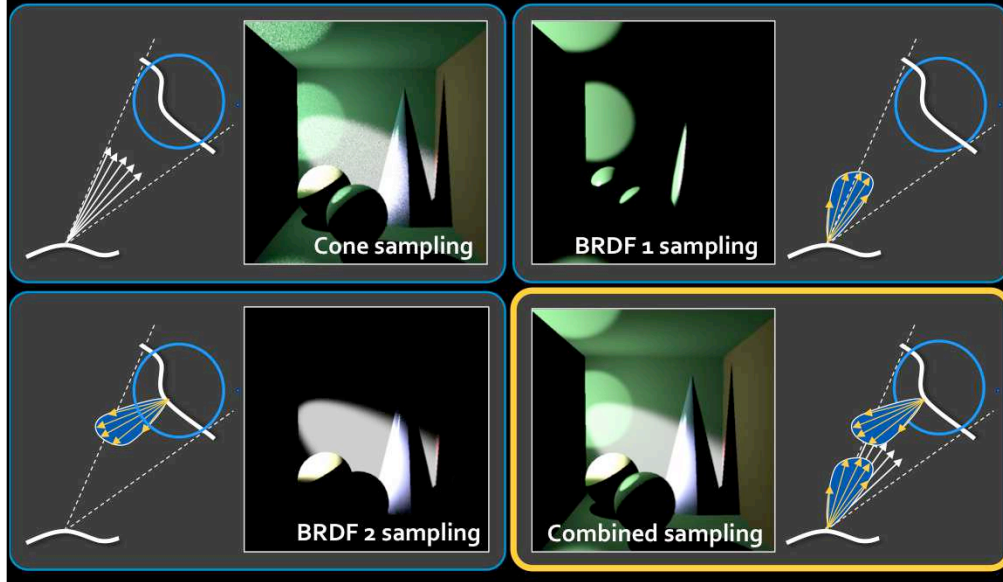
$$V \frac{\Phi}{\pi r^2} \int_{\Omega} f_r(\mathbf{x}) \cos \theta_{\mathbf{x}} f_r(\mathbf{p}) \cos \theta_{\mathbf{p}} d\mathbf{l}$$

16

This formulation is very practical – no ray-tracing is necessary to compute the contribution, and all necessary parameters are local to either the shading point or the light. It still does require Monte Carlo integration, but this is purely numerical and can be done in a shader. We can use shadow maps for visibility if desired, as usual.

Computing the VSL integral

- Stratified Monte Carlo in a shader



As I mentioned, we can use stratified Monte Carlo to compute the VSL integral. One possible issue, in case one or both of the BRDFs involved have a glossy component, is that uniform sampling of the solid angle cone will not be well adapted to the integrand and lead to noise.

However, we can use multiple importance sampling – a variation of the same technique used in bidirectional path tracing. In addition to cone sampling, we can importance-sample either of the two BRDFs, and combine these estimators using the classic balance heuristic.

You can find the shader that computes the VSL integral online.

Implementation

- Matrix row-column sampling [Hašan et al. 2007]
 - Shadow mapping for visibility
 - VSL integral evaluated in a GPU shader
- Need more lights than in diffuse scenes
- VSL radius proportional to local VSL density
 - determined by k-NN queries

18

Our implementation uses the row-column sampling technique to reduce the number of VSLs, which I will also describe later, with shadow mapping for visibility.

However, the VSL can be included into any many-light renderer, even lightcuts, as Adam Arbree will describe later.

We generate about 200,000 VSLs and reduce them to 10,000 using MRCS. These numbers are higher than in diffuse scenes – the VSL does not magically make glossy scenes as easy as diffuse, but does make them tractable for many-light approaches.

An important detail is setting the VSL radius: we find the k nearest virtual lights (say 10) and set the radius as a multiple of that. This makes the radii larger in areas where lights are sparse.

Results: Kitchen

- Most of the scene lit indirectly
- Many materials glossy and anisotropic



19

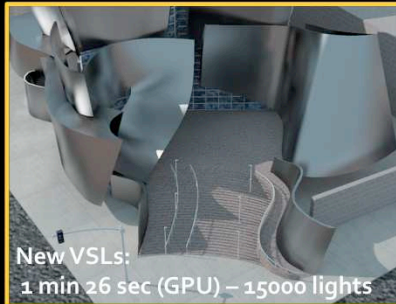
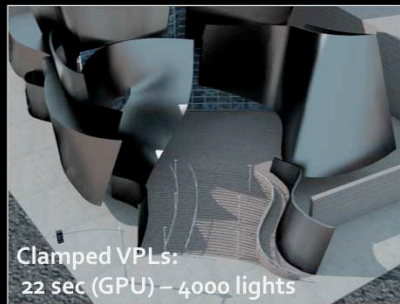
Here's an example of the results we can get with VSLs. This scene is lit by mostly indirect light, through reflection from the shiny metallic surface on the right. There are several other highly glossy and anisotropic materials.

The image computed using the classic approach of clamping and diffuse VPLs looks clean but obviously dark, with some metals close to black.

In contrast, the VSL image is quite close to the path-traced ground truth, with a bit of blurring.

Results: Disney Concert Hall

- Curved walls with no diffuse component
- Standard VPLs cannot capture any reflection from walls



20

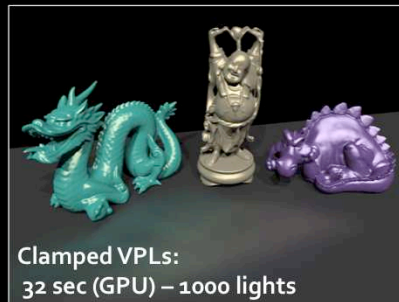
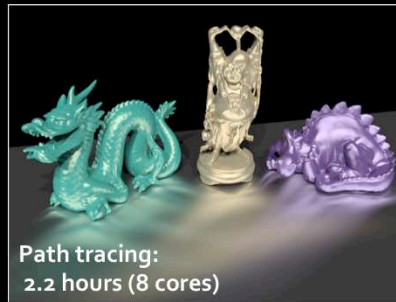
A similar result arises in this model of the Disney concert hall, lit by a sun-sky model.

The walls of the building are purely glossy with no diffuse component: the blue color comes from the sky and the brown is a reflection of the ground.

Obviously, using diffuse VPLs will not capture any illumination off the walls; on the other hand, with VSLs the image looks quite close to the reference, again with some blurring.

Results: Anisotropic Tableau

- Difficult case
- Standard VPLs capture almost no indirect illumination



This tableau consists of an anisotropic metallic plane with some objects, and 3 strong directional lights. This is a really bad case for clamping, which removes much of the reflection. On the other hand, VSLs capture it nicely, only slightly blurred.

Limitation: Blurring

- VSLs can blur illumination
- Converges as number of lights increases



5,000 lights - blurred



1,000,000 lights - converged

22

From the results it is apparent that the main limitation of VSLs is bias in the form of blurring. However, this is a very predictable effect, and in many cases may be acceptable. It is also consistent similar to photon mapping – results get more correct as VSL number increases (and therefore VSL radii decrease).

Jaroslav will later describe the local light technique, which improves upon this limitation of VSLs, at the cost of somewhat higher complexity.

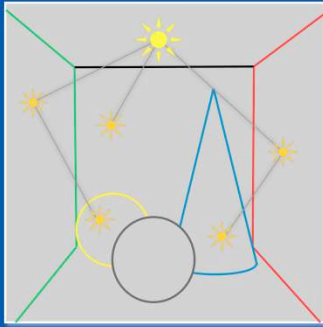
Realistic Rendering with Many-Light Methods

Improved VPL distribution

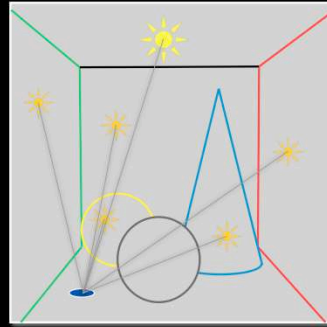
In this part of the course, I will discuss various approaches for generating VPLs where they are most needed for a given camera view.

VPL rendering

1. Distribute VPLs



2. Render with VPLs



24

Let us start by reviewing the classic VPL rendering algorithm, instant radiosity. In the first step, the VPLs are distributed on scene surfaces by tracing particles from light sources. In the second step, the image is rendered by summing contributions from all the VPLs. In this part of the course, I will focus on various approaches to distributing the VPLs.

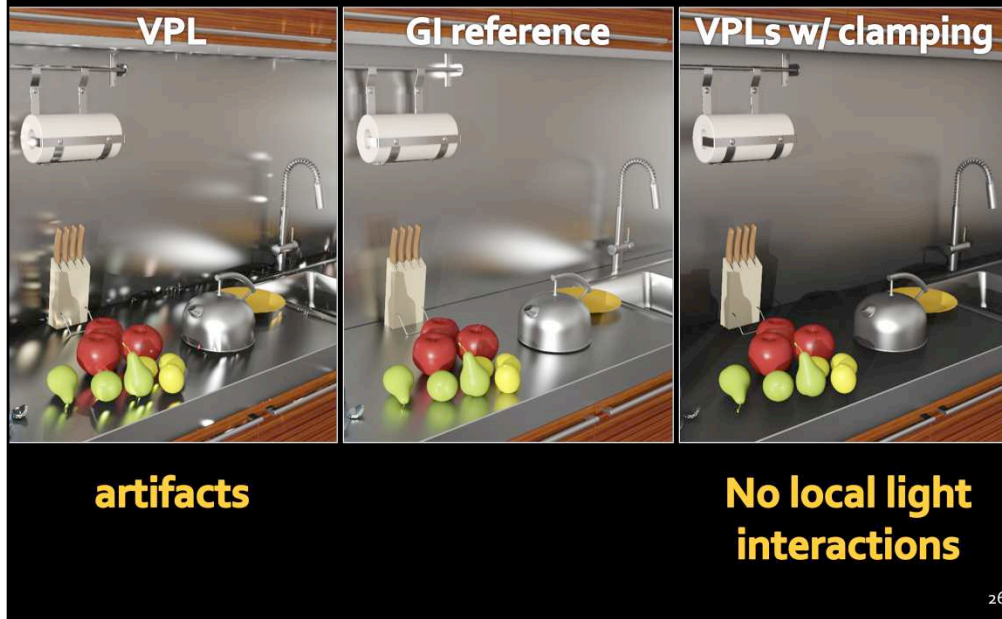
Why alternate VPL distribution?

- VPL may not end up where need
 - Large environment w/ complex visibility
 - Local light inter-reflections

25

The need to develop alternate VPL distribution approaches follow from the fact that with the basic VPL tracing algorithm, the VPLs may end up in regions where they do not contribute significantly to the image. This will be the case especially in large environments where the camera is looking at a small portion of the scene. In addition, VPL distributions generated by the basic VPL tracing algorithm cannot be used to render local light inter-reflections.

Example – local light inter-reflections



Here is an example of the inability of VPLs to reproduce local light inter-reflections. The number of VPLs along the edges is insufficient to render the local inter-reflections, resulting in artifacts in the form of light splotches.

The usual way of dealing with these artifacts is the *clamping* that we discussed previously, where we clamp limit the contribution of a single VPL to a prescribed maximum value.

But this selective energy removal can severely change material appearance, as you can see in the image on the right.

A better solution would be to ensure that more VPLs are generated in the visible areas along the edges.

Purpose & approach

- Purpose
 - Ensure VPLs end up where needed
- Approaches
 - Rejection of unimportant VPLs
 - Metropolis sampling for VPL distribution
 - Distribute VPLs by tracing paths from the camera

27

So, the purpose is to get the VPLs where they are most needed.

There has been a number of approaches proposed in the literature for this purpose and I will discuss some of them in the remaining part of my contribution.

The simplest approach is to apply rejection sampling, where VPLs that do not significantly contribute to the image are rejected.

Second, we can use a more advanced sampling algorithm such as Metropolis sampling. And finally, we can distribute the VPL by tracing paths from the camera instead of from light sources.

Rejection of unimportant VPLs

Rejection of unimportant VPLs

- Autodesk 360 Rendering
 - Covered by Adam later in the course
- [Georgiev et al., EG 2010]
 - Covered on the following slides (courtesy of Iliyan Georgiev)
- Good for large environments but not for local interactions

29

A form of rejection sampling is used for VPL distribution in the Autodesk 360 Rendering solution that will be later described by Adam Arbree. So I will only briefly mention the approach presented by Georgiev et al. in their EG 2010 short paper.

VPL rejection – idea

- Accept VPL proportionately to their total image contribution
 - Reject those that contribute less than average

30

The main idea is very simple:

They use the exact same VPL tracing algorithm as in instant radiosity but they probabilistically reject the VPLs if their image contribution is less than an average.

VPL rejection – algorithm

- Want N VPLs with equal image contribution Φ_v/N
- For each VPL candidate i with energy L_i
 - Estimate total image contribution Φ_i
 - Accept w/ probability $p_i = \min\left\{\frac{\Phi_i}{\Phi_v} + \varepsilon, 1\right\}$
(update energy of an accepted VPL to L_i / p_i)

31

So our goal is to end up with N VPLs, each having some “average” total contribution to the image Φ_v/N .

The algorithm generates one VPL at a time. For each candidate VPL generated by the original VPL tracing algorithm, the total image contribution of the VPL, Φ_i , is estimated. Then the VPL is accepted with a probability given by the ratio of the VPL contribution to the target contribution. If accepted, the VPL energy is boosted by dividing by the acceptance probability: Russian roulette in action.

Estimating image contribution

- No need to be precise
- Estimating Φ_v (average VPL contribution)
 - Based on a few pilot VPLs
- Estimating Φ_i (contribution of VPL candidate i)
 - Contribution to only a few image pixels

32

So how do we estimate the target “average” VPL contribution? A simple way to do it is to run a number of pilot VPLs and render a low-resolution image. Another possibility is to use information from the previous frame, if rendering an animation.

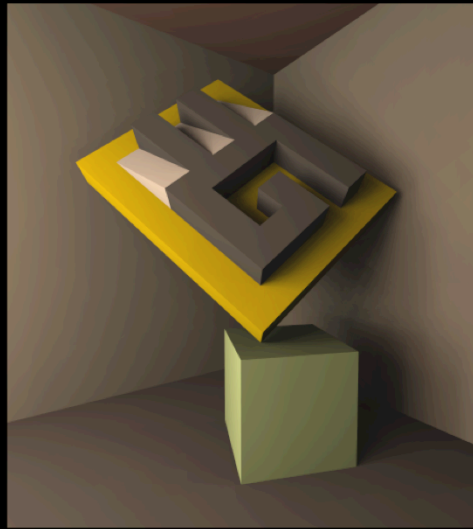
To estimate the image contribution of a candidate VPL, we simply render a “low-res” image by picking only a couple of pixels.

There is no need to be very precise - it's no use to spend much time on estimating the VPL contributions. The algorithm will produce correct results no matter how accurately the VPL contribution is estimated.

Results

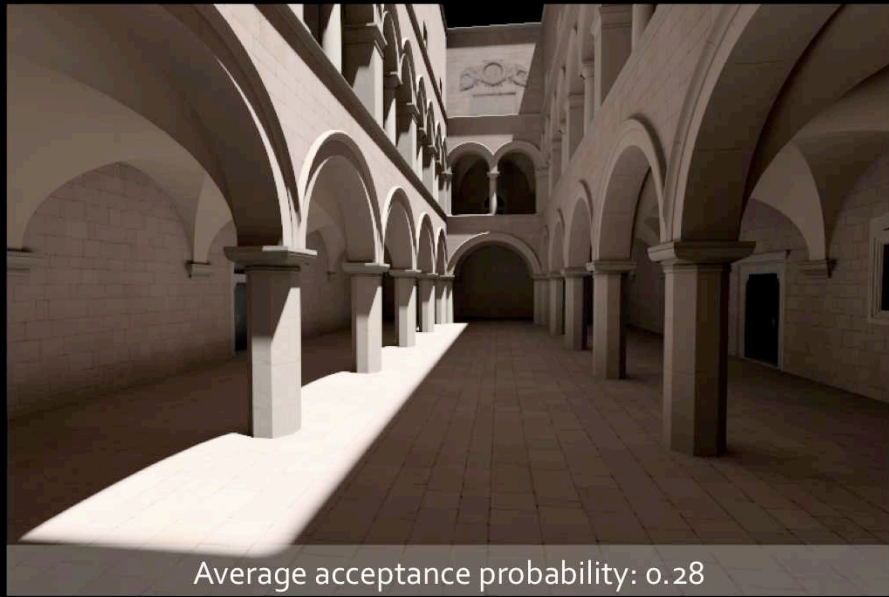


Instant Radiosity



[Georgiev et al. 2010]
(0.07 acceptance)

Results



Average acceptance probability: 0.28

Results



Average acceptance probability: 0.23

Georgiev et al. 2010 – Conclusion

- Cheap & simple
- Can help a lot
- “One-pixel image” assumption
 - Not suitable for local light inter-reflections

36

To conclude, VPL rejection sampling is cheap and simple and can help a lot. There's really no reason for not using it, especially in mostly diffuse scenes.

The problem is that it makes the “one-pixel image” assumption – it will not help us to resolve the local inter-reflection problem.

Metropolis sampling for VPL distribution

Metropolis sampling for VPL distrib.

- “Metropolis instant radiosity”
[Segovia et al., EG 2007]

- Good for large environments but not for local interactions

-
- Slides will be added in the final version
 - please see the course web page

Sampling VPLs from the camera

Sampling VPLs from the camera

- Guaranteed to produce VPLs important for the image
- Technical issues
 - Connection to light sources
 - Computation of pdfs

41

Another option is to distribute the VPLs by tracing paths from the camera instead of from the light sources.

This approach is bound to produce VPLs in locations important for the image to be rendered, but it also has some issues:

First, we need to explicitly connect these VPLs to the light sources so that they can form complete light transport paths.

Second, computing the VPL intensity involves the evaluation of the probability density of generating the particular VPL position. The probability calculation is significantly more complex (and costly) when the VPLs are generated from the camera.

Sampling VPLs from the camera

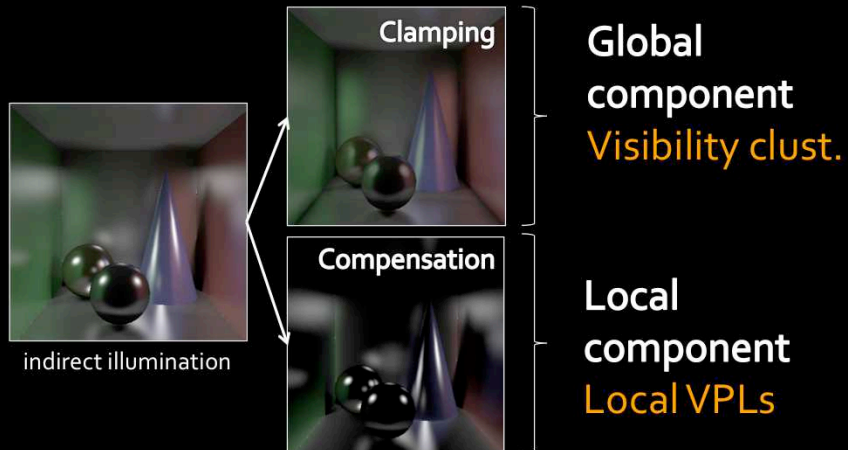
- “Bidirectional instant radiosity”
[Segovia et al., EGSR 2006]
- “Local lights”
[Davidovič et al., SigAsia 2010]

42

On the following slides, I will discuss the method proposed by Davidovič et al. in our SIGGRAPH Asia 2010 paper.

[Davidovič et al. 2010]

- Split illumination

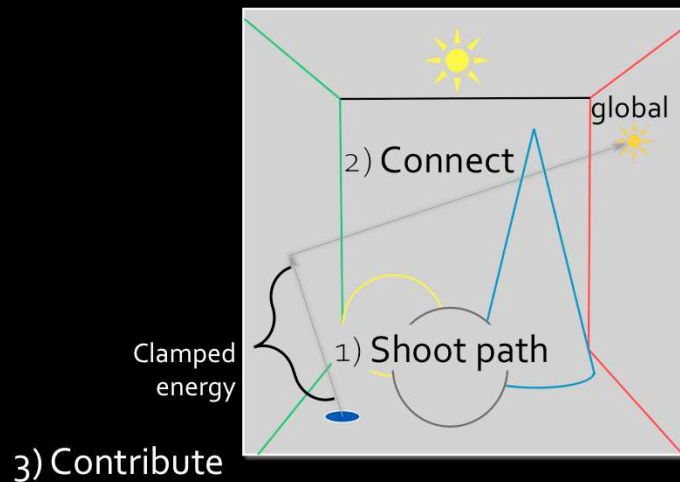


43

We use the idea of separating the light transport into the clamped, global component, and the local component, as previously discussed by Alexander and Miloš. The global component accounts for the long-distance light transfer, while the local component corresponds to the short-range inter-reflections, and indirect glossy highlights. We take advantage of the specific structure of each of the two components to design a solution for each of them that is substantially more efficient than a general GI solution. Specifically, we handle as much energy as possible in the global component which leaves only the local inter-reflections for the local component, which we handle by the so called **local VPLs** that are distributed by tracing paths from the camera.

Review of compensation

- Kollig & Keller compensation

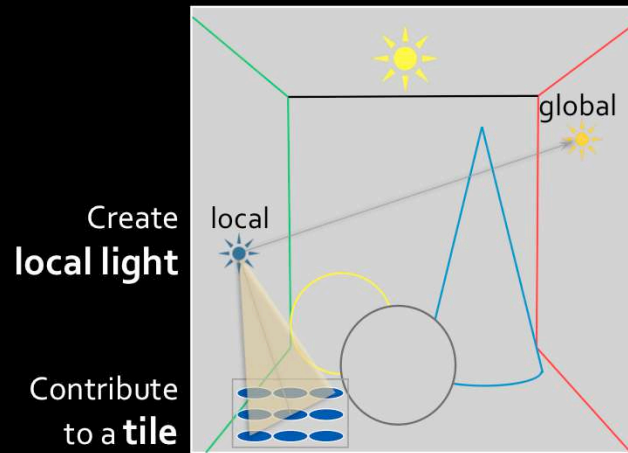


Let us start with the overview of the standard Kollig & Keller compensation. They first trace a ray from the hit point, and connect the target to global light to obtain the intensity. **This path tracing result is then used in place of the clamped away energy.**

This is a lot of effort for compensating just a single pixel.

Local lights – idea

- Our approach



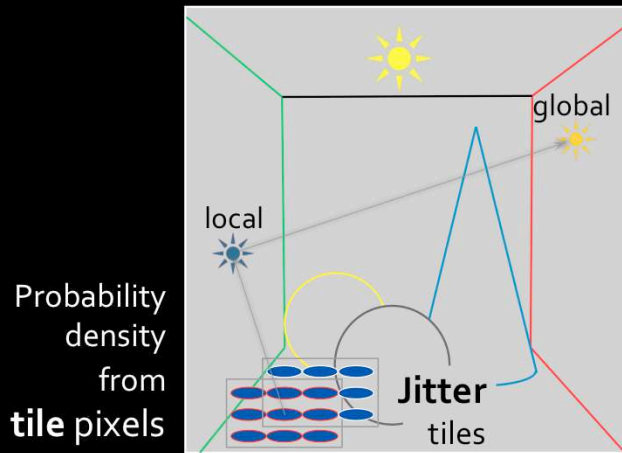
45

Our idea is to create a VPL at the intersection of the camera path. This way, the cost of tracing that path is amortized by letting it contribute not just to the pixel that generated the VPL, but also to its neighboring pixels.

With this basic idea, let us take a closer look at what is necessary to actually make it work.

Local lights – technical solution

- Our approach



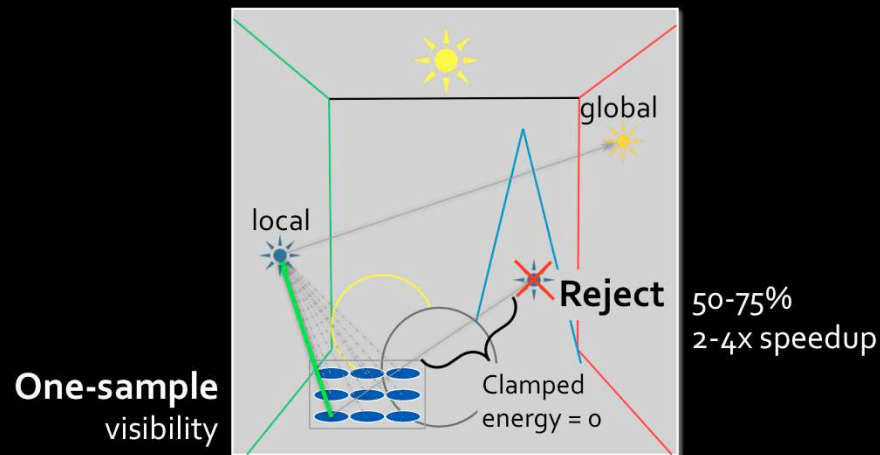
46

The first important thing we have to compute is the probability density of the generated local VPL.

We cannot simply use the probability with which it has been generated, but we have to sum the probabilities over all the pixels in the tile it contributes to. The reason is that all the neighboring pixel could potentially have generated the VPL at this particular location. The second important thing is that if we used a fixed tile grid, the boundaries would be fairly visible. To break this coherence, we jitter the tiles for each VPLs.

Local lights – technical solution

- Our approach



- Key idea: **Tile visibility approximation**

47

Now we come to the part that would simply not be possible without splitting the light transport.

To compute the full probability density for our light, we would normally need to compute visibility to all pixels, which would be prohibitively expensive.

However, we do have the global component that contains most of the energy and handles most of the shadows.

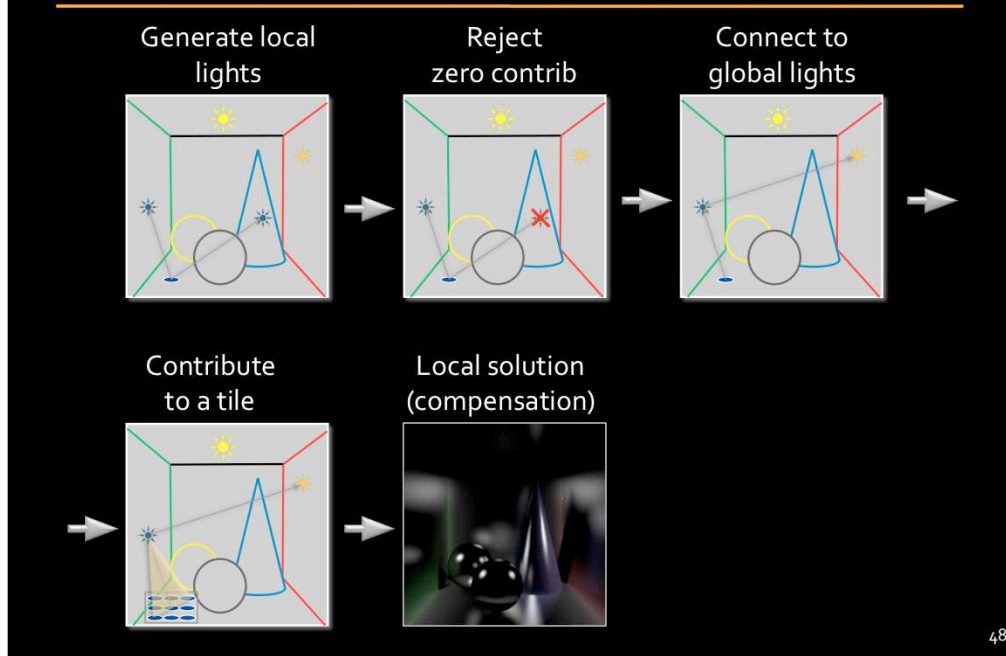
The local lights can then have their visibility approximated because they only handle local inter-reflections.

We approximate it by just one visibility sample, which is actually the ray that generated the light in the first place.

So the key insight here, the light transport split made tile visibility approximation possible.

The second thing the split allows us to do is rejecting lights. Not all local lights actually have contribution to the tile they belong to, and we can flat out reject these, giving us another 2-4x speedup.

The complete local solution



48

So, for the overview of the whole process.

We generate local lights

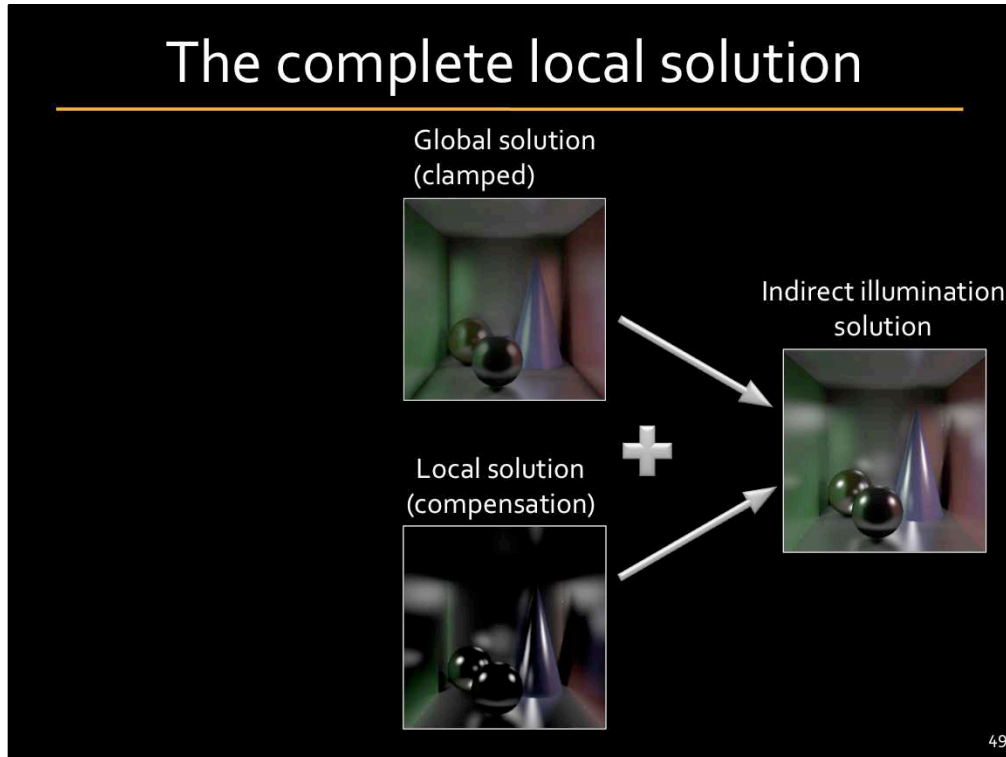
Reject lights with zero contribution

Connect the surviving local lights to global lights

Have them contribute to a tile

And after repeating this about 20 million times, we get the final local solution.

The complete local solution



Now we have the result of the local solution. We simply add the result of the global solution and obtain the final indirect illumination solution.

The global solution can be computed by any VPL method, for example Lightcuts. In the original paper we used a visibility clustering algorithm.

Results



- local lights: 17,100,000

Results



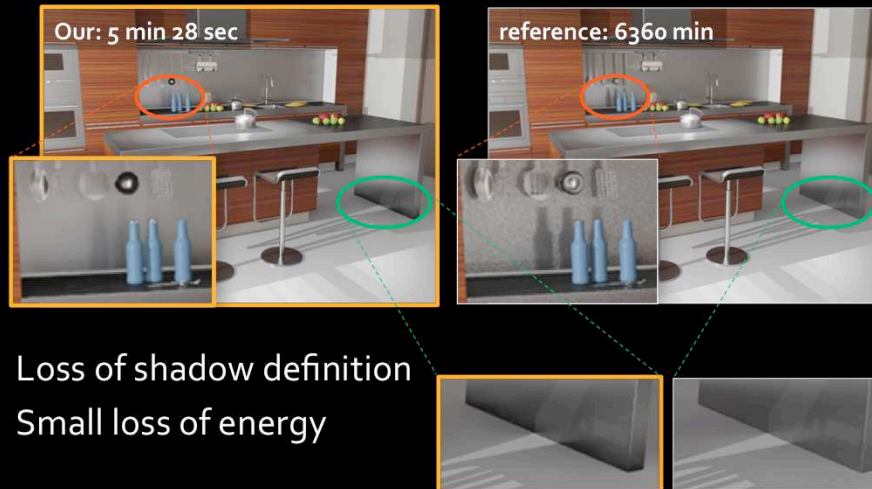
- local lights: 17,100,000



51

Here we see that the local lights nicely capture this highlight from metal stool leg or the reflection of the paper towel on the metal back wall, something that cannot be done with the “globally distributed” VSLs.

Limitations



52

However, this scene also shows some of the limitations the method has. There is a loss of definition of the shadows behind the bottles. This is caused by the fact that the local lights on the kettle in the front contain too much energy. Pushing more energy into the global component could resolve this problem.

One detail we did not mention is that in some of the scenes we still need slight clamping even on the Local VPLs, causing some darkening here. This can be solved by interpreting the Local VPLs as Local VSLs.

Local VPLs – Conclusions

- Good for local inter-reflections
- Really useful only when used in conjunction with a separate “global” solution

53

To conclude, distributing VPLs by tracing paths from the camera is very useful for resolving local inter-reflections.

They are best used in conjunction with a separate “global” solution which can take care of the smooth, long distance light transport in the scene.

Thank you



Scalability with many lights II

Miloš Hašan

Scalability with many lights II

(row-column sampling, visibility clustering)

Miloš Hašan

Scalability with many virtual lights

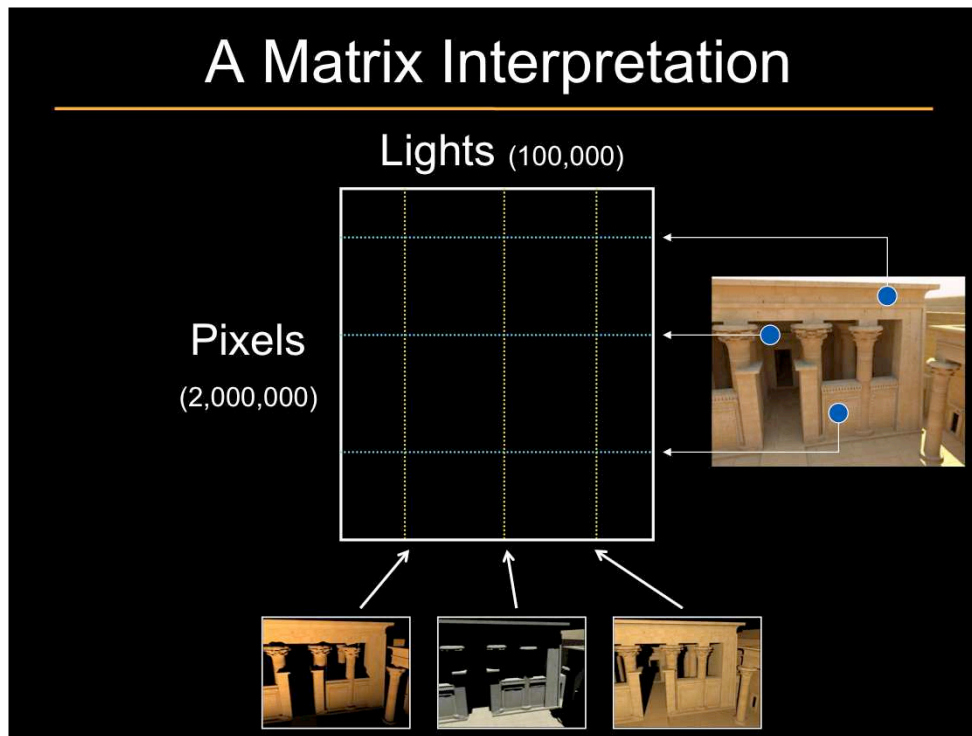
- Alternatives to lightcuts
 - Matrix row-column sampling
 - Visibility clustering
- Potential advantages
 - Shadow mapping instead of ray tracing
 - Simpler to implement
 - No bounds on BRDFs required
 - Faster in occluded environments

In this part of the course, I will talk about several techniques that improve the scalability of many-light methods with the number of virtual lights. In other words – we have reduced our rendering problem to computing the connections between many VPLs and many receivers, or “gather points”.

Of course, one way to do it are the lightcuts algorithms that Bruce described, which are very reliable and high-quality. I will introduce alternative techniques like row-column sampling and visibility clustering, which can have some other advantages.

For example, one may be able to use shadow maps instead of ray casting for visibility checking, which tends to be faster in most cases. No bounds on shaders are required, which can lead to simpler implementations. There may also be advantages in highly occluded environments, where lightcuts will conservatively evaluate illumination assuming full visibility.

A Matrix Interpretation



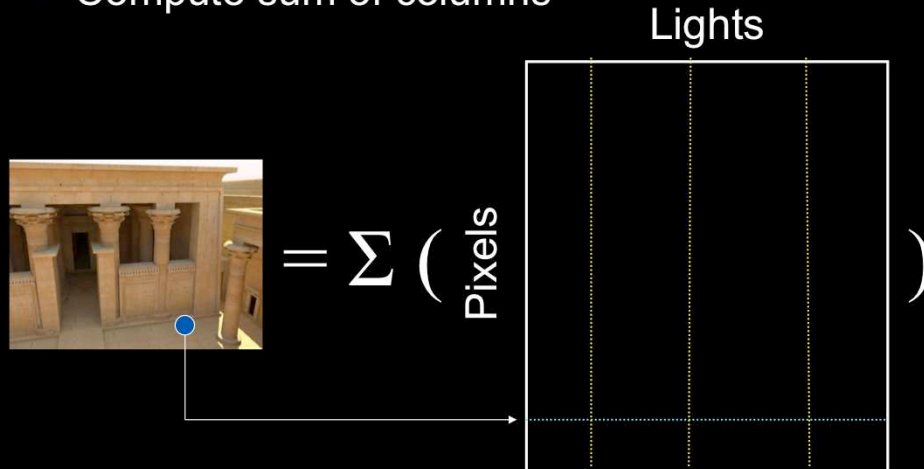
To get started in describing the algorithms, we first need to interpret the many light problem as a matrix of light-pixel interactions. This means that each element of the matrix is the contribution of a single light to a single pixel.

So, the columns of the matrix are really images rendered with a single point light.

The rows represent contributions of all the lights to a particular pixel.

Problem Statement

- Compute sum of columns



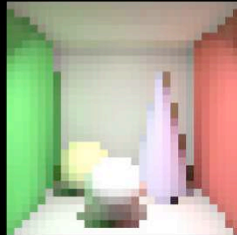
- **Note:** We only have oracle $A(i,j)$

In this setting, the ideal image we would like to render is equal to the sum of the columns of the matrix.

Or, to put it differently, the color of each pixel is equal to the sum of the matrix row corresponding to that pixel.

It is important to note that we're not given the matrix data, we just have an "oracle" – a function that can evaluate the elements on demand. Our goal is to compute the sum without evaluating most of the elements. How is this even possible?

Matrix has structure



A simple scene
30 x 30 image

900 pixels

643 lights



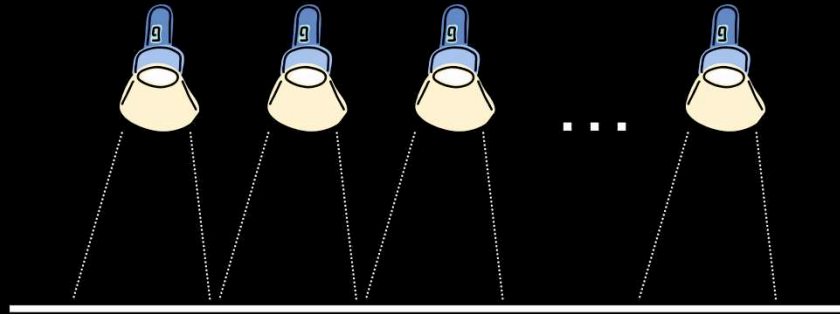
The matrix

The trick is that the matrix is highly structured. Here is an example with a Cornell box, with a single direct light, and Numerically, the matrix often close to low-rank: its columns can often be approximated by linear combinations of other columns.

Therefore, we can get away with computing only a very small subset of the elements, and still gather enough information to render an accurate image.

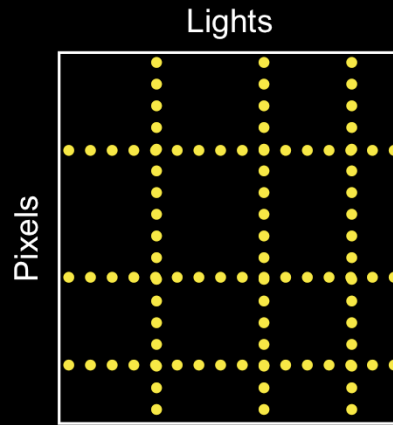
Low Rank Assumption Violation

- Bad case: lights with very local contribution



Of course, the low rank assumption is not always valid – here is an example of a really bad case, where the light's contributions are linearly independent. Fortunately, this does not usually happen in practice.

Sampling Pattern Matters



Point-to-point visibility: Ray-tracing

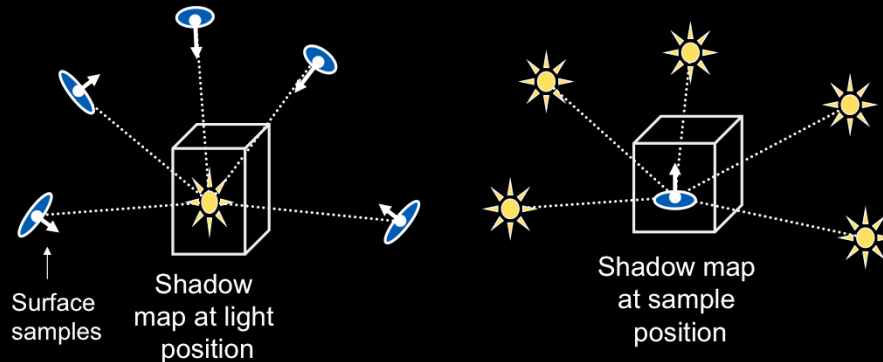
Point-to-many-points visibility: Shadow-mapping

So, we want to sample a subset of the matrix elements, but which ones should we choose?

If we sample complete rows and columns, we can use GPU shadow mapping as our visibility algorithm. This way we can compute elements at very high rate. Furthermore, it is easier to reason about rows and columns, so even if we use a ray-tracer, it may still be an advantage to sample like this.

Row-Column Shadow Duality

- Columns: Regular Shadow Mapping
- Rows: Also Shadow Mapping!

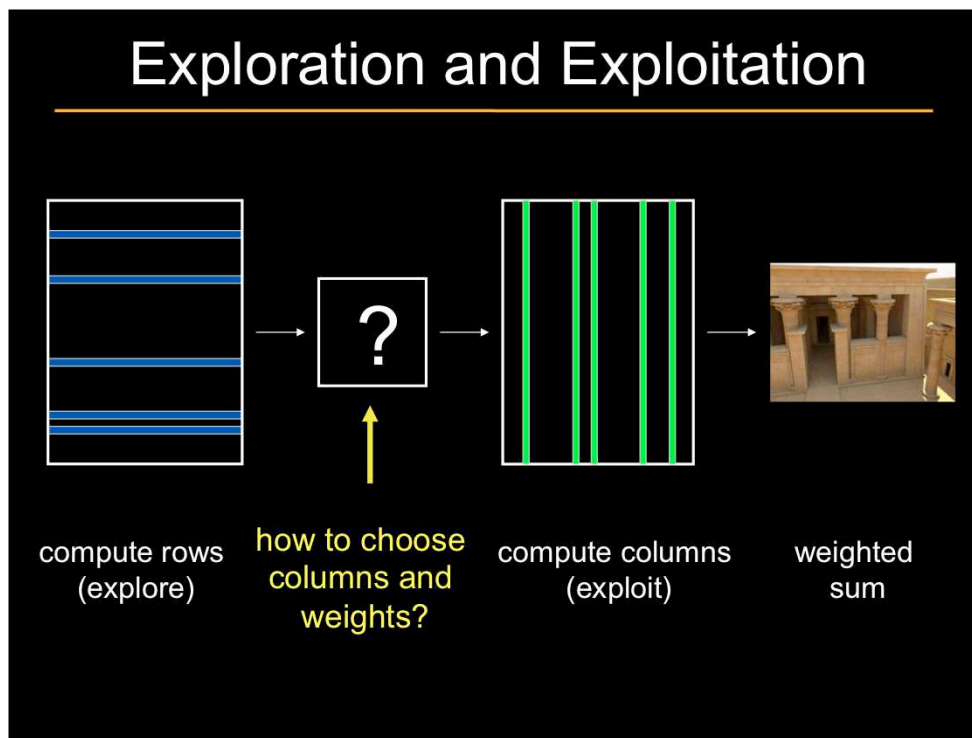


One may wonder how to compute the row contributions using shadow mapping. After all, shadows normally come from the light!

However, shadow mapping is simply a way to determine the visibility from a point to many other points at once.

We can compute a cube shadow map at the VPL position, and determine the visibility of the surface samples, as usual. Or, we can compute the cube at the receiver position, and query the light positions against it.

Exploration and Exploitation



OK, we know how to compute rows and columns. How do we design the rest of the algorithm based on them? We use an idea that combines exploration and exploitation.

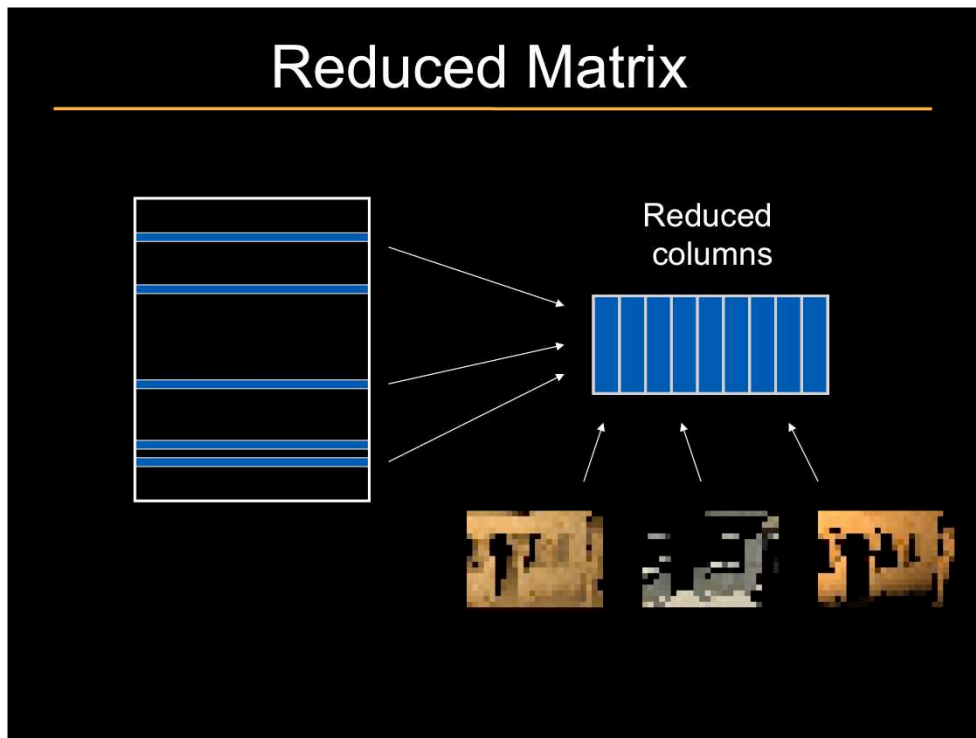
As a first step, we explore the structure of the matrix by computing a small, randomly chosen subset of rows.

Next, we analyze the gathered information, and decide which columns to choose and their appropriate weights.

The exploitation step then computes the selected columns, which are finally accumulated into an image.

The only thing missing is the contents of the black box that analyzes the rows and chooses the columns.

Reduced Matrix



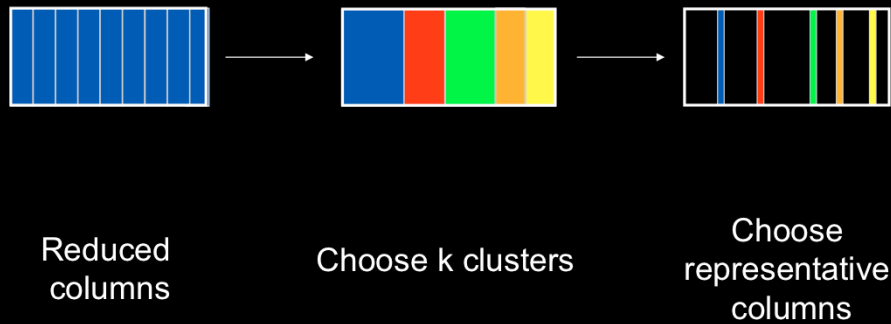
To understand this, let's take the rows that we computed in the exploration stage,

And assemble them into this long, but not very tall reduced matrix.

Now let's flip attention to the columns of this matrix, which we'll call reduced columns.

These can in fact be thought of as tiny images that are sub-sampled versions of the full columns.

Clustering Approach

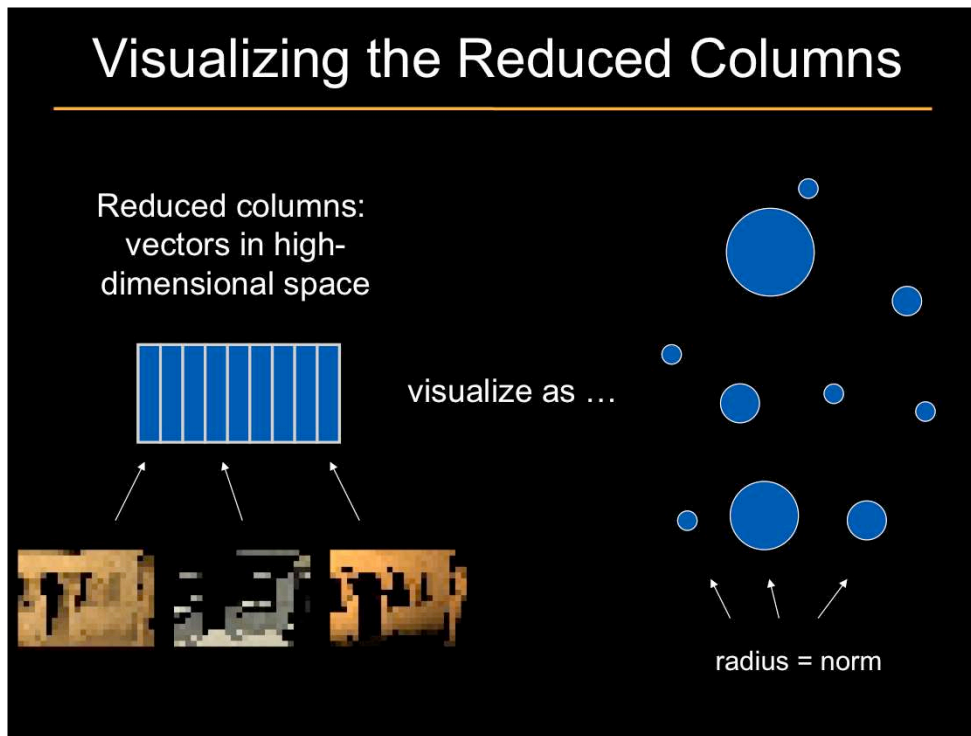


We will use a clustering approach to choosing columns: we cluster similar reduced columns, and we pick a representative in each cluster. Of course, we can arbitrarily re-shuffle the columns – there’s no such thing as “neighboring columns”.

Note that this is actually an unbiased Monte Carlo estimator: each representative, if properly weighted, computed an unbiased estimate of its own cluster, and

The accuracy of the algorithm is highly dependent on the quality of the clustering, so we should design it carefully.

Visualizing the Reduced Columns



First, let's think about the reduced columns as vectors in a high-dimensional space.

We're going to visualize these high-dimensional vectors as circles.

The radius of each circle will correspond to the norm of the reduced column, or equivalently, to the brightness of the little image.

The positions will correspond to the positions of normalized reduced columns in the high-dimensional space.

With a bit of simplification, we can say that circles that are close to each other represent similar lights, and large circles represent lights with strong intensity.

The Clustering Metric

- Minimize:
$$\sum_{p=1, \dots, k} cost(C_p)$$

total cost of all clusters

- where:
$$cost(C) = \sum_{i, j \in C} w_i w_j \|\mathbf{x}_i - \mathbf{x}_j\|^2$$

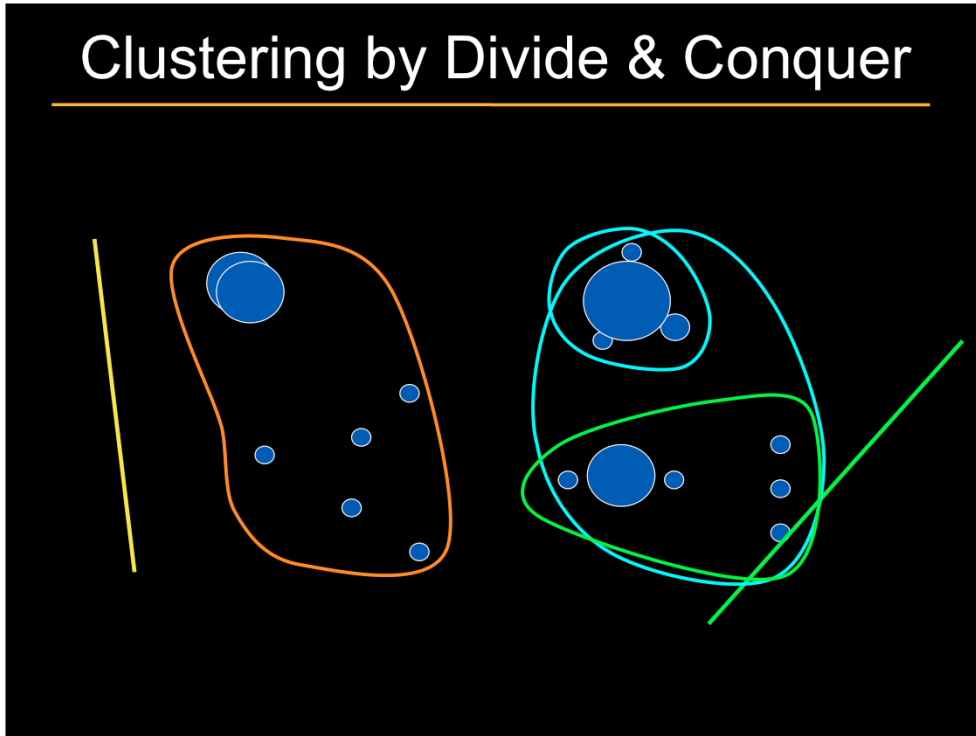
cost of a cluster sum over all pairs in it norms of the reduced columns squared distance between normalized reduced columns

We can prove that the following formula gives the optimal clustering that minimizes the expected error.

So let's look at its meaning. We're minimizing the sum of costs of all clusters, where the cost of a cluster is defined as the sum over all pairs of elements in the cluster of the product of norms and squared distance.

This confirms the intuition that strong lights, or lights that are far from each other, should be in separate clusters.

Clustering by Divide & Conquer

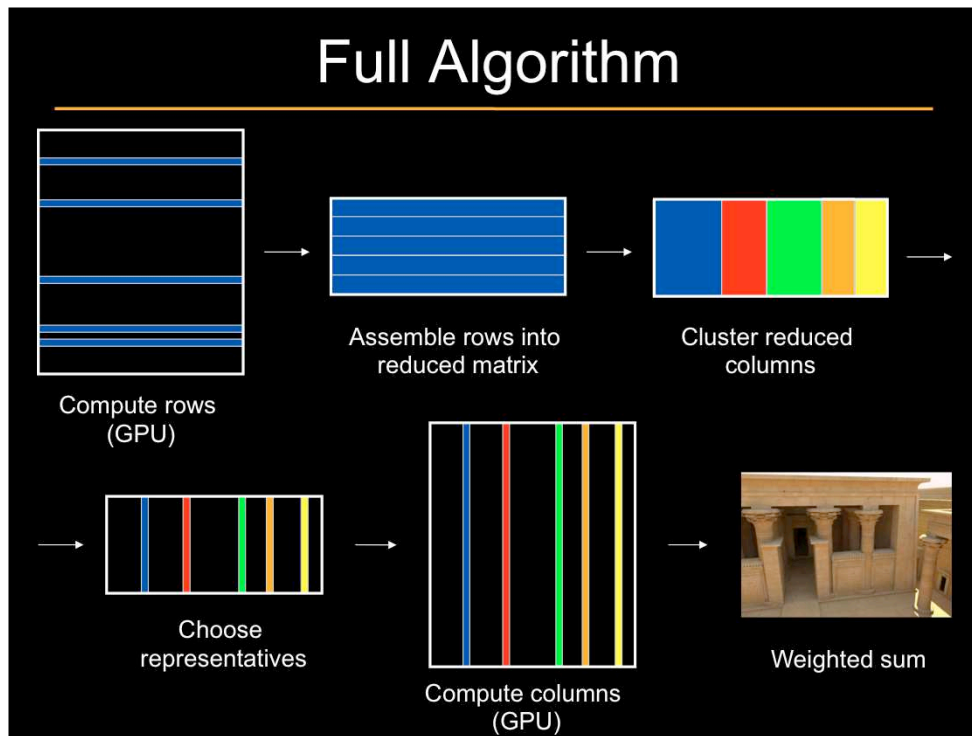


This is an NP-hard problem, but a good approximation can be found by the following divide & conquer algorithm.

We pick a plane with a random orientation. Remember this is in many dimensions, here I'll just visualize it as a line.

Then we move the plane to its optimal position and split the points into two clusters. There are only n possibilities so we can check them all to find the best one.

We then continue this process recursively on the cluster with the currently highest cost.



To summarize, here's the full algorithm:

In the exploration stage, we evaluate some rows on the GPU, then focus on the reduced columns.

We obtain a clustering through the techniques I just described, then pick representatives.

Finally, we accumulate the corresponding columns with the correct weights into an image.

Let's not lose the big picture: all of this is an abstraction that lets us increase the scalability of an underlying many-light algorithm.

Results: Temple

- 2.1m polygons
- Mostly indirect & sky illumination
- Indirect shadows



Our result: 16.9 sec
(300 rows + 900 columns)



Reference: 20 min
(using all 100k lights)

The temple is quite a large scene, not only in terms of the number of polygons, but also its spatial extent. In fact, only a tiny portion of the scene is in view. Most of the illumination in the scene is indirect or sky illumination, with the only pixels lit directly by the sun form the small bright patches on the right.

The method can quite effectively pick the small subset of VPLs that captures the illumination. Many VPLs are either invisible or have a small contribution, and can be aggressively clustered.

Results: Trees and Bunny

- Complex incoherent geometry
- Low rank, not low frequency



Our result: 2.9 sec
(100 rows + 200 columns)



Our result: 3.8 sec
(100 rows + 200 columns)

These scenes are designed to test the algorithm by complex incoherent geometry, and it works quite well.

Approaches based on interpolating illumination across coherent surfaces would have a difficult time rendering these images.

However, our algorithm makes no assumptions about image-space coherence. In this sense, low-rank light transport can be a better approximation than smooth, low-frequency irradiance.

Results: Grand Central

- 1.5m polygons
- Point lights between stone blocks



Our result: 24.2 sec
(588 rows + 1176 columns)



Reference: 44 min
(using all 100k lights)

This scene, the Grand Central station, is a bit of a difficult case. A unique feature of this scene are the omni-directional lights positioned in small recesses between stone blocks.

These lights pose a problem since they violate the low-rank assumption. The columns corresponding to these lights are pretty much linearly independent. Therefore, a larger number of rows and columns will be required here, and some lights are still missing on the left side.

The Value of Exploration



Our result
(432 rows + 864 columns)



No exploration
(Using 1455 lights)

Equal time comparison

An important question is whether the row sampling step of our algorithm is more useful than brute force.

Indeed, instead of creating 100,000 lights and trying to pick a small subset of them, we could simply create a smaller number of lights and render them all.

However, if we compare these images, we find that the simpler approach produces much more objectionable artifacts despite taking a bit longer.

The Value of Exploration



Our result



No exploration

Equal time comparison:
5x difference from reference

Here are difference images as well.

Run MRCS for every frame

- 100 rows, 200 columns per frame
- Flickers too much
 - Randomized, no use of temporal coherence



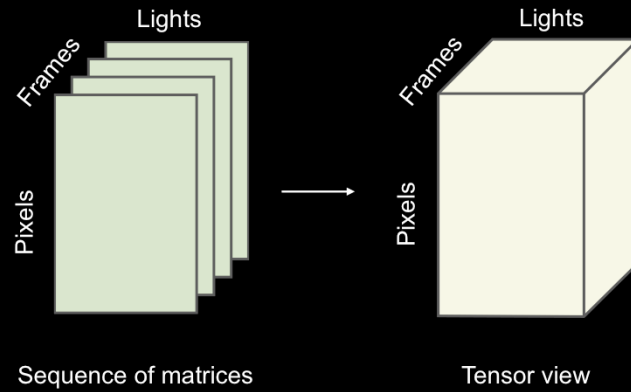
Let's now turn attention to extending row-column sampling to animation.

Here is what we get by running row-column sampling per frame with 100 rows and 200 columns.

There is noticeable flicker, which is not a surprise given that MRCS is a randomized algorithm with no knowledge of the time dimension.

We could fix this by increasing the row and column numbers, but it would be nice to find an algorithm that takes advantage of temporal coherence.

A Tensor Extension

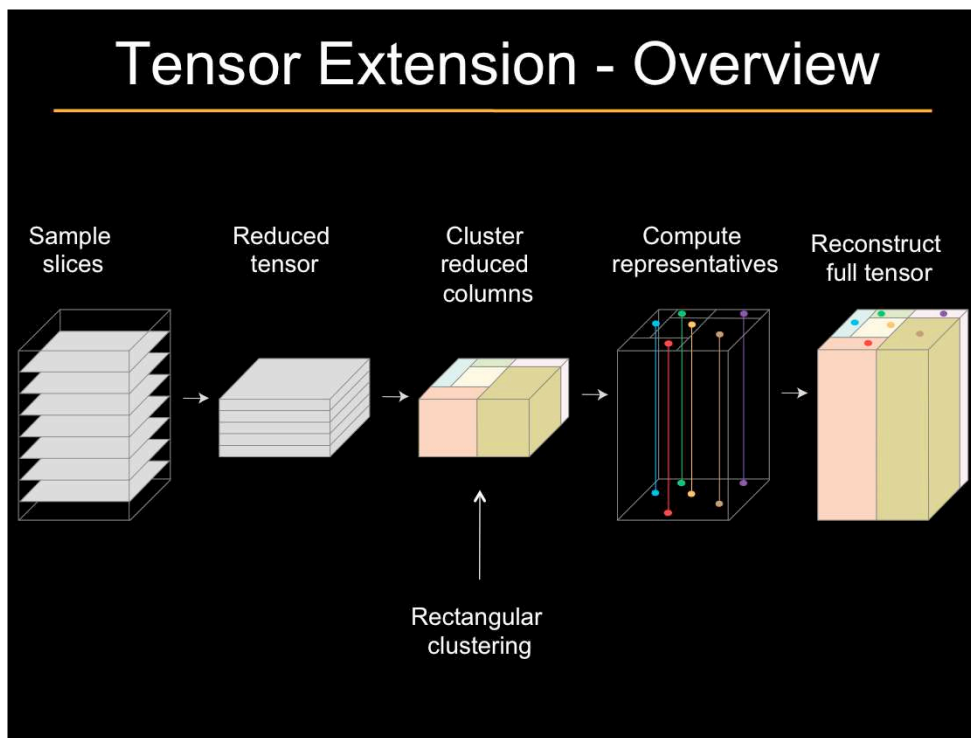


- Size of tensor in our results: 307,200 x 65,536 x 40

In row-column sampling, we had a different matrix in every frame. Let's concatenate them, and consider all of them at once as one large 3D array (or tensor) of lights contributing to pixels over frames.

The amount of data is very large, and we need to avoid constructing the whole tensor at any point in the algorithm, similar to the matrix case.

Tensor Extension - Overview



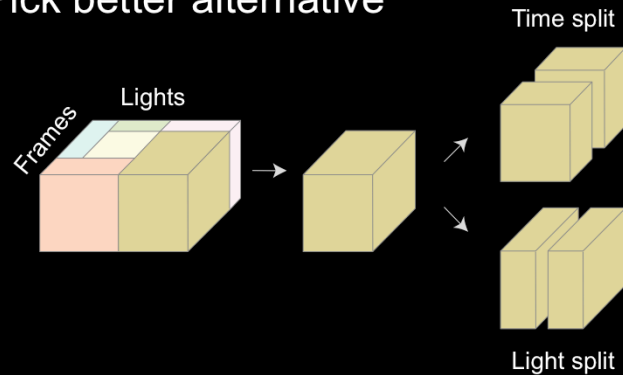
The extension will be done as follows: We compute a set of rows in every frame, resulting in a set of horizontal slices of the tensor.

Then we cluster the columns of this reduced tensor; I call this a “rectangular clustering” since every cluster is a cartesian product of a light subset and a frame subset.

We can then use a similar representative selection and reconstruction as before, except here we have to be careful, because pixels move between frames. A pixel mapping trick similar to optical can be used to warp the representatives to avoid this problem.

Splitting a cluster

- Pick cluster with highest cost
- Try splitting in time
- Try splitting in lights
- Pick better alternative



How do we find a rectangular clustering? This problem is again NP-hard, but we can adapt the divide-and-conquer as follows.

We will start with all light-frame pairs in one cluster, and keep splitting the cluster with highest cost until we reach the desired number of clusters.

We will try splitting the cluster in time and in lights, and pick the split that gives us the better objective function value.

Now the question is, how to split in time and in lights; we can answer this by trying both splits and seeing which one decreases the objective function more.

Results - Iris

- 51k triangles, 65,536 lights
- Deforming objects, high-frequency shadows
- 6.9 sec / frame (brute-force: 2 min / frame)



Here is an example animation rendered with this method. On the left is the result, and on the right is reference with all VPLs, which took about 9 times longer.

Results - Temple

- 2.1m triangles, 65,536 lights
- Sun & sky lighting, moving sun
- Multiple indirect light bounces
- 26 sec / frame (brute-force: 33.5 min / frame)

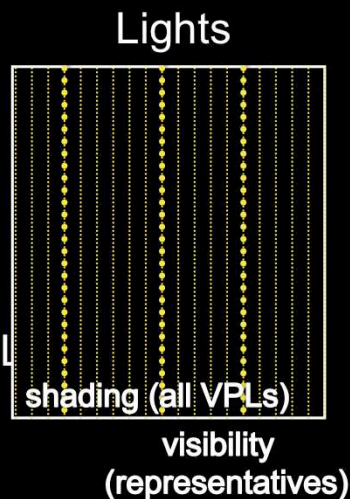


And here is another example with a temple and a moving sun. This one is about 77x faster than the brute-force solution.

Visibility Clustering

27

- Many VPLs needed for shading
 - Shading is cheap → shade from all VPLs
- Cannot afford visibility for every VPL
- Key idea:



Separate shading from visibility

Next, I will introduce an interesting variation of the row-column approach that is some cases more robust.

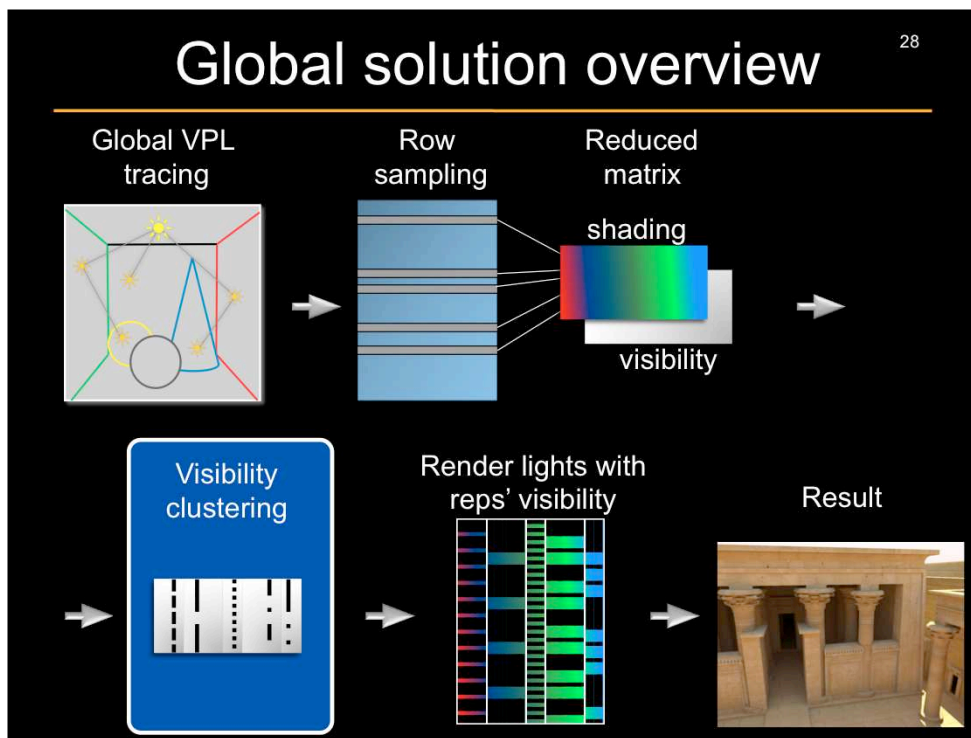
The design of the algorithm is motivated by several observation on shading and visibility in glossy scenes.

First, we observe that in glossy scenes we cannot easily pick a small subset lights to approximate the solution, because shading from the individual lights is quite different. In other words, we want to compute the shading from all the global VPLs. And that's actually doable because the GPU is extremely efficient at computing the shading.

But we just cannot afford to evaluate a SM for each of those 200k lights! So the key insight is to separate shading from visibility, use ALL the light for shading, and only a small number of representative lights for visibility.

Global solution overview

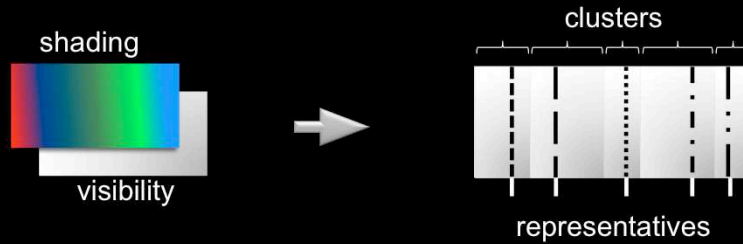
28



Here's the overview of visibility clustering. We start by distributing the global VPLs in the scene by tracing particles from light sources. After that, we sample a number of rows of the interaction matrix, which means that we pick a number of pixels and for each of them, we evaluate the shading and visibility for all the lights. That gives us the reduced shading and visibility matrices. The visibility clustering then analyzes these matrices and yields clusters of lights that will share the same shadow map. After that, we render all the VPLs with the shadow map of the representative light for each cluster. This yields the complete global solution. There's only one bit that remains to be explained here, and that's the visibility clustering algorithm.

Visibility clustering

29



- Clustering algorithm
 - Hierarchical splitting
 - Minimize the clustering cost
 - L2 error of reduced matrix due to visibility approximation

The goal of the visibility clustering algorithm is to group VPLs into clusters that will share the shadow map of its representative VPL. We use a data-driven approach where we analyze the row samples from the light interaction matrix.

The clustering algorithm proceeds in a top-down fashion, splitting clusters with highest cost, which is defined as the L2 error of the matrix incurred by the visibility approximation.

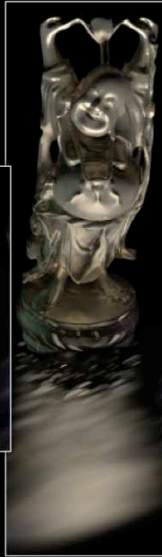
Visibility clustering result

30

Matrix row-
column
sampling



10k shadow maps
10k shading lights



Our visibility
clustering

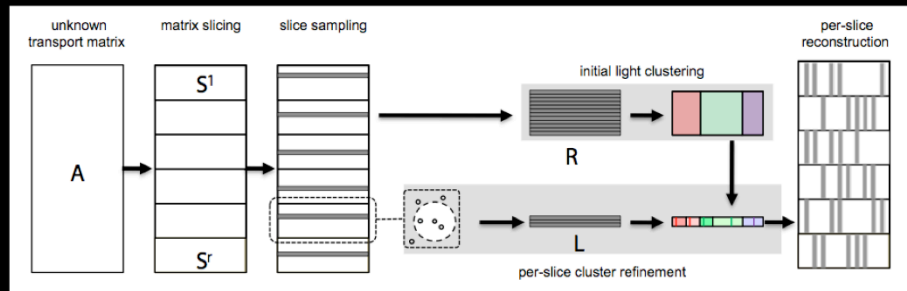


5k shadow maps
200k shading lights

Here's a comparison of the result of our visibility clustering with MRCS in the same time for a simple scene of the happy Buddha statue reflected in a glossy plane. The MRCS result uses 10k representative lights (i.e. 10k shadow maps and 10k lights for shading), which leaves visible splotches in the image. The visibility clustering, in the other hand, uses only 5k shadow maps for visibility but all the 200k lights for shading, substantially improving the smoothness of the reflections.

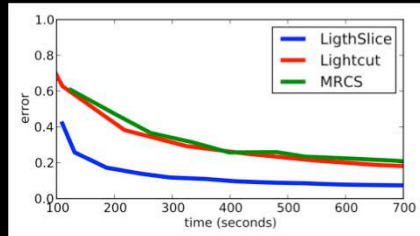
LightSlice (Ou and Pellacini)

- Compute standard clustering
- Refine it differently in different “slices”
- Use neighboring slices to get more rows



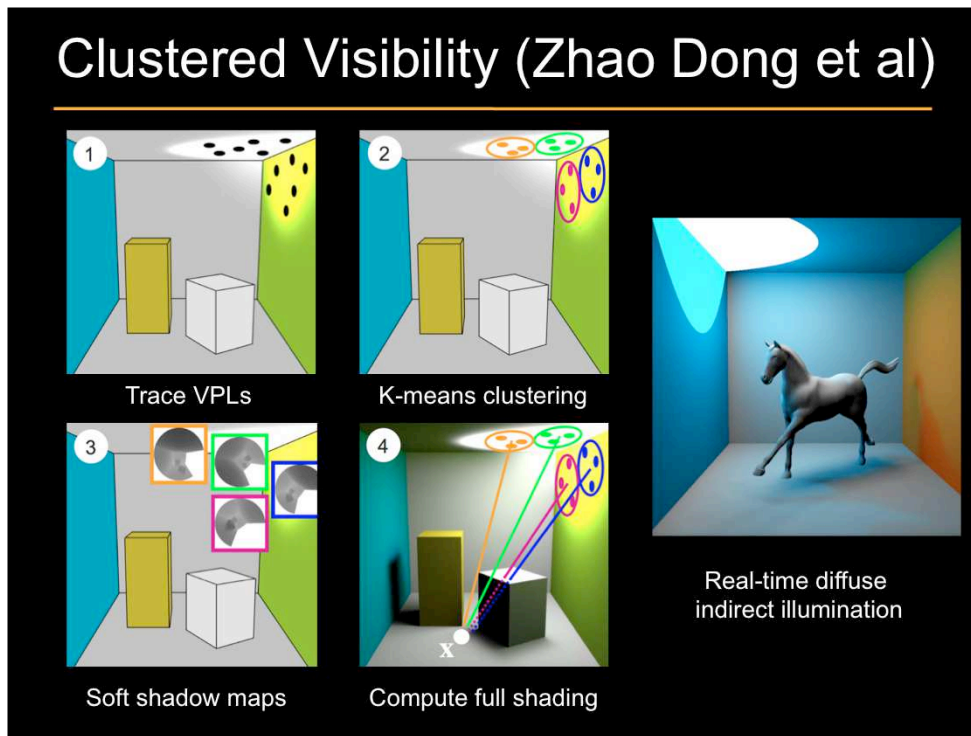
Another nice idea, published recently in the LightSlice paper, is to refine the original clustering within “slices” of the image. One problem is that each slice will only have one row sample, and one-dimensional data is not enough to run a clustering algorithm. This is addressed by using neighboring row samples to help.

LightSlice: Results



This works very well - in fact, on this fairly complex scene, the algorithm seems to outperform both lightcuts and row-column sampling.

Clustered Visibility (Zhao Dong et al)



Another alternative approach I should mention uses no matrix formulations, but instead uses a simple visibility clustering idea that nevertheless works fine in simple scenes and can even achieve real-time performance.

The idea is to use k-means clustering for visibility, and render full shading. One may think that this for highly variable VPL intensities, but for a single bounce, and one lightsource, one can easily make all VPLs same intensity. The approach also uses soft shadow maps, so it can get away with pretty small numbers of shadow maps.

Open Problems

- How many rows + columns?
 - Pick automatically
- Row / column alternation
- Progressive algorithm:
 - stop when user likes the image
- Comparison to matrix completion

How to pick the number of rows and columns automatically?

It would be nice to design a variation of the algorithm that alternates row and column sampling, since knowing some columns might tell us which rows to sample, but no one seems to know how to do it.

It might also be useful to make the algorithm progressive, so the user can stop the evaluation when they are satisfied with the image.

We are also interested in developing a temporal version of the technique that renders multiple frames at once. The problem is – how to do it so that the representatives do not have to be kept in memory?

Finally, there are some other approaches for matrix completion in the literature, but we tried them for rendering and they did not turn out to be better. However, there might be ways to improve them, they just haven't been studied systematically.

Real-time many-light rendering

Carsten Dachsbacher

Real-time Many-light Rendering

Carsten Dachsbacher

Real-time Many-light Rendering



Outline

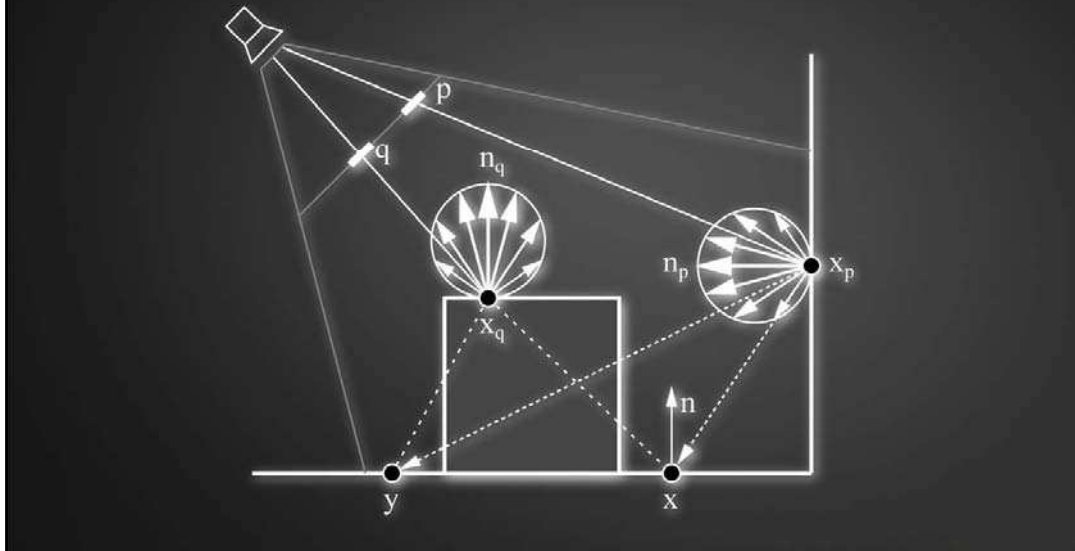
- ▶ main difference to „offline“ methods: visibility
 - ▶ rasterization instead of raycasting
 - ▶ focus on surface lighting here
 - ▶ real-time constraints ↔ mostly diffuse scenes
- ▶ creating VPLs with rasterization/GPUs
- ▶ shading and shadowing from VPLs
 - ▶ imperfect shadow maps, micro-rendering
- ▶ bias compensation in screen-space

Reflective Shadow Map



From-Point Visibility

- ▶ one-bounce illumination is caused by surfaces visible in the shadow map
- ▶ extended shadow map contains required information



let's first start with single-bounce indirect illumination, which essentially means placing VPLs only at surfaces directly visible from the light sources

these surfaces can simply be rasterized, actually these are the surfaces one would store in a shadow map

Reflective Shadow Map



From-Point Visibility with Additional Information

- ▶ store for every pixel
 - ▶ depth value
 - ▶ world space position
 - ▶ normal
 - ▶ reflected radiant flux



In addition to the depth value such an extended shadow map – reflective shadow map – stores the world space position, the surface normal and the reflected radiant flux for every pixel.

The world space position could be reconstructed from light source positioning and the depth value - storing it is just a trade-off between computation overhead and memory consumption.

So with this data we know everything about the surface to compute single-bounce indirect lighting.

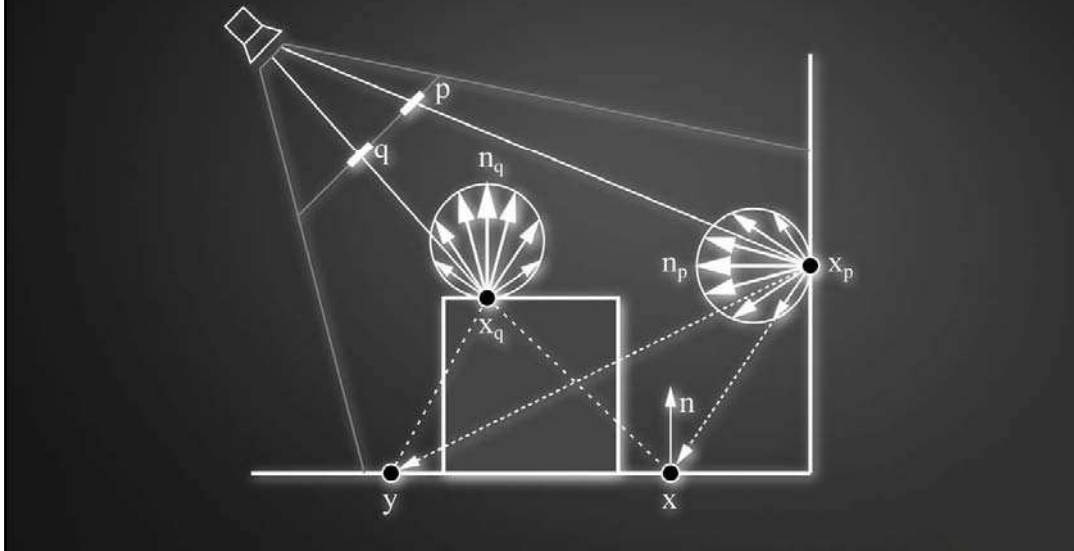
Now the flux defines the brightness and color of the surface and the normal its spatial emission characteristics.

Reflective Shadow Maps



From-Point Visibility with Additional Information

- ▶ each pixel is a virtual light source
- ▶ indirect irradiance from pixel lights



The total indirect irradiance at a surface point could be approximated by summing up the illumination due to all pixel lights.

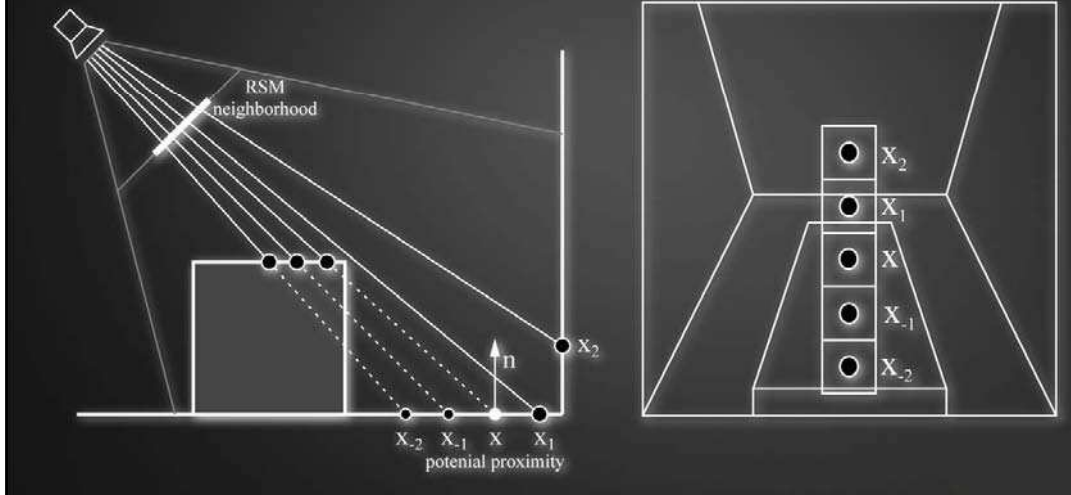
If we do not do any additional work, and just use the RSM as is, we cannot consider occlusion for the indirect light sources and e.g. the floor is wrongly lit on the left side of the box.

Reflective Shadow Maps



Light Gathering

- ▶ sample potentially lights
- ▶ pixels close in world space are close in RSM (not vice versa)



The classic RSM method performs a gathering approach computing the sum of pixel light contributions for each pixel in the image by sampling the extended shadow map.

As the number of pixels is typically too large, we reduce the sum to a restricted number of a few hundred samples, concentrating on the most relevant pixel lights.

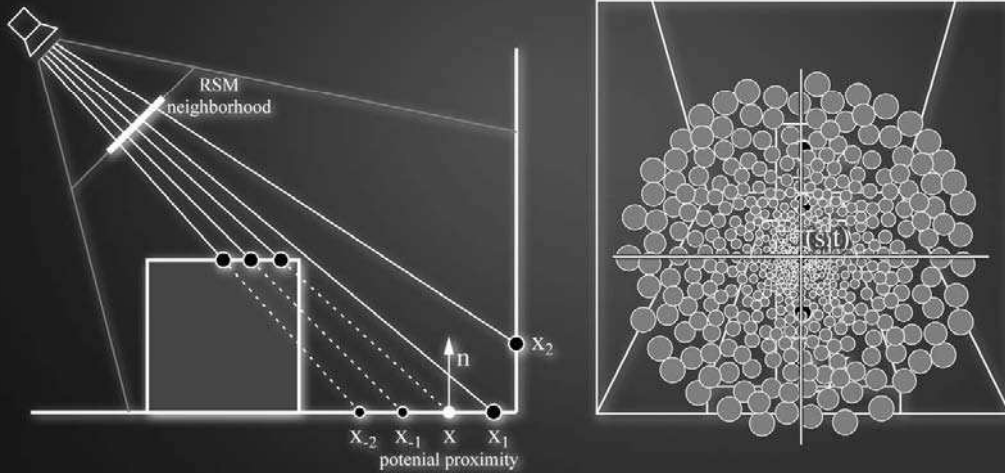
In general those pixel lights are relevant which are close in world space and of course they are also close in the shadow map projection – in reverse this is of course not true.

Reflective Shadow Maps



Light Gathering

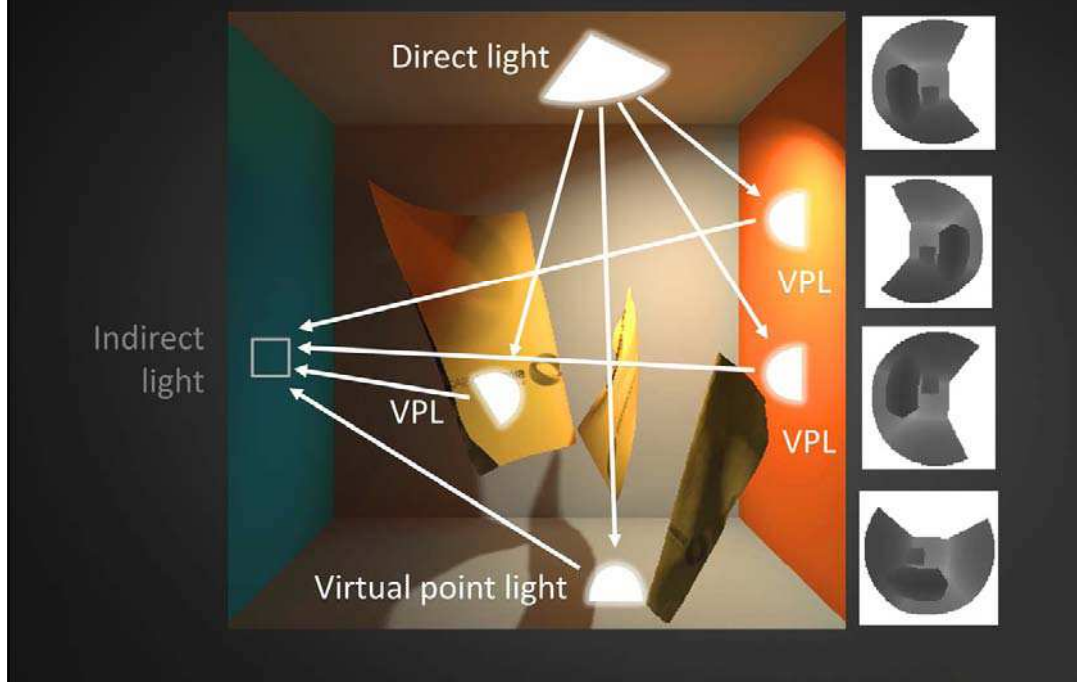
- ▶ sample potentially lights
- ▶ pixels close in world space are close in RSM (not vice versa)
- ▶ for classic instant radiosity: pick one set of VPLs for all surfaces



For this, we precompute a sampling pattern for selecting pixel lights which gets centered around the projection of the surface point to be lit.

By this, we reduce the number of samples to a few hundred per pixel.

Instant Radiosity



But we obviously we are lacking two aspects: multiple bounces of indirect light and shadowing of indirect light.

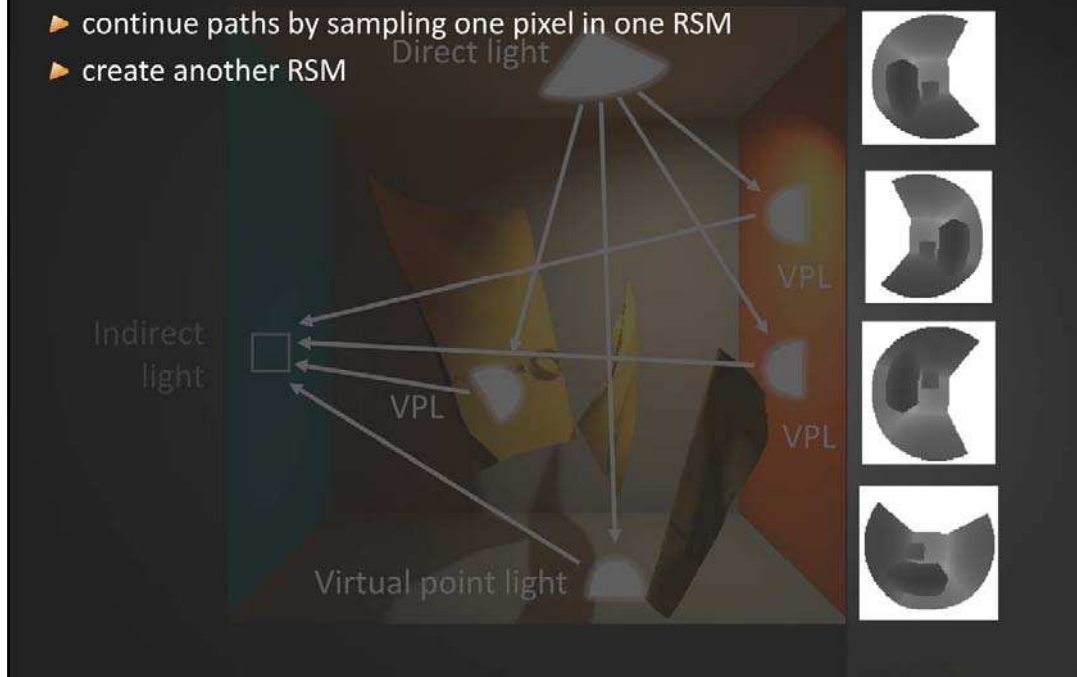
The first problem can be dealt with relatively easily: as in standard random walk...

Instant Radiosity



Multiple Bounces

- ▶ continue paths by sampling one pixel in one RSM
- ▶ create another RSM



we can pick a random pixel, which maps to a direction, in every such RSM, take the surface location and orientation from there, and recursively continue rendering RSMs.

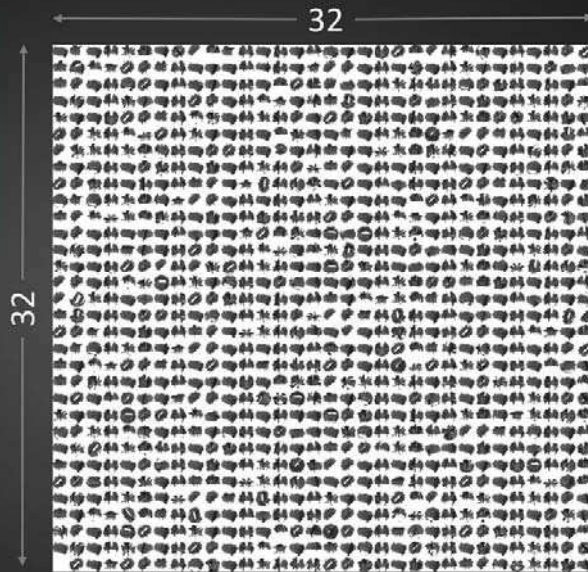


Visibility is a problem, as we typically have a large number of VPLs and thus shadow maps. Even in diffuse scenes, 1000 or more VPLs is common.

Instant Radiosity bottleneck



- ▶ 1024 VPLs
- ▶ 100k 3D model
- ▶ 32x32 depth map
- ▶ ~300M transforms
- ▶ 100x overdraw



In fact, this shadow map generation is the bottleneck:

Assuming we use 1024 VPLs and a 100k triangle 3d-model.

First, vertex processing requires to do 300M vertex transforms when drawing all triangles into all shadow maps.

Second, drawing 100k tris into a 1k texel depth map is a hundredfold overdraw.

Let's find something better ...

Imperfect Shadow Maps



Fast and Approximate Shadow Maps

- ▶ observation: low quality (imperfect) depth maps sufficient when using many VPLs form smooth lighting
- ▶ idea: point-based rendering is great for approximate renderings
- ▶ main steps of the algorithm
 - ▶ point-based depth maps
 - ▶ pull-push to fill holes

What we propose as a solution is based on the observation, that **low quality depth maps are sufficient**.

This is, because the **individual contribution** of every VPL is only **small**.

Also indirect lighting **varies smoothly** in most scenes or at least in mostly diffuse scenes.

Our contribution is to allow imperfection when creating a depth maps that allow for a much more efficient generation.

And this is achieved by using a point-based rendering approach which is typically well suited when you need LOD and approximate rendering.

The main steps that I will outline here are:

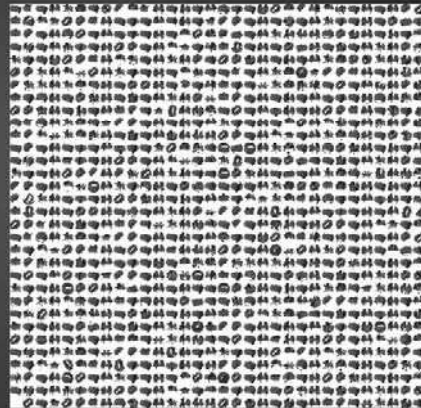
1. Point-based depth map generation
2. A pull-push operation to fill holes from point rendering

Imperfect Shadow Maps



Point-based Depth Maps

- ▶ goal: Fill ~1000 depth maps for every frame?
 - ▶ classic approach takes hundreds of milliseconds for “Sponza”
 - ▶ as correct as possible
 - ▶ using only little bandwidth
- ▶ solution:
 - ▶ use points (no connectivity)
 - ▶ roughly as many as there are pixels



So the goal is to generate as many depth maps as possible, but each of them is allowed to be of a low resolution, say 32x32 pixels.

Using classic depth maps for this, takes hundreds of milliseconds for the Sponza scene.

We use point rendering, because point representations decouple from the input geometry, and they can easily rendered into a single render target for multiple views. Also LOD for points is very simpler, because they don't require connectivity.

Imperfect Shadow Maps



Point-based Depth Maps

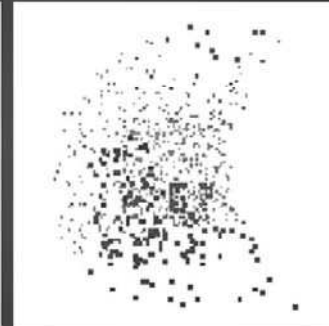
- ▶ point-representation: no connectivity, simple LOD



standard rasterization



imperfect (point-based)



imperfect
(smaller, less points)

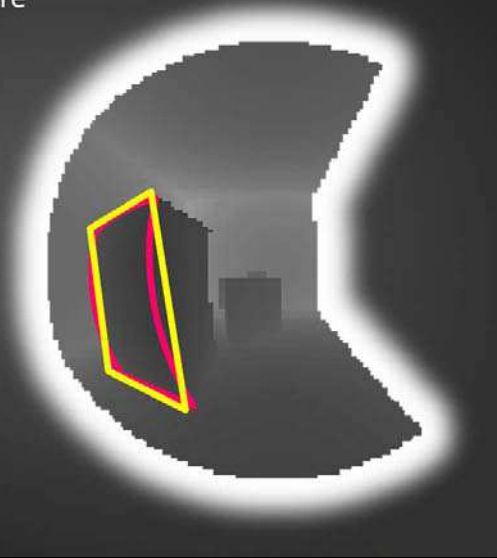
Here you can see a comparison...

Imperfect Shadow Maps



Point-based Depth Maps

- ▶ point-representation: no connectivity, simple LOD
- ▶ paraboloid parameterization
 - ▶ one projection for entire hemisphere
- ▶ no tessellation problem as with triangles



As surface-VPLs emit into the **positive hemisphere**, we chose a paraboloid parameterization of directions.

Points fit this well, as lines map to curves when using this parameterization which can cause triangle meshes to be problematic.

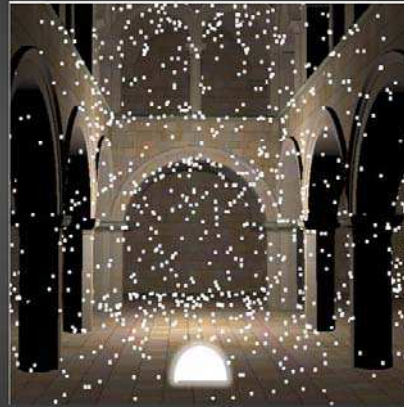
Further **benefits** of paraboloid shadow maps are: single-pass/projection for hemisphere, relatively low distortion.

Imperfect Shadow Maps



Point-based Depth Maps

- ▶ point-representation: no connectivity, simple LOD
 - ▶ generated in a pre-process
 - ▶ ~8k points for every VPL
 - ▶ different set for every VPLs



The point-representation is precomputed, each VPL has its own randomly created point set (to avoid banding) of about 8k points.

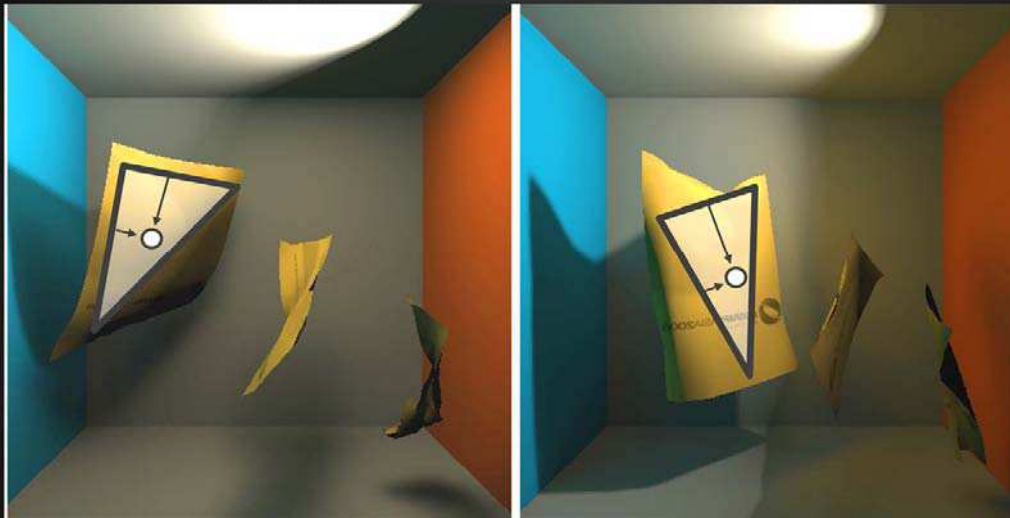
The image shows one VPL and its point set.

Imperfect Shadow Maps



Frame t

Frame $t+1$



- ▶ store points with barycentric coordinates and triangle ID

Instead of storing an x - y - z -position for every point, we store it as a generalized barycentric coordinate instead, that is: a triangle index and a barycentric coord inside this triangle.

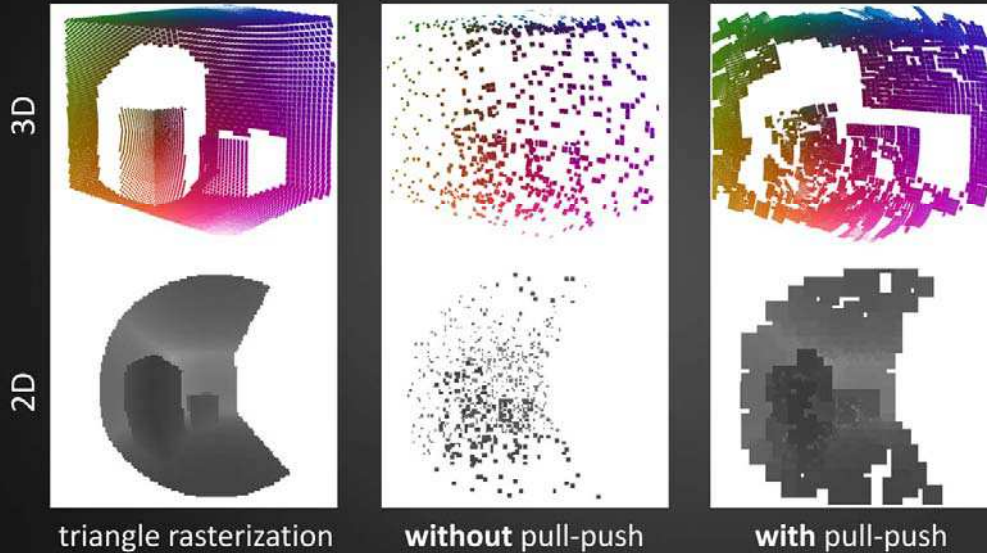
By doing so, the points can naturally follow the deformation of the mesh the approximate.

Imperfect Shadow Maps



Pull-Push Steps

- ▶ depth maps from points have holes



While this is a classic depth map, drawing a low number of points leads to holes, such as in this example.

We fill those holes using a pull-push step, **similar to Grossman/Dally, to fill holes.**

To this end, we build a pyramid of depth values, where we average only valid depth values in a pull step.

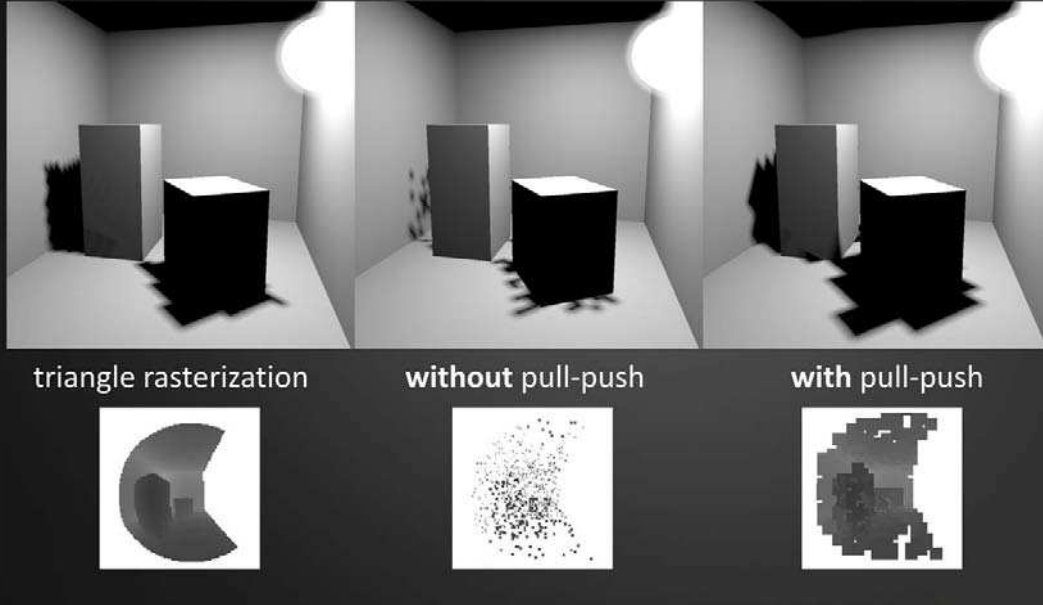
In the subsequent push step, every undefined pixel at every level, is replaced by the average of the defined pixels.

Imperfect Shadow Maps



Pull-Push Steps

- ▶ depth maps from points have holes



Here we show the imperfect shadow of an individual VPL.

A depth map without pull-push will have **light leaks**, that are **fixed** by pull-push.

However, pull-push sometimes miss-classifies depth values, and gives **false positive** or **false negatives**.

Small holes might be „real“ holes and small objects could be isolated, pixel-sized floating objects.

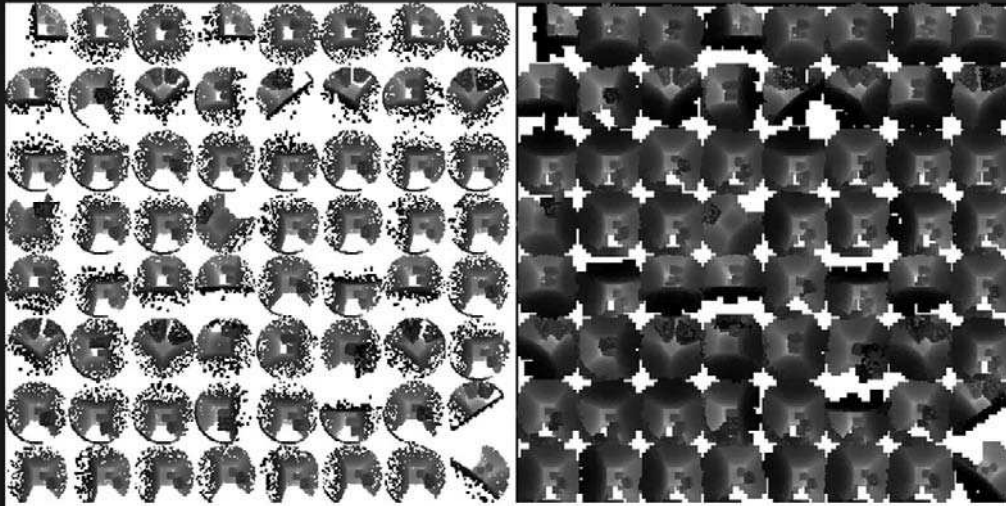
If such miss-classification happens, the only negative result is, that such objects will not cast shadow or let light through, which is not very important, because they are **small**; we have **many VPLs**; and such errors are **uncorrelated**.

Imperfect Shadow Maps



Pull-Push Steps

- ▶ ... on all depth maps in parallel



without pull-push

with pull-push

We do this pull-push step on **all depth maps in parallel**.

As we work in depth map space now, we are now **independent of the geometric complexity**: It is irrelevant if this is an image of a Cornell Box or an XYZ dragon: it's just an image.

Imperfect Shadow Maps: Results



cornell box
horse



20.9 fps, 1280, 8k

sponza



11.8 fps, 2560, 8k

multiple
bounces



complex, local
area lights



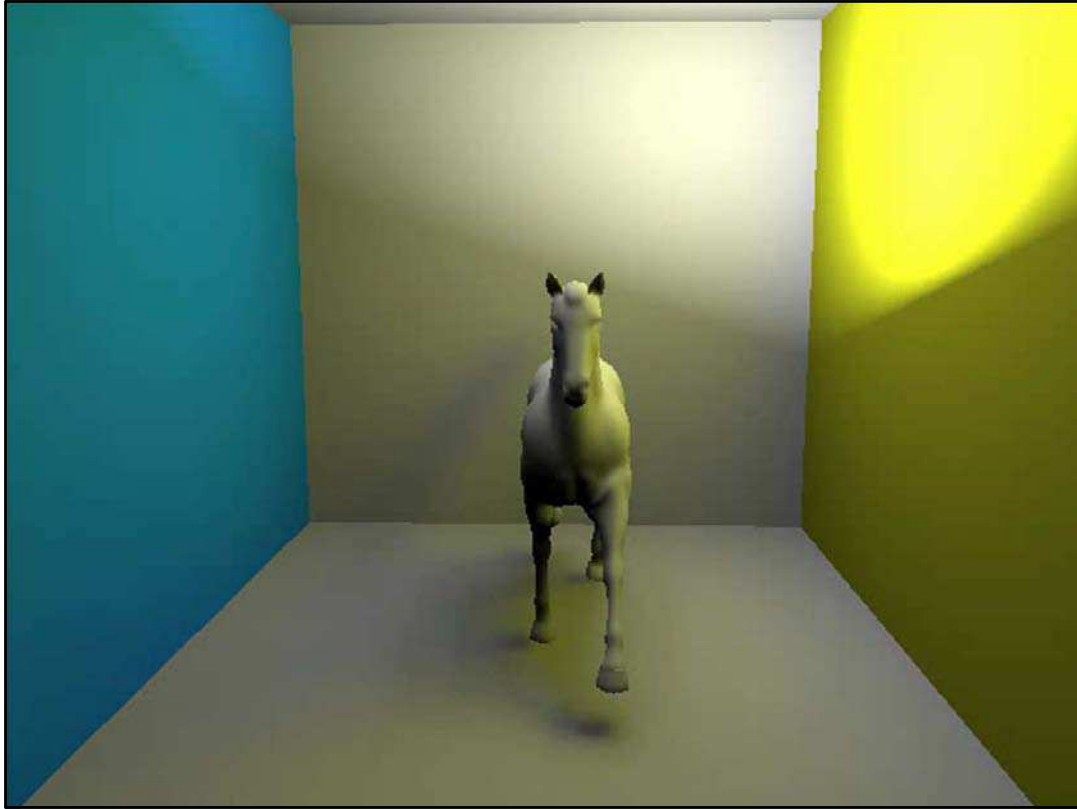
natural
illumination



caustics



Let me now present some of our results, which range from diffuse bounces in a Cornell box to complex scenes, including multiple bounces, arbitrary local area lights, natural illumination to caustics.



Here are **animated meshes** inside the Cornell box with a dynamic direct light.

Most of the **light** in this scene **is indirect**.

Note, how the animals feet cast high-frequency shadows, whereas the animal itself casts a correct soft shadow.

Also note, the subtle **variations in shadow color**.

Despite the fundamental changes, in indirect lighting there is **no flickering**.

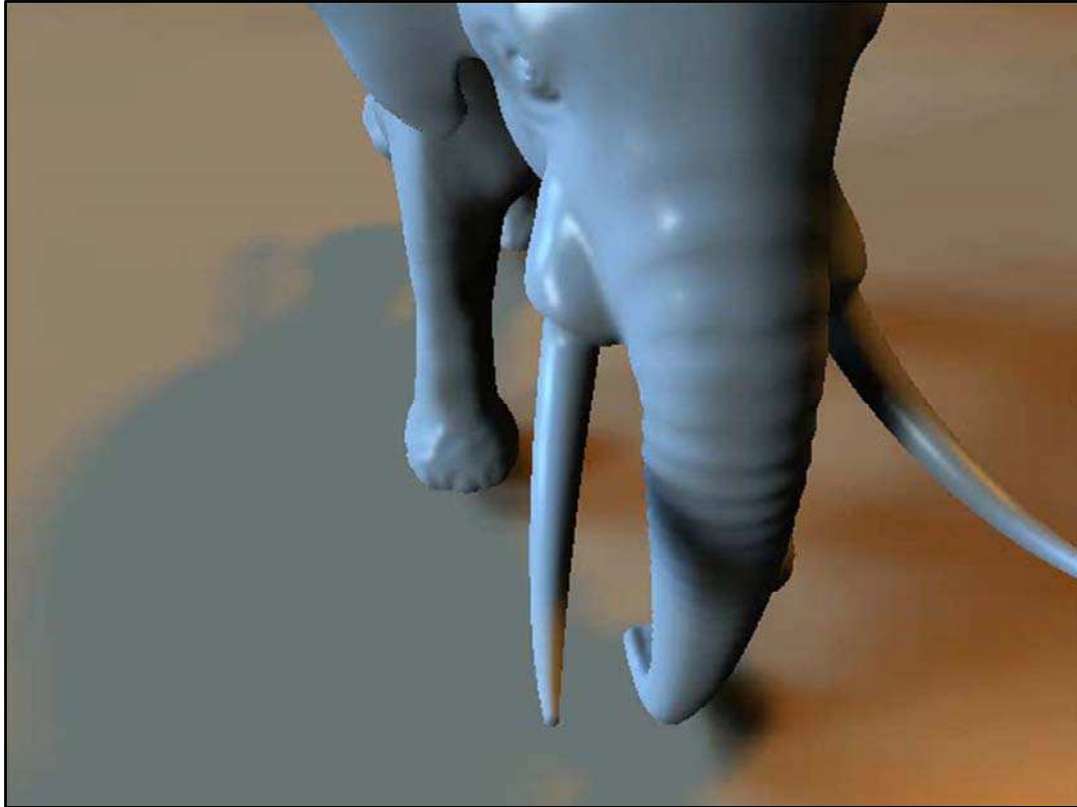


This is a **more complex scene**, where a cloth deformation is placed inside the well known Sponza model.

To achieve sufficient **temporal coherence**, we need **1024 VPLs** in this example.

Note, how the bounced light color **changes drastically** when the cloth is moving.

Note, the **indirect shadow** from the columns.



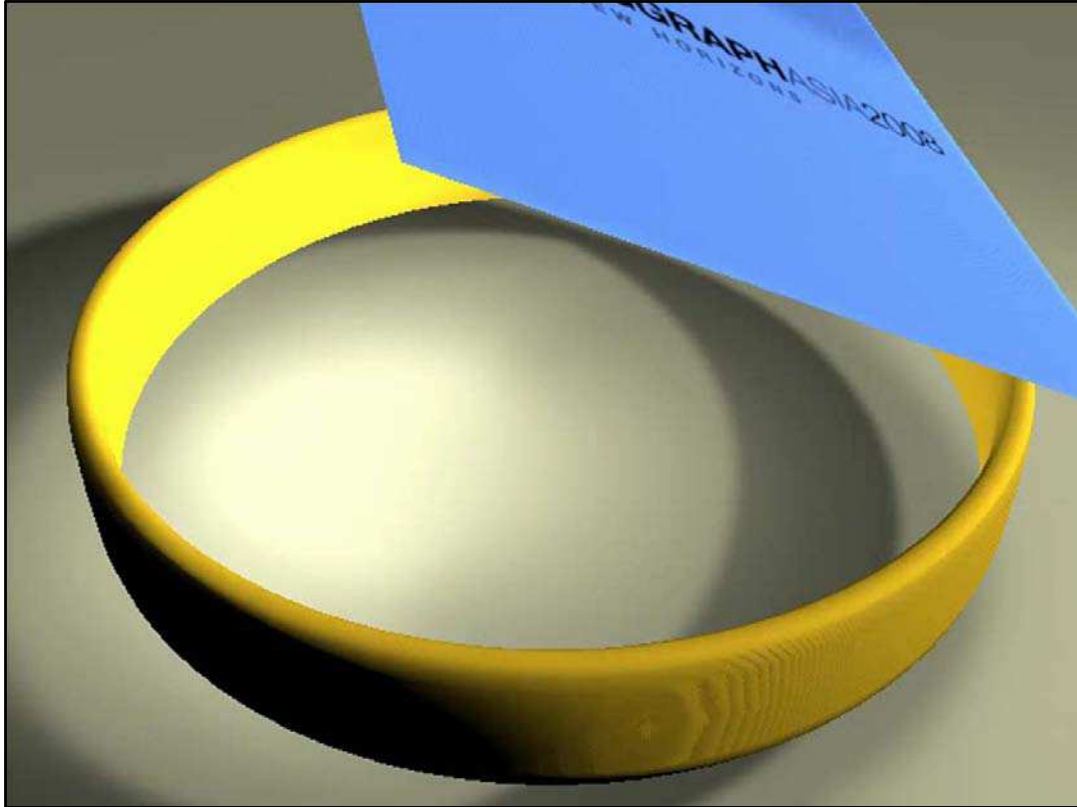
In this example, we did not use global illumination, but **direct natural illumination from an environment map.**

Distant light sources are placed on an environment map, with an **orthographic shadow** for each.



In a similar way, we can generate **local soft shadows from complex area lights, with varying color.**

We **discretize** the area light into several point lights and compute and ISMs for each.



Finally, we are **not limited to Lambertian VPLs**.

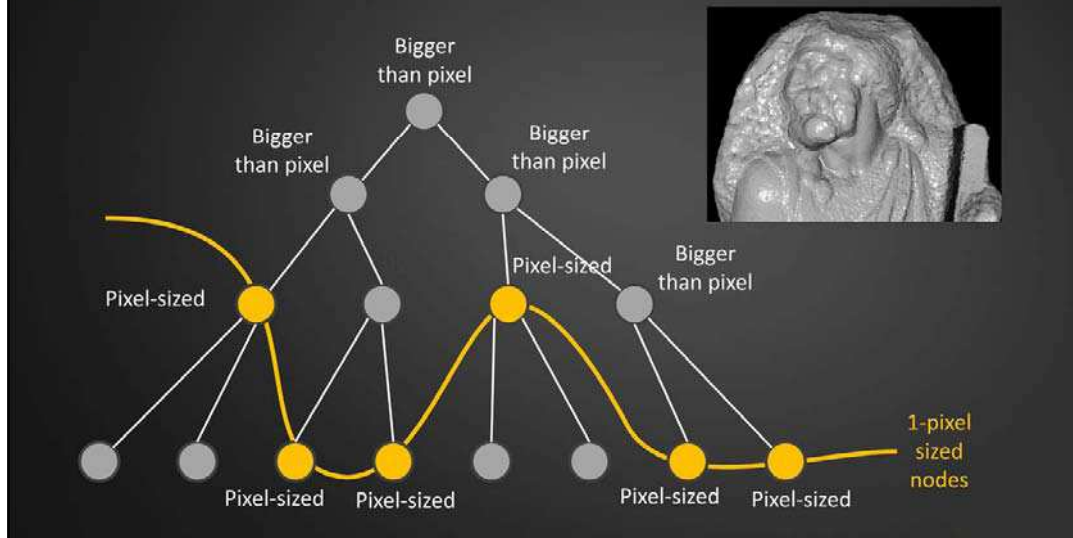
Here, we use a Phong distribution, to compute **one glossy bounce** with full **indirect visibility** which is **usually ignored**.

Micro-Rendering



Motivation

- ▶ imperfect shadow maps use low-quality point-based rendering
- ▶ methods with quality comparable to triangle-based rendering exist
 - ▶ hierarchy of point samples, e.g. Q-Splat



ISMs are based on quickly creating shadow maps from a point-based representation, but point-based rendering is not necessarily of low quality or exhibiting holes.

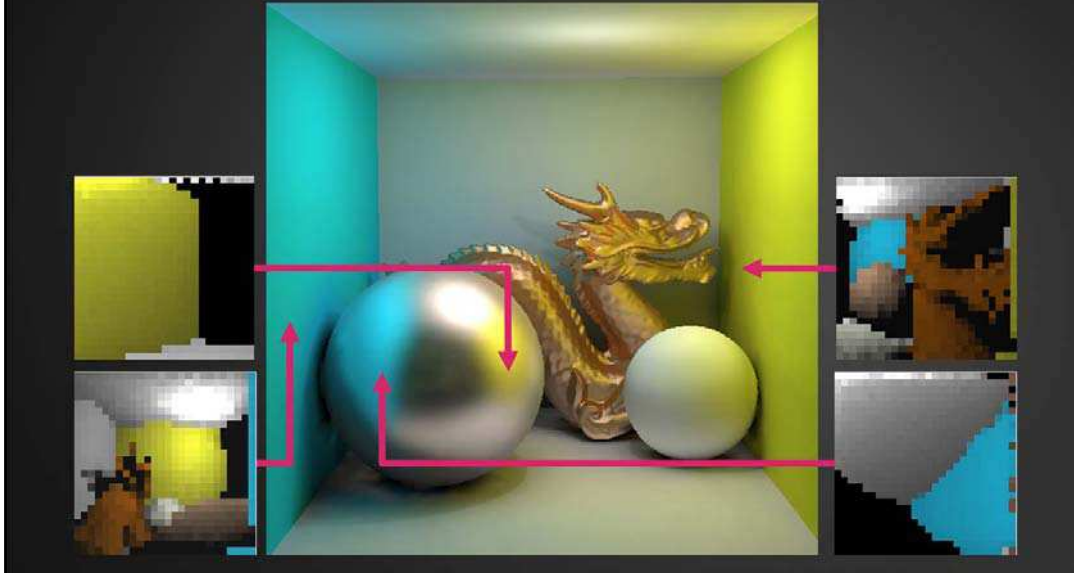
Typically points are organized in a hierarchy to allow for choosing the required level of detail on the fly. The classic example is Qsplats where the refinement criterion is the size of a point primitive on the screen.

Micro-Rendering



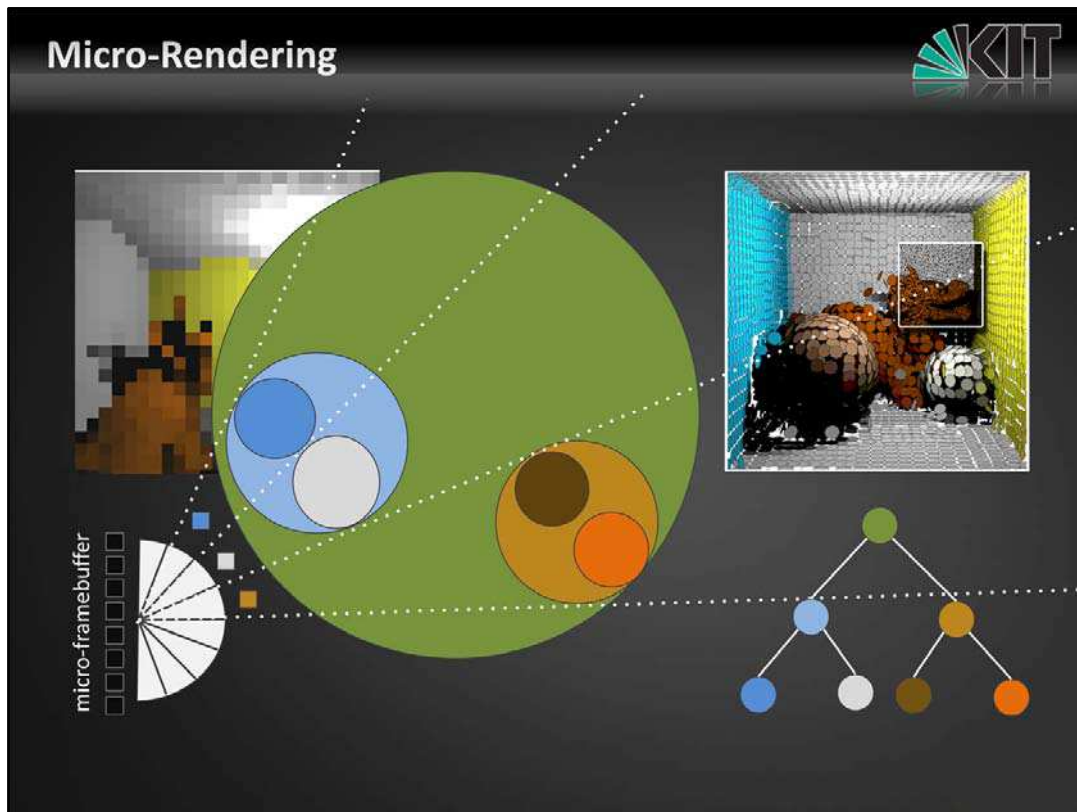
Motivation

- ▶ render high-quality environment maps for gathering incident light
- ▶ can also be used for “flawless” shadow maps or gathering from VPLs



The idea of micro-rendering was initially to do final gathering – we will see how this is related to many-lights rendering in a minute – by creating many small renderings of the incident light at many surface points.

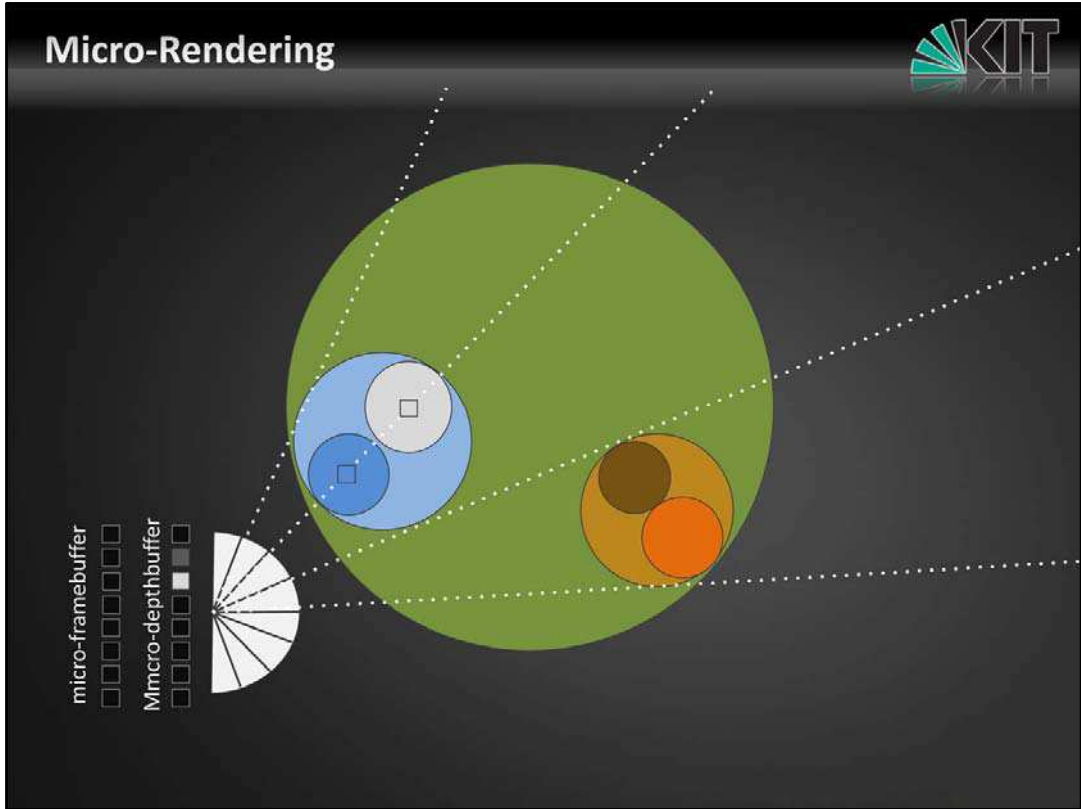
Once we have a micro-rendering, this small frame buffer with incident light, we can convolve it with the BRDF to compute the reflected light.



It essentially uses a Qsplat like point hierarchy and rendering technique, here illustrated in 1D.

The micro-framebuffer is on the left and we consider this little binary tree of points (circles) on the right.

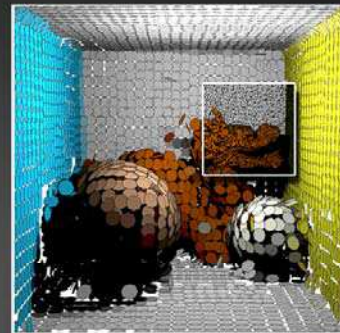
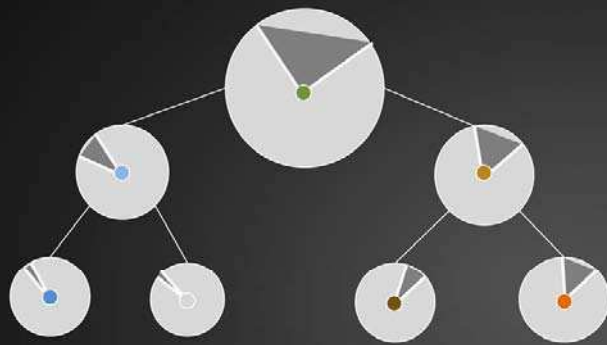
This tree as a 2d bouncing volume hierarchy, looks like this.



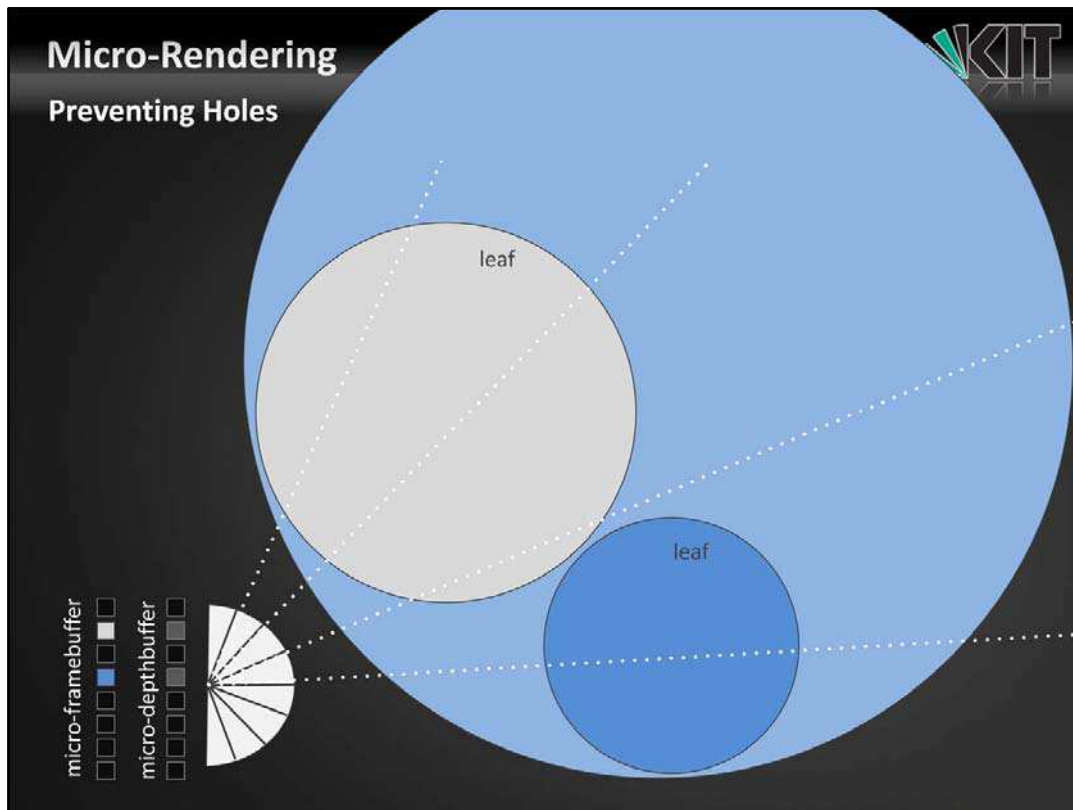
To resolve occlusions correctly, we also use a micro-depth-buffer.
 In this example, the white and blue nodes fall into a single pixel, but using the depth buffer, the correct ordering can be resolved.

Micro-Rendering

Tree Representation



We use a complete binary tree to store the geometry into a texture.
We store node position and radius, as well as normal cones for each node.



Doing all this, there is one special case that needs consideration: If a node is very close to a receiver it might become bigger than a micro-pixel.

We tried several established procedures for point rendering, including the pull-push we used in imperfect shadow maps, but the most accurate and reliable method was to raycast such surfels as small disks.

Please note, that such raytracing does not mean we have to find an intersection for the entire scene in some directions. It only means, that we need to find the intersection point of one direction and handful of disks.

Micro-Rendering

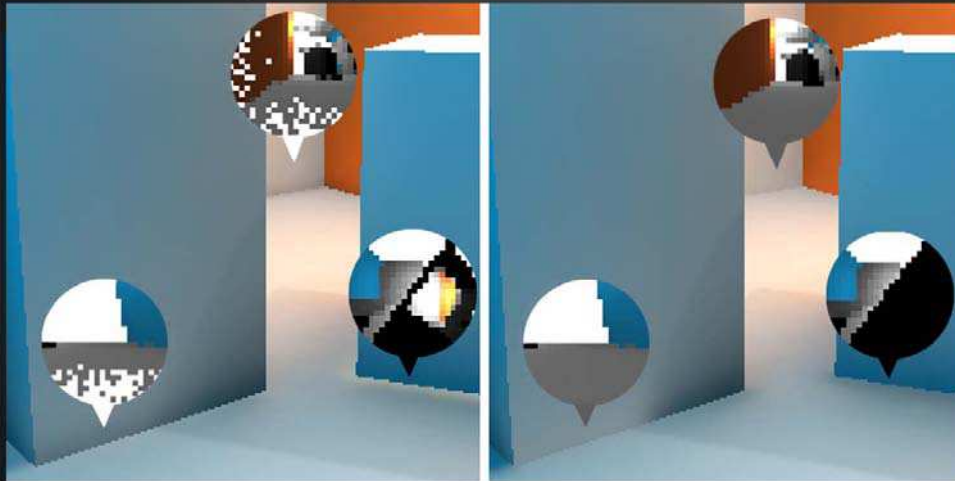


Preventing Holes

- ▶ ... by ray-casting

without ray-casting

with ray-casting



Let me show some examples, how this is useful.

Those images might look similar, but in corners, where GI is difficult, prominently for VPLs, they are different.

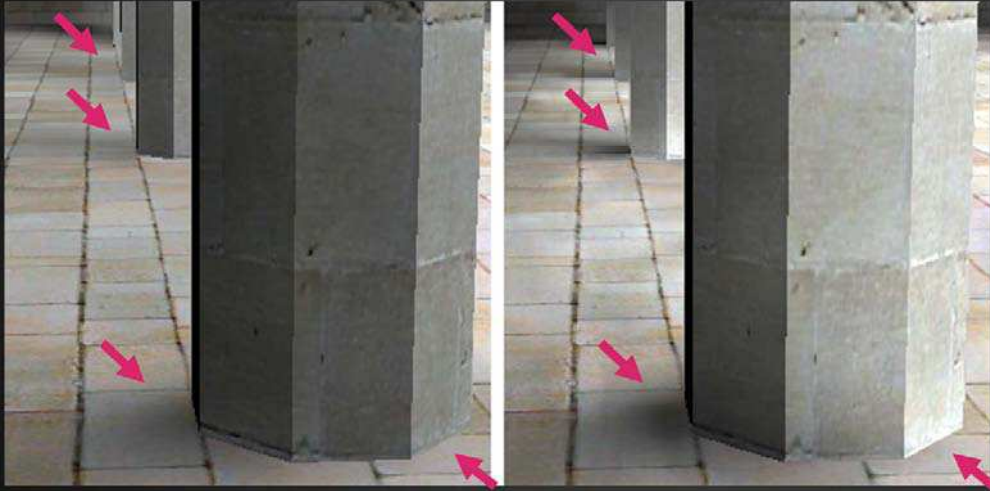
Without raycasting, the nearby surface has holes, that can be filled by raycasting.

Micro-Rendering



Comparison ISM vs. Micro-Rendering

- ▶ ISM: visibility at VPLs (left)
- ▶ micro-rendering: visibility at/near shading points (right)



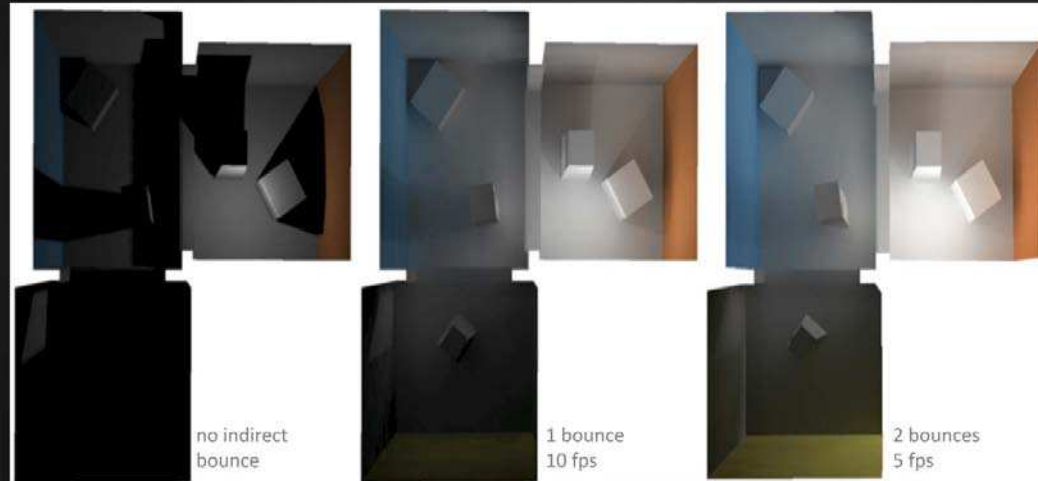
For final gathering, micro-rendering is clearly better, as it resolves visibility at surface locations, thus capturing occlusions better.

Micro-Rendering



Multiple-Bounces

- ▶ use point-hierarchy for radiosity-style light transport



Since we store a point hierarchy (with additional information such as color/BRDF, orientation) we have all information to compute a radiosity-like energy transfer.

That is, from direct illumination, we can compute point-to-point transfer, also multiple iterations and thus achieve multiple bounces – just as an excursus.

Micro-Rendering



Final Gathering

- ▶ use point-hierarchy for visibility
- ▶ gather energy from visible photons or VPLs



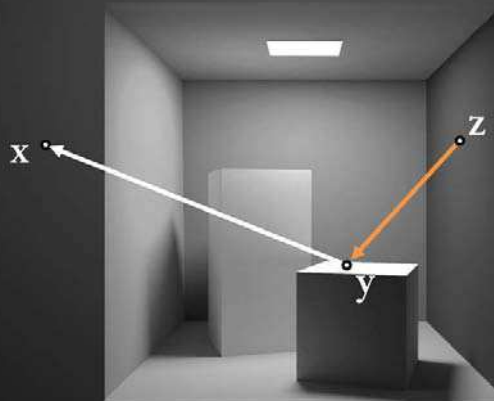
More interesting for many-lights methods are two things:

1. we can render high-quality shadow maps quickly
2. we can use final gathering like lighting by using micro-rendering to resolve visibility from a surface point to other surfaces and thus also to VPLs residing there.

There are also further extensions that I'm not mentioning today, e.g. micro-rendering can also account for the BRDF and warp the micro-buffer essentially to do importance sampling.

► rendering equation:

$$L(\mathbf{x} \leftarrow \mathbf{y}) = \underbrace{L_e(\mathbf{x} \leftarrow \mathbf{y})}_{\text{emitted light}} + \underbrace{\int_A f_r(\mathbf{x} \leftarrow \mathbf{y} \leftarrow \mathbf{z}) G(\mathbf{y} \leftrightarrow \mathbf{z}) V(\mathbf{y} \leftrightarrow \mathbf{z}) L(\mathbf{y} \leftarrow \mathbf{z}) dA}_{\text{reflected light}}$$



Now that we “solved” the problems of creating and using VPLs, I would like to address the issue of quality or correctness a bit more, speaking about bias compensation and how this can be done in interactive applications.

- ▶ rendering equation:

$$L(\mathbf{x} \leftrightarrow \mathbf{y}) = \underbrace{L_e(\mathbf{x} \leftrightarrow \mathbf{y})}_{\text{emitted light}} + \underbrace{\int_A f_r(\mathbf{x} \leftrightarrow \mathbf{y} \leftrightarrow \mathbf{z}) G(\mathbf{y} \leftrightarrow \mathbf{z}) V(\mathbf{y} \leftrightarrow \mathbf{z}) L(\mathbf{y} \leftrightarrow \mathbf{z}) dA}_{\text{reflected light}}$$

- ▶ operator notation [Arvo et al. 1994]:

$$(\mathbf{T}L)(\mathbf{x} \leftrightarrow \mathbf{y}) = \int_A f_r(\mathbf{x} \leftrightarrow \mathbf{y} \leftrightarrow \mathbf{z}) G(\mathbf{y} \leftrightarrow \mathbf{z}) V(\mathbf{y} \leftrightarrow \mathbf{z}) L(\mathbf{y} \leftrightarrow \mathbf{z}) dA$$

$$L = L_e + \mathbf{T}L$$

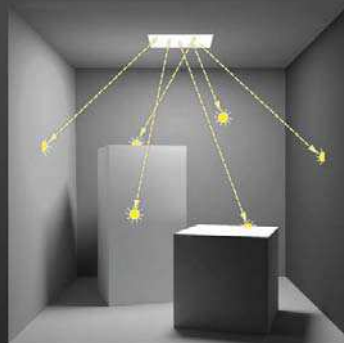
well, the rendering equation...

Many-Lights Rendering



- ▶ create **virtual point lights** as before
- ▶ we assume to use VPLs to approximate indirect illumination \hat{L} only

distribution of VPLs



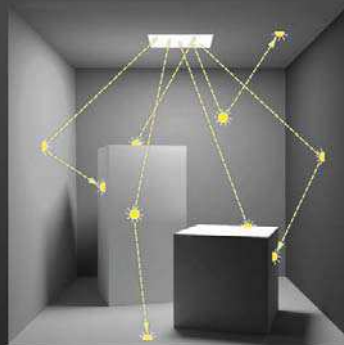
creation of random walks (here we simply assume that this can be done, no matter if using ray casting or rasterization)

Many-Lights Rendering



- ▶ create **virtual point lights** as before
- ▶ we assume to use VPLs to approximate indirect illumination \hat{L} only

distribution of VPLs



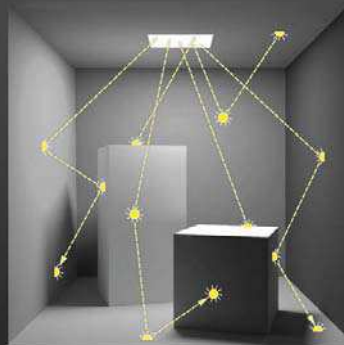
creation of random walks (here we simply assume that this can be done, no matter if using ray casting or rasterization)

Many-Lights Rendering



- ▶ create **virtual point lights** as before
- ▶ we assume to use VPLs to approximate indirect illumination \hat{L} only

distribution of VPLs



creation of random walks (here we simply assume that this can be done, no matter if using ray casting or rasterization)

Many-Lights Rendering

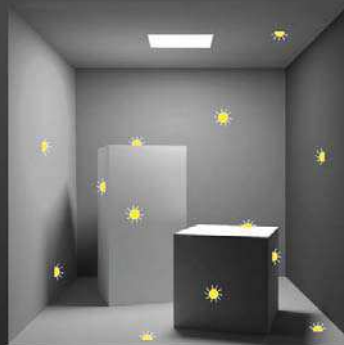


- ▶ create **virtual point lights** as before
- ▶ we assume to use VPLs to approximate indirect illumination \hat{L} only

$$L = L_e + \mathbf{T}L$$

$$L = L_e + \mathbf{T}L_e + \mathbf{T}\hat{L}$$

distribution of VPLs



we assume to use VPLs to approximate indirect illumination only – direct illumination is typically computed using dedicated methods with some shadow mapping variant.

Many-Lights Rendering

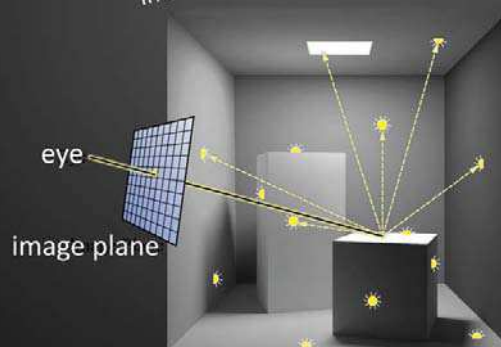


- ▶ create **virtual point lights** as before
- ▶ we assume to use VPLs to approximate indirect illumination \hat{L} only

$$L = L_e + \mathbf{T}L$$

$$L = L_e + \mathbf{T}L_e + \mathbf{T}\hat{L}$$

direct emission
direct illumination
indirect illumination



Let's see what happens when computing the indirect light...

here everything is fine, just accumulate the contributions from all VPLs.

Many-Lights Rendering

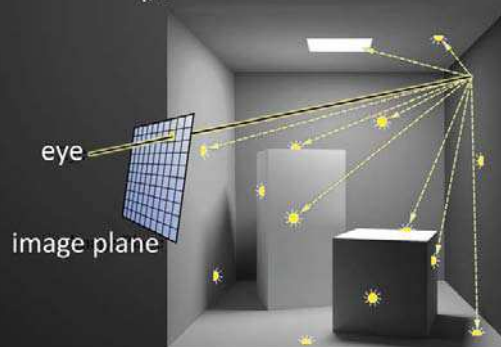


- ▶ create **virtual point lights** as before
- ▶ we assume to use VPLs to approximate indirect illumination \hat{L} only

$$L = L_e + \mathbf{T}L$$

$$L = L_e + \mathbf{T}L_e + \mathbf{T}\hat{L}$$

direct emission
direct illumination
indirect illumination



... here it is different...

Many-Lights Rendering – Singularities



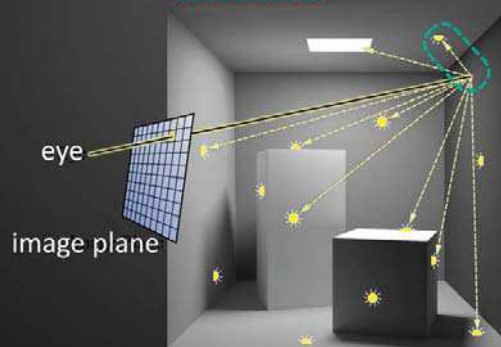
$$L = L_e + \mathbf{T}L_e + \mathbf{T}\hat{L}$$

transport operator:

$$(\mathbf{T}\hat{L})(\mathbf{x} \leftarrow \mathbf{y}) = \sum_{i=1}^N f_r(\mathbf{x} \leftarrow \mathbf{y} \leftarrow \mathbf{z}_i) G(\mathbf{y} \leftrightarrow \mathbf{z}_i) V(\mathbf{y} \leftrightarrow \mathbf{z}_i) \hat{L}(\mathbf{y} \leftarrow \mathbf{z}_i)$$

geometry term:

$$G(\mathbf{y} \leftrightarrow \mathbf{z}_i) = \frac{\cos^+(\theta_{\mathbf{y}}) \cos^+(\theta_{\mathbf{z}_i})}{\|\mathbf{y} - \mathbf{z}_i\|^2}$$



... here it is different... as there's a nearby VPL, which – because of the geometry term – will create a bright splotch.

Artifacts - Splotches



reference (slow) rendering



fast rendering with less VPLs

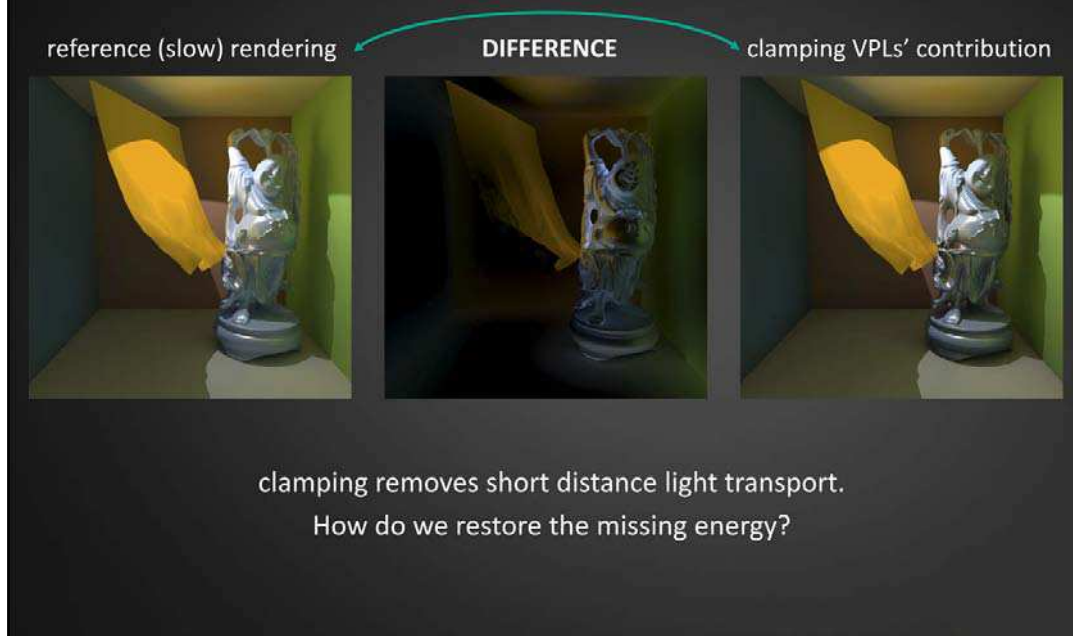


clamping VPLs' contribution



clamping the contribution of nearby VPLs
by bounding the geometry term

the naïve solution – as we have seen – is to clamp the VPLs' contributions...



however, bounding the geometry term introduces a systematic error.

- In fact, we remove a short distance light transport
- The bias is visible as artificial darkening around concave features

Bounded and Residual Light Transport



full LT: $L_e + \mathbf{T}L_e + \mathbf{T}\hat{L}$

$$\mathbf{T}\hat{L} = \sum_{i=1}^N f_r G V \hat{L}$$

bounded indirect LT: $L_e + \mathbf{T}L_e + \mathbf{T}_b \hat{L}$

$$\mathbf{T}_b \hat{L} = \sum_{i=1}^N f_r \min(G, b) V \hat{L}$$

residual indirect LT: $\mathbf{T}_r \hat{L}$

$$\mathbf{T}_r \hat{L} = \sum_{i=1}^N f_r \max(G - b, 0) V \hat{L}$$

b : user-defined bound

We can now have a closer look what's happening when clamping.

The clamped transport operator \mathbf{T}_b describes VPL lighting with clamped contributions and is shown on the slide.

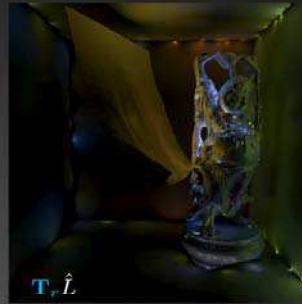
We can now define another transport operator (\mathbf{T}_r for residual transport), which describes just this amount of light which has been clamped away.



-

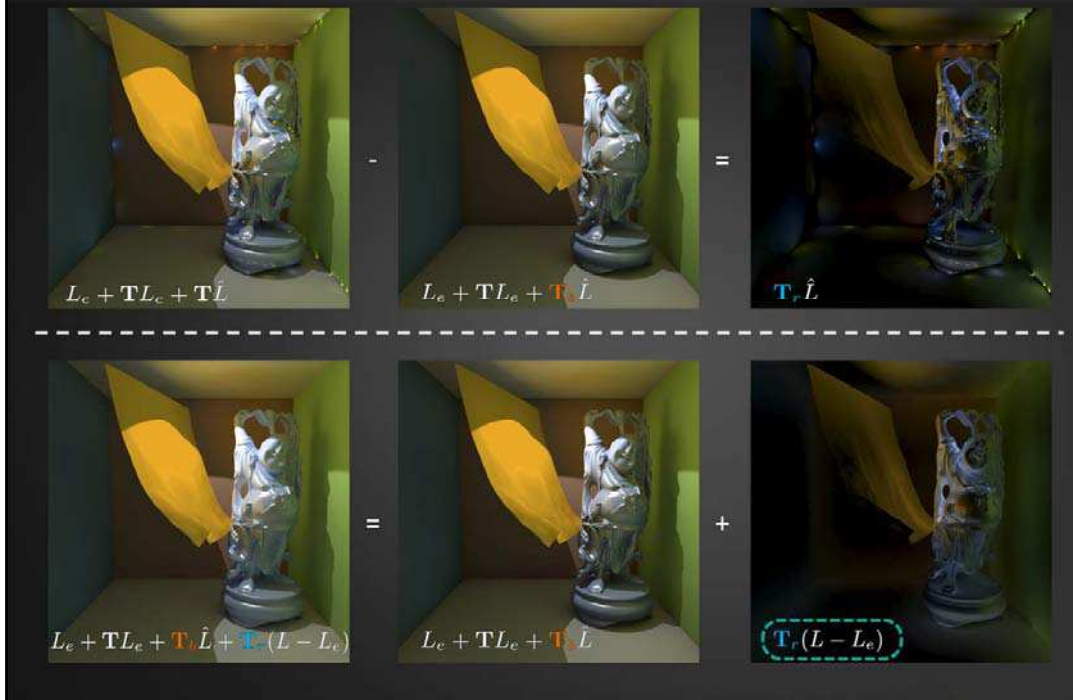


=



$$\mathbf{T} = \mathbf{T}_b + \mathbf{T}_r$$

Bounded and Residual Light Transport



Here you can see how these transport operators related to each other

Illumination in the Presence of Weak Singularities [Kollig and Keller 2004]

- ▶ bias compensation via tracing additional rays
- ▶ indirect illumination for residual transport computed via path tracing
- ▶ unbiased but computational very intensive (~hours)
- ▶ infeasible for interactive applications



$$L = L_e + \mathbf{T}L_e + \mathbf{T}_b\hat{L} + \mathbf{T}_r\hat{L}$$

$$L = L_e + \mathbf{T}L_e + \mathbf{T}_b\hat{L} + \mathbf{T}_r(L - L_e)$$

← computed using path tracing

The standard way of compensating the bias – the error due to clamped-away energy – is what we have already seen in the previous parts.

Reformulated Bias Compensation:

- | | |
|---|---|
| <ul style="list-style-type: none"> ▶ motivation: <ul style="list-style-type: none"> ▶ restore energy, remove bias ▶ interactive frame-rates | <ul style="list-style-type: none"> ▶ solution: <ul style="list-style-type: none"> ▶ re-use the existing (clamped) solution ▶ iteratively apply the residual transport |
|---|---|

$$L = L_e + \mathbf{T}L_e + \mathbf{T}_b\hat{L} + \mathbf{T}_r(L - L_e)$$

$$L = L_e + \sum_{i=0}^{\infty} \mathbf{T}_r^i (\mathbf{T}L_e + \mathbf{T}_b\hat{L})$$

← compute once
← apply iteratively

design choice: compute and apply in screen-space

We have shown that you can compute the bias compensation differently (see paper for details).

The reformulation itself does not tell you how to actually compute it, but it was done having in mind that we render an image with clamped VPLs first, and then use only this operation to recover the missing energy.

- ▶ precomputation:
 1. distribute VPLs (as before)
 2. create an imperfect shadow map for every VPL

- ▶ rendering:
 1. render the scene to find visible surfaces
 2. apply deferred direct and **bounded** VPL lighting $\mathbf{T}L_e + \mathbf{T}_b \hat{L}$
 3. N-times in screen-space:
compute **residual** transport and add it to the image

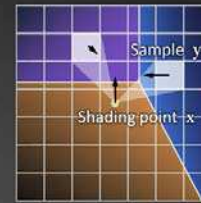
$$\sum_{i=0}^{\infty} \mathbf{T}_r^i (\mathbf{T}L_e + \mathbf{T}_b \hat{L})$$

This is the algorithm overview...

Screen-Space Integration



- ▶ FOR EACH pixel:
 - ▶ iterate over neighboring pixels
 - ▶ IF $G(\mathbf{x} \leftrightarrow \mathbf{y}) > b$:
 - ▶ add contribution in "radiosity style"



camera view

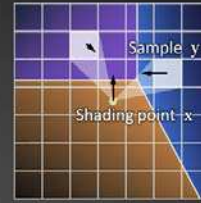
Screen-Space Integration



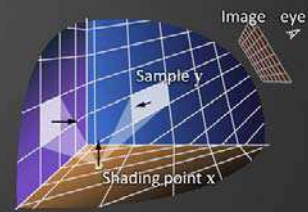
- ▶ **FOR EACH** pixel:
 - ▶ iterate over neighboring pixels
 - ▶ **IF** $G(\mathbf{x} \leftrightarrow \mathbf{y}) > b$:
 - ▶ add contribution in "radiosity style"

$$\left(\frac{\cos^+(\theta_x) \cos^+(\theta_y)}{\|\mathbf{x} - \mathbf{y}\|^2} - b \right) A f_r \bar{L}$$

G-buffer stores all necessary information.



camera view

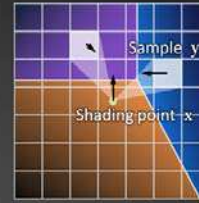


side view

Screen-Space Integration

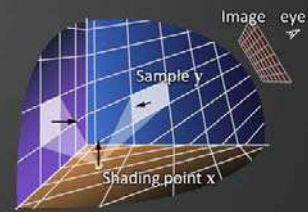


- ▶ FOR EACH pixel:
 - ▶ iterate over neighboring pixels
 - ▶ IF $G(x \leftrightarrow y) > b$:
 - ▶ add contribution in "radiosity style"



camera view

we are only interested in the clamped energy



side view

Screen-Space Integration



▶ FOR EACH pixel:

▶ iterate over neighboring pixels

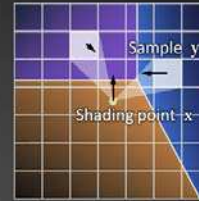
▶ IF $G(\mathbf{x} \leftrightarrow \mathbf{y}) > b$:

▶ add contribution in "radiosity style"

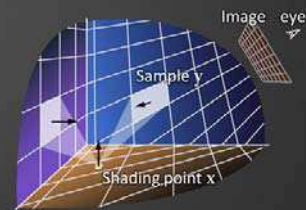
clamping occurs in a close neighborhood only!
close in world space = close in screen-space

$$G(\mathbf{x} \leftrightarrow \mathbf{y}) = \frac{\cos^+(\theta_x) \cos^+(\theta_y)}{\|\mathbf{x} - \mathbf{y}\|^2}$$

we can conservatively estimate a bounding radius
and restrict the integration to it



camera view



side view

Screen-Space Integration



▶ FOR EACH pixel:

▶ iterate over neighboring pixels

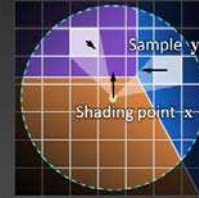
▶ IF $G(\mathbf{x} \leftrightarrow \mathbf{y}) > b$:

▶ add contribution in "radiosity style"

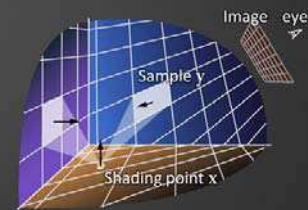
clamping occurs in a close neighborhood only!
close in world space = close in screen-space

$$G(\mathbf{x} \leftrightarrow \mathbf{y}) = \frac{\cos^+(\theta_x) \cos^+(\theta_y)}{\|\mathbf{x} - \mathbf{y}\|^2}$$

we can conservatively estimate a bounding radius
and restrict the integration to it



camera view

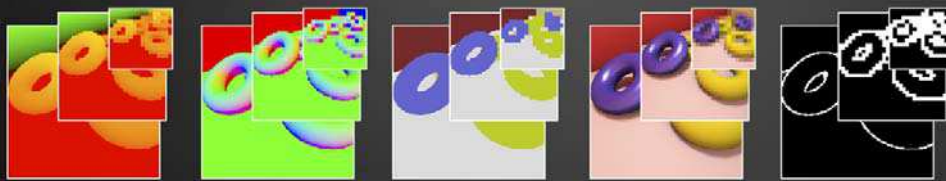


side view

Hierarchical Integration

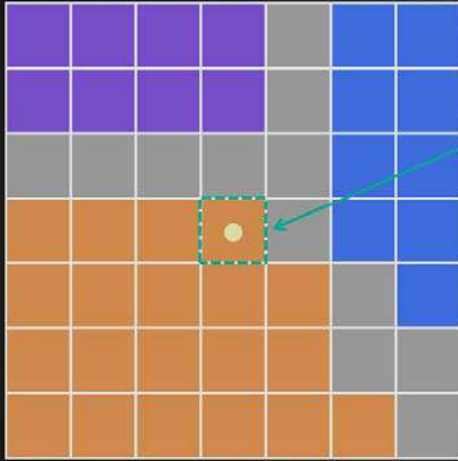


- ▶ still too many samples (even with the bounding radius)
- ▶ multiresolution top-down integration (in spirit of [Nichols et al. 2009])
- ▶ requires
 - ▶ a mip-map chain of the G-Buffer and bounded illumination
 - ▶ discontinuity buffer



In order to speed up the rendering we use a hierarchical screen space approach inspired by multiresolution splatting of Nichols and Wyman, please see the paper for details.

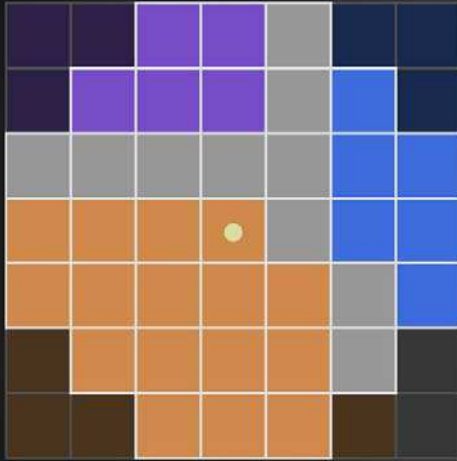
Hierarchical Integration



bias compensation for this pixel

coarsest level

Hierarchical Integration



limit the integration domain

coarsest level

Compute a conservative bounding radius outside which, none of the pixels can contribute to compensation

Hierarchical Integration



		1	1	R		
	1	R	R	R	1	
1	R	R	R	R	R	1
0	0	0	●	R	R	1
0	0	0	0	0	R	1
	0	0	0	0	0	
		0	0	0		

coarsest level

for each pixel
if *possible* add contribution
else refine

1 : non-zero contribution
0 : zero contribution
R : must be refined

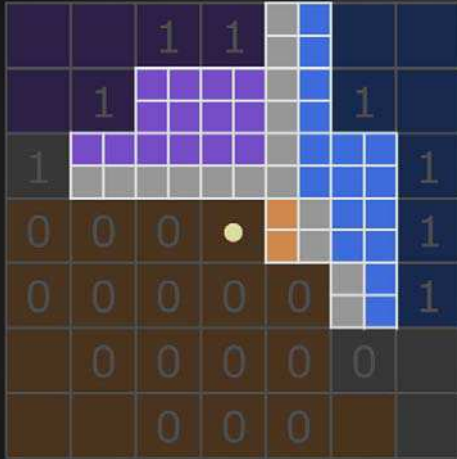
- ▶ refine if
 - ▶ G-term too large
 - ▶ discontinuity

If **possible**:

Gterm not too high, otherwise error

No discontinuity, otherwise data in G-Buffer inaccurate

Hierarchical Integration



finer level

refine

Hierarchical Integration



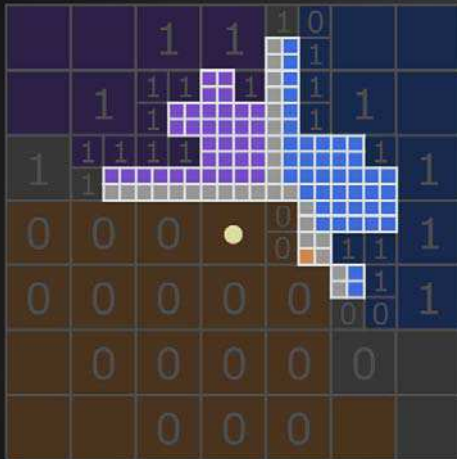
finer level

for each pixel
 if *possible* add contribution
 else refine

1 : non-zero contribution
 0 : zero contribution
 R : must be refined

- ▶ refine if
 - ▶ G-term too large
 - ▶ discontinuity

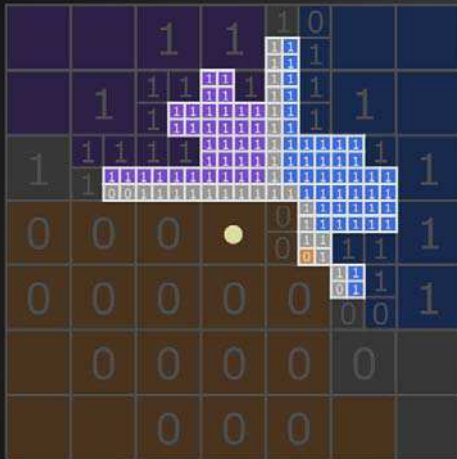
Hierarchical Integration



refine

finest level

Hierarchical Integration



finest level

for each pixel
add contribution

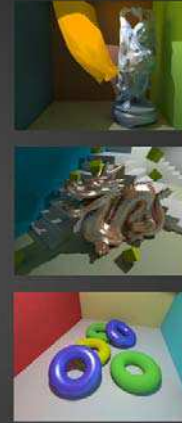
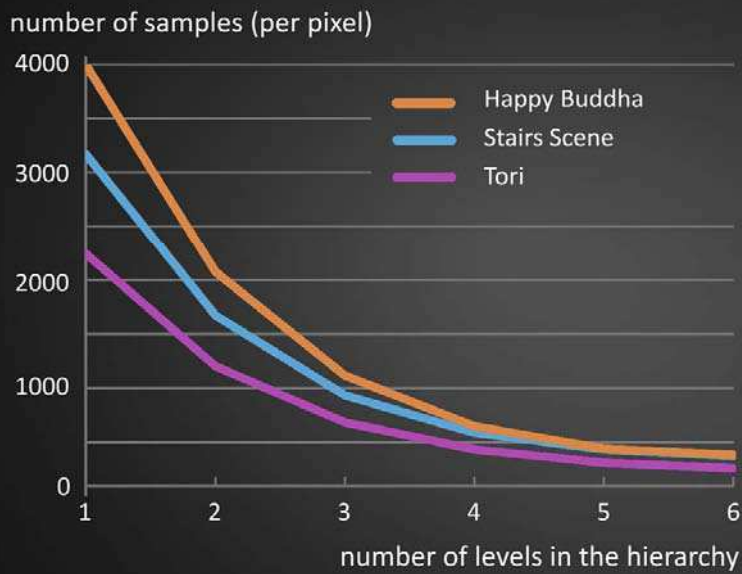
1 : non-zero contribution
0 : zero contribution

Hierarchical Integration – Overview



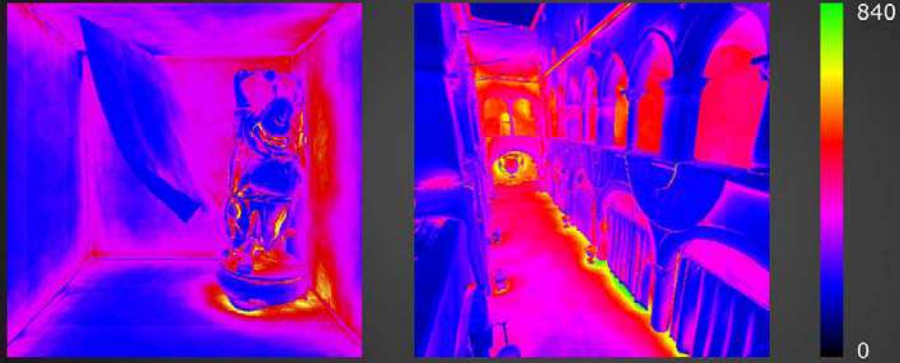
		1	1	1	0		
	1	1	1	1	1	1	
1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1
	0	0	0	0	0		
		0	0	0			

Hierarchical Integration – Evaluation



The hierarchical integration obviously reduces the computational demands, while not degrading the quality noticeably.

number of samples (per pixel)



The number of samples (i.e. queried pixels) has important impact on the rendering performance and is shown here.

Rendering Results – Dragon



Bounded Light Transport

Residual Light Transport

Final Image



rendered with:
1024x768 at:

no SSBC
10.3 FPS

1-step SSBC
8.2 FPS

2-step SSBC
6.4 FPS

These are some results showing the clamped image, recovered energy, and final image.

Rendering Results – Dragon



1st step of SSBC



2nd step of SSBC



rendered with:
1024x768 at:

no SSBC
10.3 FPS

1-step SSBC
8.2 FPS

2-step SSBC
6.4 FPS

Multiple iterations recover more energy, but the additional energy drops exponentially because of the transport operator (the BRDF in there respectively)

Rendering Results – Tori



Bounded Light Transport

Residual Light Transport

Final Image



rendered with:
1024x768 at:

no SSBC
16.4 FPS

1-step SSBC
12.1 FPS

2-step SSBC
9.3 FPS

These are some results showing the clamped image, recovered energy, and final image.

Rendering Results – Crytek Sponza



Bounded Light Transport

Residual Light Transport

Final Image



rendered with:
1024x768 at:

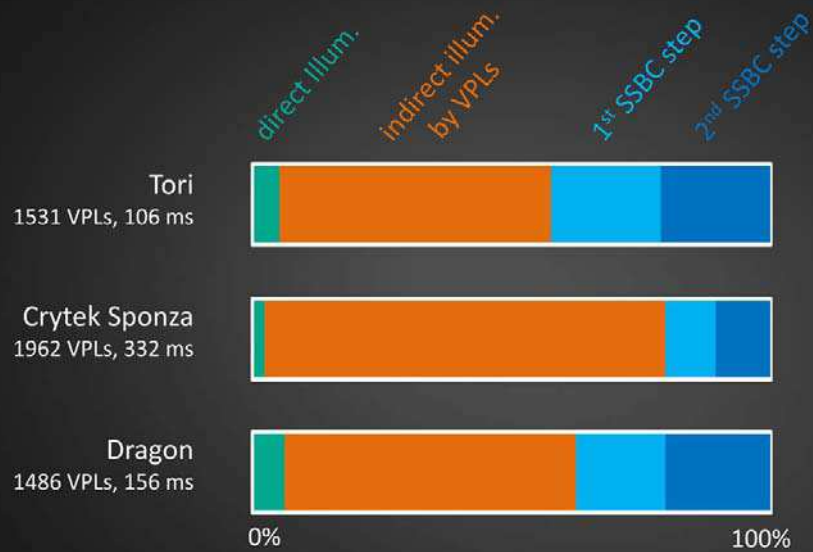
no SSBC
3.8 FPS

1-step SSBC
3.4 FPS

2-step SSBC
3.0 FPS

These are some results showing the clamped image, recovered energy, and final image.

SSBC Timings



Computing the illumination due to VPLs is still the most costly part of the algorithm, i.e. bias compensation is really not expensive to add when computing it in screen space.

Comparison – [Kollig and Keller 2004] vs. SSBC



Compensation Only

Final Image

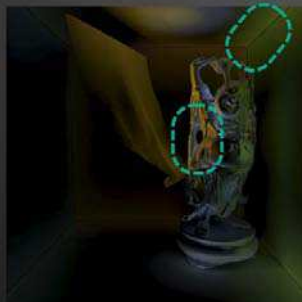
Bias Compensation
[Kollig and Keller 2004]

CPU ~ 10.9 hours
(8-core, 4GB RAM)



Screen-Space
Bias Compensation
(3 steps)

GPU ~ 550 ms
(ATI Radeon HD 5870)

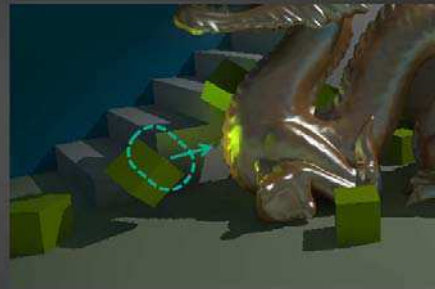
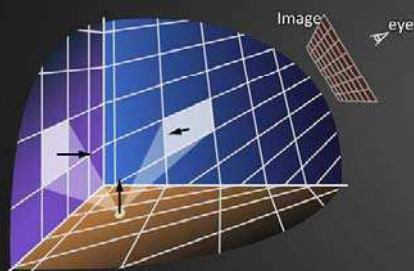


Comparison to ray casting based bias compensation illustrating the impact of a screen-space approximation of the scene's surfaces.

Artifacts in Screen-Space Integration



- ▶ sources of artifacts:
 - ▶ surface sampling at grazing view angles



work-around: conservative bias compensation

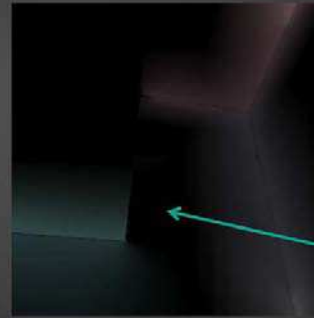
screen-space approximation of the scene's surfaces means: not information about all surfaces in the scene and irregular sampling of the scene's surfaces.

Surfaces seen under grazing angles are overestimated, as a pragmatic solution, we bound their contributions.

Artifacts in Screen-Space Integration



- ▶ sources of artifacts:
 - ▶ surface sampling at grazing view angles
 - ▶ hidden surfaces



missing
compensation

work-around: use multiple viewports

And of course we have not information on hidden surfaces.

Note the screen-space approach is a design chose, the reformulated compensation would work with other representations (e.g. finite elements) as well.

Conclusion



Many-Lights Methods and Interactive Applications

- ▶ it's all about visibility computation
 - ▶ rasterization to resolve from-point visibility
- ▶ image-space techniques can also be used to accelerate bias compensation

Acknowledgements: many of the slides used in/adapted for this presentation have been created by Tobias Ritschel and Jan Novak

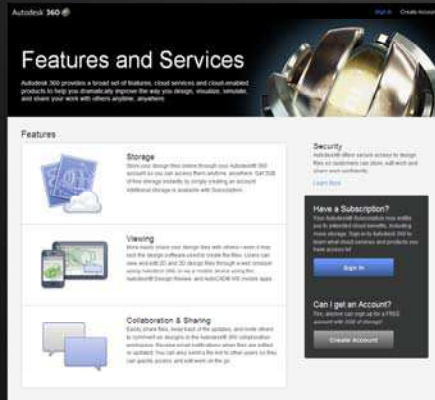
Many-lights methods in Autodesk 360 Rendering

Adam Arbree



This final section of the course discusses the use of many lights algorithms in our cloud rendering service, Autodesk® 360 Rendering.

Autodesk® 360




- Cloud Application Suite
- Goals
 - Storage
 - Sharing
 - Collaboration
- Integrated into our core desktop applications




Autodesk

To begin I want to quickly introduce our service to illustrate the challenges that many lights algorithms help to us address. Our service is a component in a system of cloud applications, called Autodesk 360, launched last March. The goal of the system is to provide a suite of tools, accessible everywhere through web and mobile front ends, that allows users to store, share and collaborate on projects using our software. Our desktop applications are integrating with this cloud system to enable a seamless transition between local work and the cloud resources.


Autodesk® 360 Rendering



Autodesk® Revit® 2013



Autodesk® AutoCAD® 2013

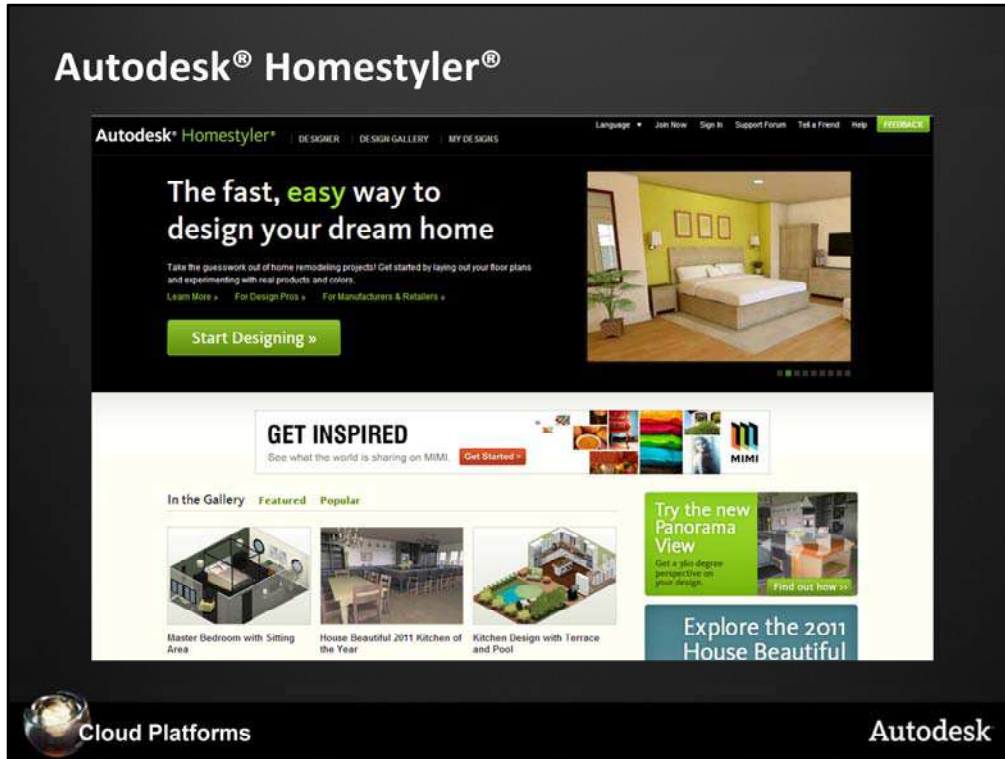


Render Gallery

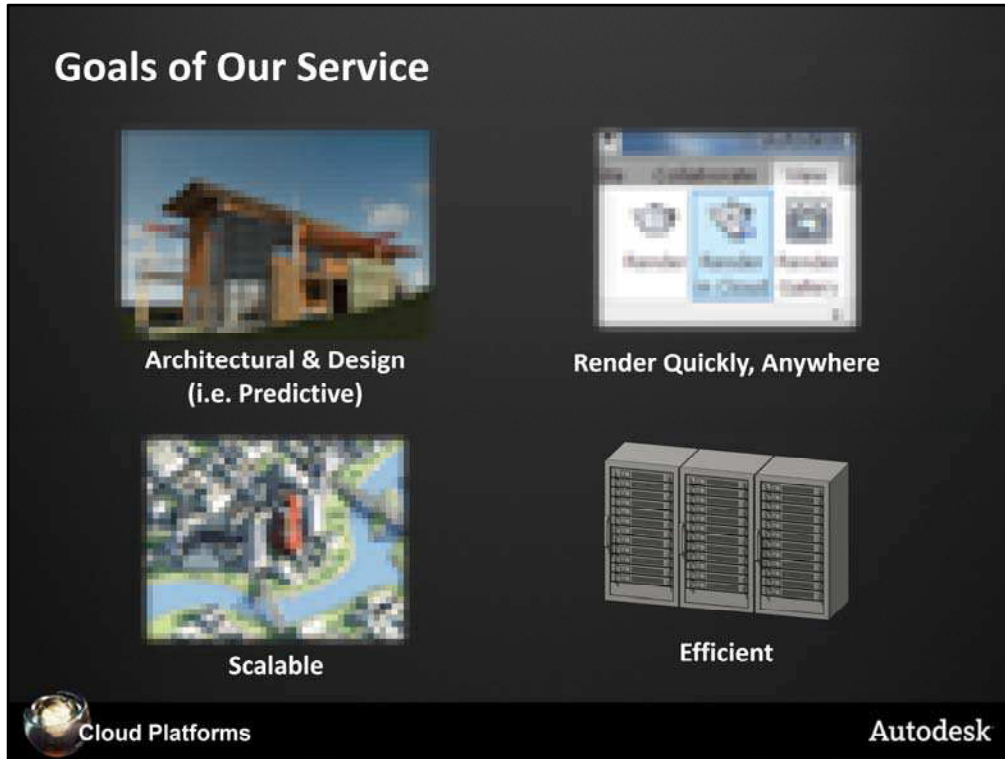
Cloud Platforms

Autodesk

For rendering specifically, we have added cloud rendering options, mirroring to the previous desktop rendering options, to our Autodesk® Revit® and Autodesk AutoCAD® applications. Users can use these cloud rendering tools to upload and render their models remotely in the background. Completed renderings become available in our render gallery website where they can be viewed, modified and re-rendered with a list of advanced features, such as panorama views.



In addition our service provides back-end visualization support for consumer products such as Autodesk® Homestyler®, a tool lets users build and render their own interior design plans.



Working back from these applications, we can sketch out the goals of our cloud rendering system. First we focus primarily on architectural and design visualization. This focus is both challenging, because these applications demand higher-quality, predictive simulations, and simplifying, because these applications use a reduced palette of materials, lights and geometry that are generally physically modeled. Second, we want to make rendering a one-click option available anywhere in any product. This allows us to support consumer applications, such as Homestyler, and challenges us to make rendering simpler and easier to use. Third, we need our renderer to scale. Our users turn to the cloud to render their largest and most complex scenes, those beyond the capacity of their desktops, and they expect results quickly. And, finally of course, we need the renderer to be efficient since Autodesk bears the cost of the cloud compute resources.

Problem

How to **automatically, efficiently and reliably** produce a large number of physically-accurate renderings in a **predictable** amount of time?

Solution?

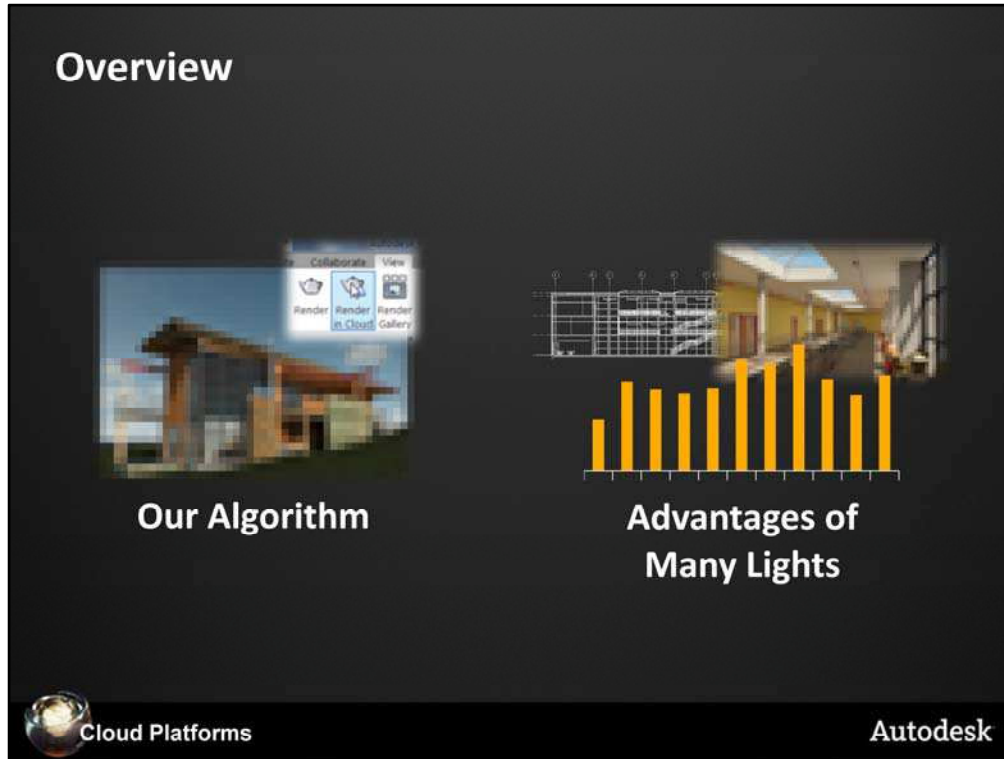
Use a many-lights rendering algorithm.



Cloud Platforms

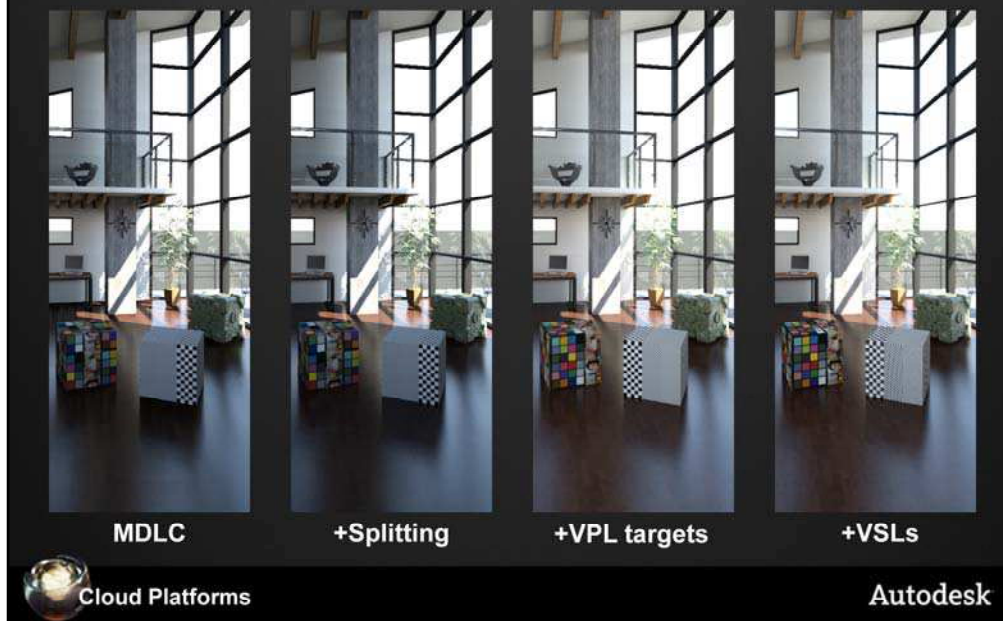
Autodesk

Now, meeting these goals can be consolidated into the central problem of our cloud rendering application: how can we automatically, efficiently and reliably produce a large number of physically-accurate renderings in a predictable amount of time? For our application, the solution to this problem was largely to use a many lights rendering algorithm. This talk discusses how we implemented many lights rendering in our system and why it was critical to our success.



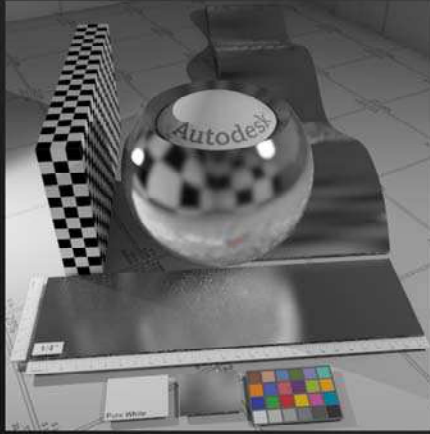
This talk will have two parts. The first part discusses the rendering algorithm we built at Autodesk and some of the implementation issues we addressed when developing that system. Then the second part discusses the advantages a many lights rendering algorithm brought to our application. The overall point of this second discussion is that many lights algorithms have proven to be faster and are fundamentally more scalable. Our results show that this holds true across a very wide array of images and scenes. However, since raw performance has been discussed at length in this course, this talk will focus on additional consequences of that scalability. Specifically, the goal of this talk is to describe how these algorithms also improve the reliability and predictability our rendering system and how their advantages have been critical to the success to our service. They have helped us to make rendering easier for novice users, to provide more consistent results for our customers and to improve the quality of images under fixed cost constraints.

Rendering Algorithm

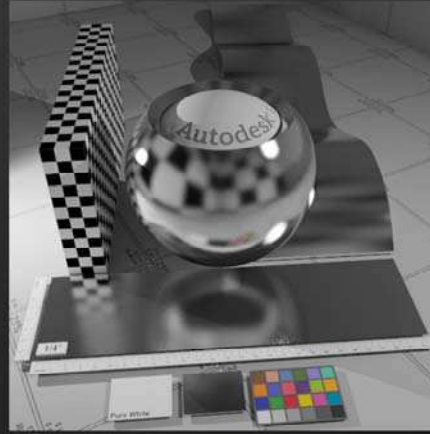


Our service uses Multidimensional Lightcuts to compute its images. We implemented virtual spherical lights instead of VPLs to avoid some clamping and reduce the appearance of corner darkening. Our basic implementation is essentially identical to those described earlier in the course. However, in implementing these algorithms, we needed to address three important issues. First, we needed to add some form of bias compensation to better render glossy materials, particularly when there are glossy inter-reflections. We introduce an eye ray splitting heuristic solve this problem. Second, many scenes had difficult lighting occlusion and we faced issues generating robust VPL distributions. To generate more efficient VPLs sets, we added a VPL targeting method. Finally, we needed support for directional point light emission so we created a simple directional VPL type.

Issue #1: Glossy Objects



Basic VSLs



Our Solution



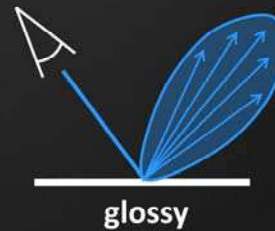
Cloud Platforms

Autodesk

High gloss materials are common in our scenes. We found that clamping, sufficient to avoid noise, could significantly effect a material's appearance and that VSLs alone could not solve the problem (see the image on the left above and note the missing reflections of the glossy highlights in particular). To address this problem, we recursively continue our eye paths for glossy materials.

Solution #1: Eye Ray Splitting

- Split and recursively trace eye rays for glossy materials
- Heuristic determines split rate from material's glossiness
- Increase maximum cut size to accommodate increased sampling



Cloud Platforms

Autodesk

This recursion is identical to the recursion used for delta specular materials. When an eye path hits a sufficiently glossy surface, instead of creating a gather point immediately, we sample secondary rays and continue. The number of secondary rays and the decision to split is determined by a heuristic. Unfortunately, splitting does increase cost significantly; requiring an increase in the allowed maximum cut size.

Issue #2: High Occlusion



Our Solution

Naïve MDLC



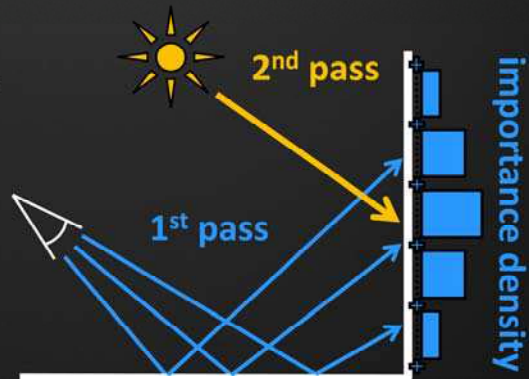
Cloud Platforms

Autodesk

Our second issue was large scenes with significant occlusion to the lighting. In this example, we have a large office building model. In this image we are viewing only a single hallway in the model. Without modification to the VPL tracing algorithm, it is difficult to generate a sufficient density of VPLs in this region; instead VPLs are distributed over large regions of the model out of frame. As can be seen on the left side of this image, these low densities result in a negative, darker bias and banding artifacts on the ceiling. To address this issue, we resample the VPLs to ensure a higher density near the camera. To do this, we use a VPL targeting method to redistribute these VPLs. Empirically we have found that this targeting reduces bias, improves quality and reduces cost.

Solution #2: VPL Targeting

- Modeled on [photon mapping with importance](#)
- Two pass algorithm
 - 1st pass: Trace eye ray samples
 - 2nd pass: Use importance density estimates to reject VPLs with Russian roulette

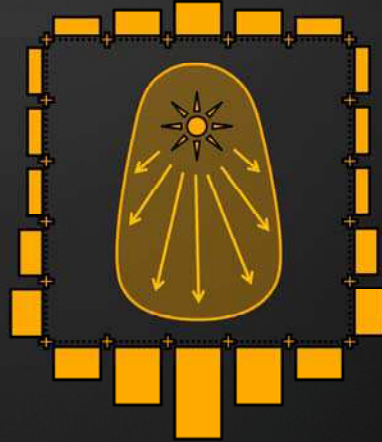


Autodesk

Our targeting method uses image importance to estimate the quality of a potential VPL. This method is modeled on methods for photon mapping with importance discussed by Per Christensen ([Christensen, P. "Adjoints and Importance in Rendering: An Overview." TVCG 9\(3\). 2003.](#)) . Our targeting algorithm splits the VPL tracing into two passes. In the first pass, importance carrying paths are generated from the eye. Exactly as in photon mapping, the resulting set of intersections ("importons") is stored and placed in a hierarchical acceleration structure. During the second VPL tracing pass, this importon map is queried to estimate the local importon density and we use Russian roulette to reject VPLs in low density regions.

Issue #3: Directionally Variant Lights

- Measured light emission profiles are commonly used
- Easy to add
- Use the material bounding cube map to bound the light emission function



Cloud Platforms

Autodesk


Finally, many of Autodesk's products support directionally emitting point lights and we had to implement a VPL type that could represent them. We found that the omni-directional VPL described in the original Multidimensional Lightcuts paper could be trivially extended to model directional emission. The only issue was bounding the light's emission function. To do this, we repurpose the cube maps that bound material reflectance also bound emittance.

Typical Results

1,000's of images/day

150s/megapixel (64 cores)

1st million images this year



Cloud Platforms Autodesk

The results from our first few months \have been robust and promising. We produce images using small clusters. Globally the cluster size is heterogeneous but on average we allocate 64 cores per image and produce a megapixel in 150s. Every day we render several thousand images and we should produce our millionth image sometime this year. The images on this slides are typical results shared by our customers.



A selection of images rendered by customers of 360 Rendering and shared on our Facebook® page (as of mid-May 2012).

Overview



Our Algorithm



Advantages of Many Lights



Cloud Platforms

Autodesk

In the second part of this discussion, I want to highlight the advantages of the many lights rendering algorithm in our application.

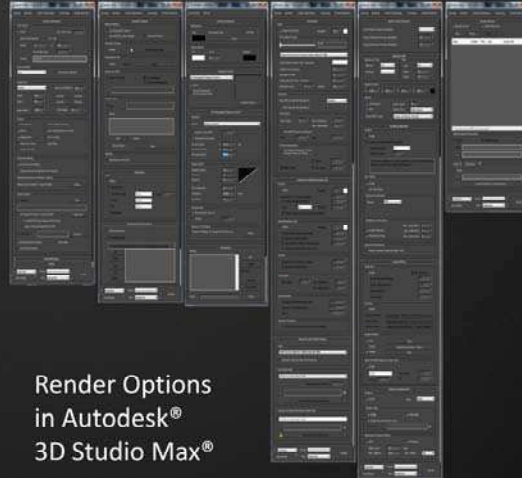
Advantage #1: Performance



Undoubtedly, the most important advantage of the many lights system is its performance. This point has been emphasized throughout this course but I want to reiterate it one last time. The performance advantages these algorithms have been significant and meaningful to our application for Autodesk. Not only has the performance impressed our customers but it has significantly reduced our costs. Every CPU/second our renderer saves is a CPU/second we don't have to buy and these algorithms save a lot of them. This is extremely valuable. However, since the performance has been stressed throughout the course, I want to take the remainder of this talk to discuss several additional advantages of these algorithms.

Advantage #2: Automatic Rendering

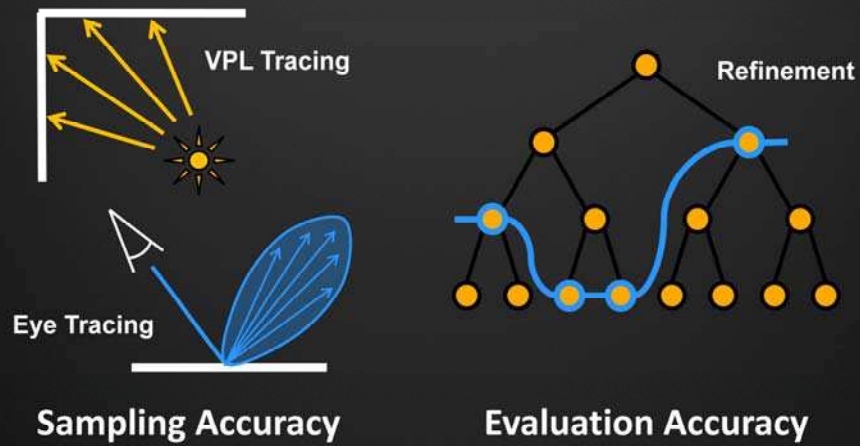
- Configuring a render can be a challenge...
 - Requires expertise
 - Image dependent
 - Time consuming
- Especially in design visualization where users want predictive images.



Autodesk

The first of these is that many lights algorithms are easy to automate. You can leverage their scalability, not just to make the rendering faster, but also to make the process of rendering easier. First, I note that configuring a render is a challenging task. On this slide, I have lined up the render options available in Autodesk® 3D Studio Max® (there are a lot of them). For an expert user, the complexity of these controls is extremely useful. They can tune the renderer to achieve a certain artistic look, tailor the simulation to be maximally efficient for a particular scene and selectively downgrade the computation for pre-visualization. However, for novice users these controls are confusing and, moreover in our cloud application, they are somewhat unnecessary. Users assume that in the cloud there are enough resources to compute their image and they are less concerned with performance tuning options. Our users just want a certain predictive quality without a lot of effort.

Advantage #2: Automatic Rendering

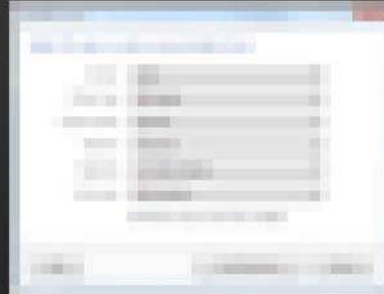


Autodesk

Conveniently the structure and scalability of many lights algorithms make this easy to achieve. One can divide a many-lights algorithm into two components. The VPL tracing and the eye ray generation represent a sampling component. The evaluation of the samples to generate an image is a second component. Largely users care about the how the second component behaves. It encapsulates an intuitive quality/cost tradeoff. However, only expert users understand how to correctly tune the first sampling component. Most users just want it to be set “correctly”.

Advantage #2: Automatic Rendering

- Many-lights algorithms facilitate automation
 - Set conservative sampling settings internally
 - Hide complex details the user
 - Use predefined quality settings for eye sampling rate and error thresholds
 - Rely on the scalable evaluation to avoid extra work



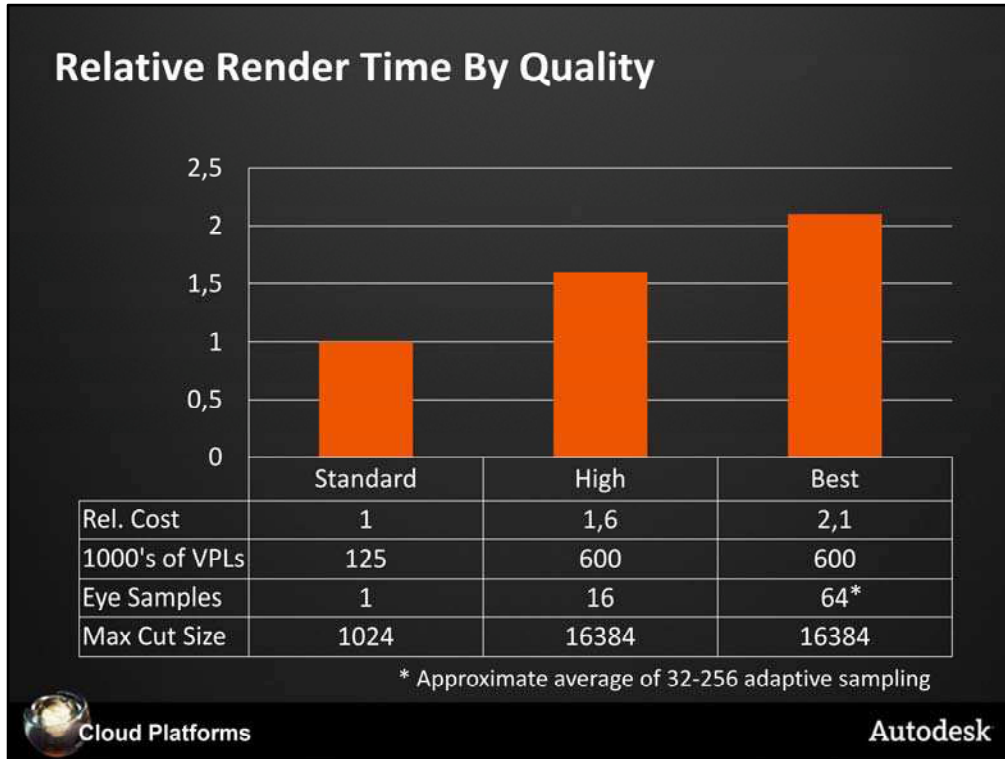
Render Options
in Autodesk®
360 Rendering



Cloud Platforms

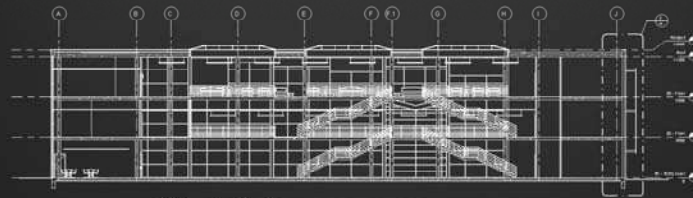
Autodesk

With many lights, we can do exactly this. In our application, we can permanently set conservative, “correct” VPL and eye sampling parameters internally in the renderer. This hides detailed mathematical information about our algorithm from the user. Instead we let the user control quality and time by choosing from a predefined set of quality choices controlling the error-bounds, cut size and sampling rates. This works because we can rely on the fundamental scalability of the renderer to avoid extra work if our sampling is a little too conservative. This lets the user focus on more intuitive parts of their request (see our render settings dialog) such as image size, quality and format.



I can demonstrate that this works well in practice. Here we look at the relative render times per megapixel for approximately 50,000 jobs rendered in April 2012. The results demonstrate two important features of a many lights algorithm. One, fixed conservative setting reliably produce images across a wide range of scenes. Two, the algorithms robustly avoid extra work: the highest quality is, on average, only 2x the cost of the baseline despite many more VPLs, eye samples and a much larger maximum cut size.

Advantage #3: Supports Design-size Models



Models have many purposes.



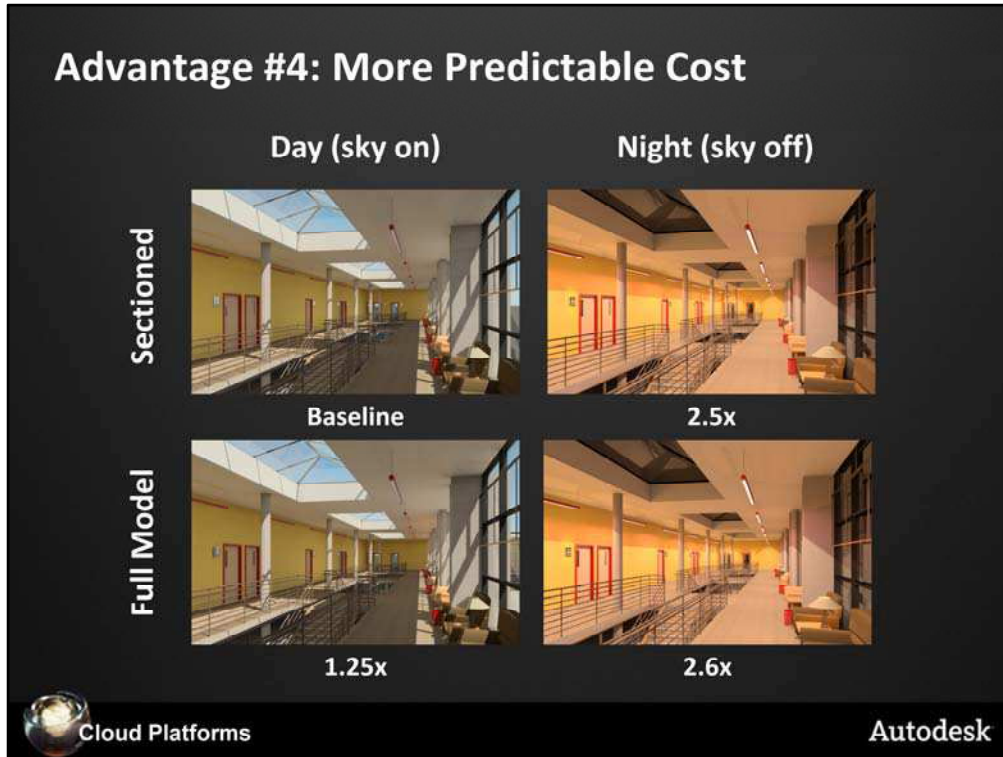
Rendering should have minimal impact on these other applications.



Cloud Platforms

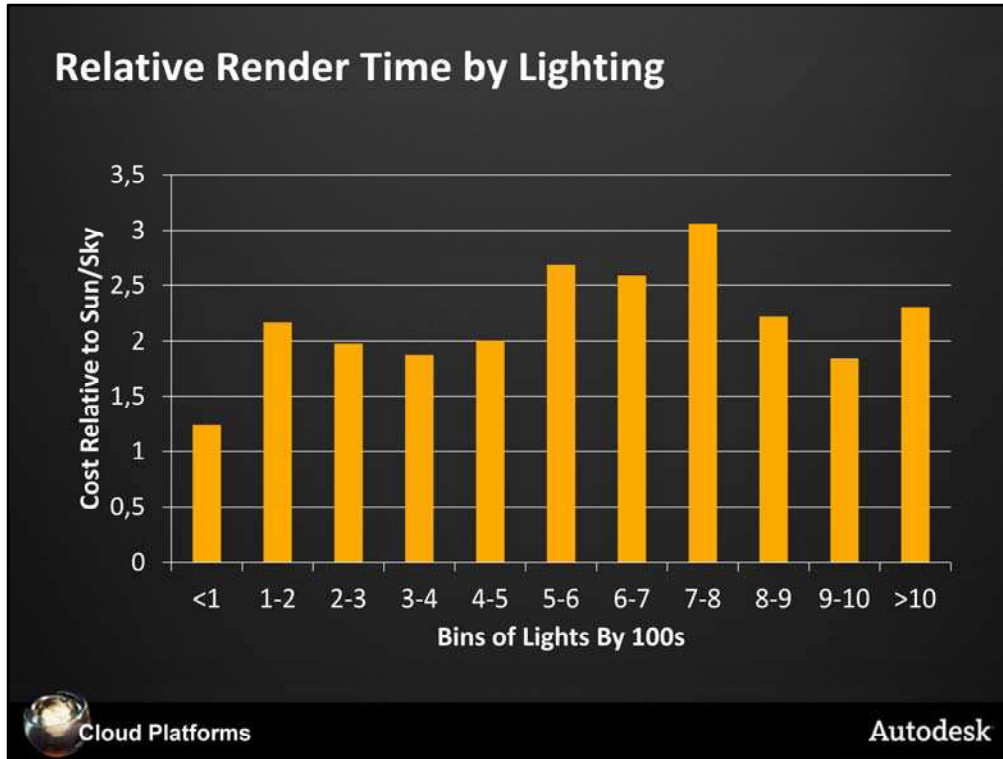
Autodesk

The third advantage of many lights technology is related to the last. Our users are building models for a diverse set of purposes. For example, architects and engineers use our software to make products or a buildings. Rendering these models should not interfere with these other purposes but this is not always true. Making a rendering can require the user to annotate daylight portals, create sectioned models (shown here bottom left) or enable/disable lights. However, the scalability of many lights algorithm bridges this render model/design model gap. The scalability lets our system robustly support “design” size models, reducing user effort and encouraging them to use rendering more.



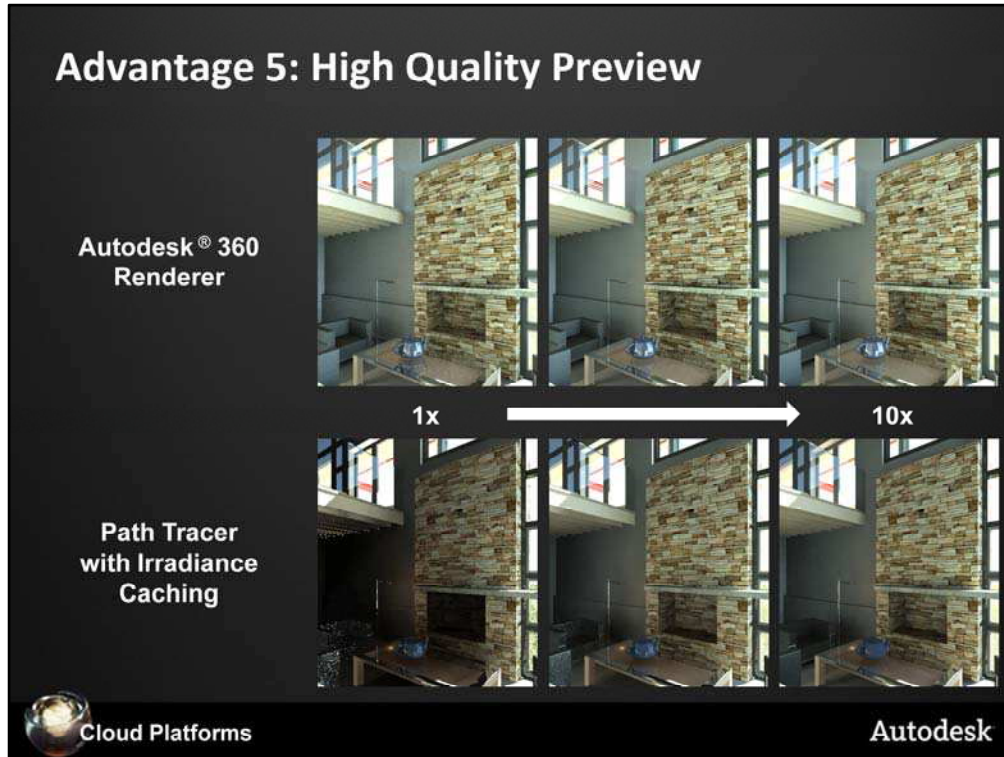
Additionally, the scalability of many lights algorithms makes rendering costs more uniform and predictable. This is important because our users want to use rendering to explore visual design options: lighting, window placements and building layout, for example. But this is harder to do if there is a huge variation in the cost of rendering these options. By making rendering more scalable and uniform, many lights algorithms allow our users to more freely render intermediate designs and enable them to use rendering to inform their designs rather than just to visualization the final choices.

To illustrate this point, I rendered the same scene four times. In the left column, the procedural sun and sky is enabled and in the right it is off. In the top row, the model has been sectioned so that only the visible region is used for simulation while in the lower half the full model is preserved (including many light fixtures out of frame). Of course, costs increase with the increased geometry and lighting but they do so slowly. The increased render time is far outweighed by the time the user saves building the sectioned model or disabling light fixtures.



But one example is not as compelling as a thousand examples. I can continue this uniformity argument to a database of approximately 20K images. Here I plot the relative cost of adding lighting fixtures to scenes. In this sample, it turns out that sun/sky models have the lowest cost so I normalized this plot to show the relative cost of rendering models with only sunlight compared to those with both sunlight and fixtures. The models are grouped into bins of 100 fixtures each. The leftmost bin includes those scenes with less than 100 and the rightmost is scenes with more than 1000. Overall the results show remarkable consistency in the rendering cost across the whole range. *This* is the advantage of a many lights approach.

Advantage 5: High Quality Preview



Finally, the last advantage is one of the biggest: the performance of the algorithm at low quality. This is very critical for our cloud application because we need to offer cheap, fast renderings, useful for intermediate feedback, that are predictive of a long-running, high-quality final result.

This slide compares this cost/quality tradeoff for our system (top) and a path tracer with irradiance caching (bottom). The images on the right are final quality and cost 10x more than the images in the leftmost column. (Note: columns are equal time comparisons between the two renderers; and the underlying material system for the two renderers is not the same so the teapot and table appear slightly different.) Because we can keep the VPL sampling rate the same across this whole set (VPL cost is included in the timings), the many lights solution tends to preserve the lighting quality and appearance across the whole range. However, the path traced solution becomes negatively biased as the irradiance cache sampling rate and density become low.

Problem – Revisited

How to **automatically, efficiently and reliably** produce a large number of physically-accurate renderings in a **predictable** amount of time?

How to you make rendering a service?



Cloud Platforms

Autodesk

To conclude this talk, I return to the problem I discussed at the beginning. How to you make rendering accurate, fast, automatic, efficient and reliable? Or more succinctly: how do you make rendering a service? That is our goal at Autodesk.

Conclusion

- **Many lights algorithms have been crucial to our success.**
- Advantages
 - Robust and well-researched technology
 - Faster, cheaper and more efficient
 - Enable automatic render configuration
 - Inherent amortization across quality levels and scenes
 - Easier for novice users
 - High quality preview images



Cloud Platforms

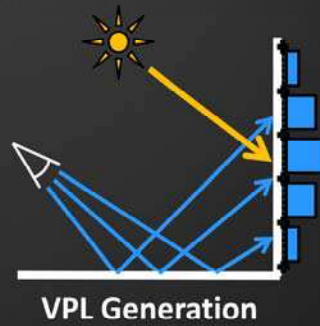
Autodesk

Many lights algorithms begin to make these type of service a reality. Our case study at Autodesk demonstrates that many lights rendering is a robust and well-studied technology ready for production. Moreover we have demonstrated that the technology has many significant advantages. It:

- Is faster, cheaper and more efficient;
- Enables automatic render configuration;
- Provides inherent amortization of costs across design and quality options;
- Makes rendering easier for novice users; and
- Produces useful preview images at very low cost.

Future Work

- VPL Generation
 - Estimates of VPL/VSL error
 - Generalized targeting
- Error and Refinement
 - Quantification of error
 - Faster convergence
 - More efficient trees
 - Representative selection
 - Refinement ordering



Cloud Platforms

Autodesk

Of course, the work is never done. To close I want to highlight two issues that would considerably improve our system. Foremost, general VPL targeting is an unsolved problem. Fundamentally, we need methods to assess the error of a particular set of VPLs and to generalize the targeting process to reduce this error. Second, there seems to be room for further performance improvements in the cut refinement process. One, if we could estimate the absolute error of an intermediate cut, we could begin to quantify the absolute accuracy of many lights methods. Also we want to continue to investigate whether efficiency could be improved by altering the VPL trees, representative selection or cut refinement ordering.

Autodesk®



Cloud Platforms

Autodesk