Real-Time Global Illumination for Dynamic Scenes

Virtual Point Lights

Carsten Dachsbacher
Visualization Research Center
University of Stuttgart

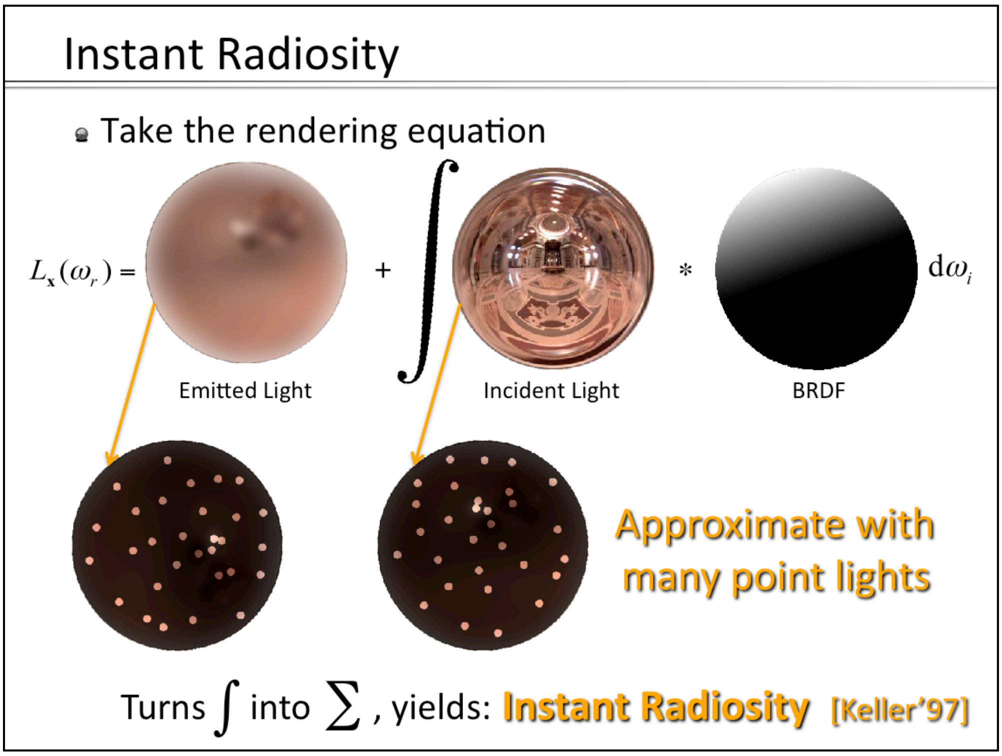
Jan Kautz
University College London

Overview

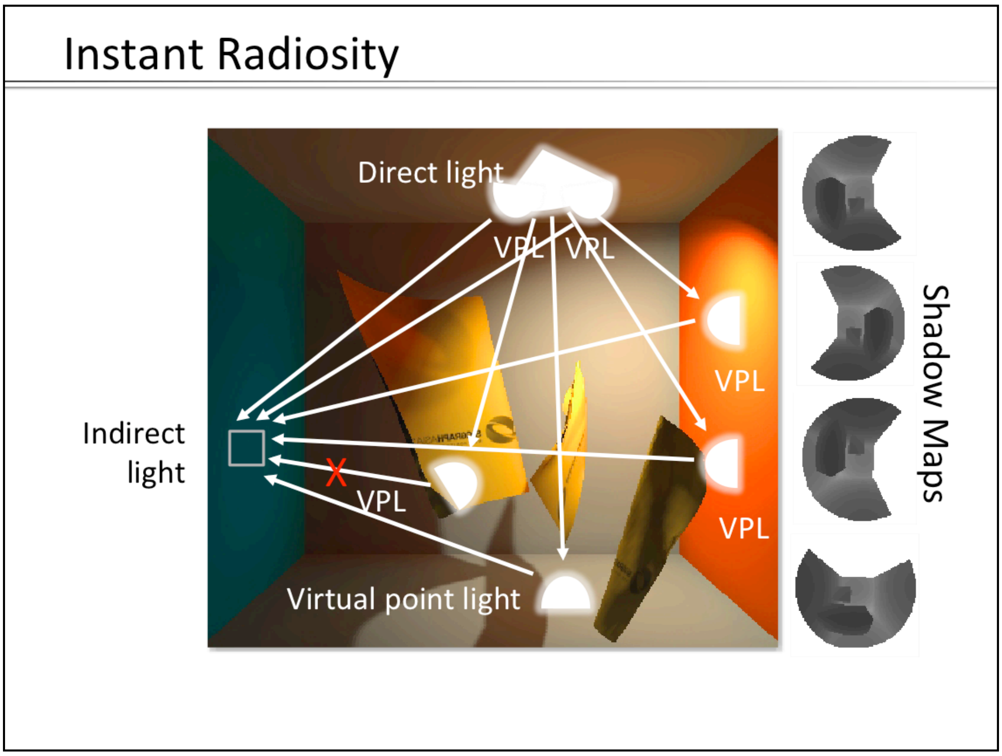
- Instant Radiosity [Keller 1997]
- Incremental Instant Radiosity
- Imperfect Shadow Maps

Instant Radiosity

- Goal is to enable "interactive" global illumination in purely diffuse environments
- Approximate direct and indirect lighting with *virtual point lights (VPL)*.



One way to think of Instant Radiosity is the following. All the direct and indirect illumination is approximated with point lights, which are used for rendering.



Instant radiosity works as follows:

Starting from a **direct light source**, so called **virtual point lights** – VPLs – are created, which **represent the indirect illumination**.

Note that a single VPL is essentially a hemispherical light with a cosine-falloff.

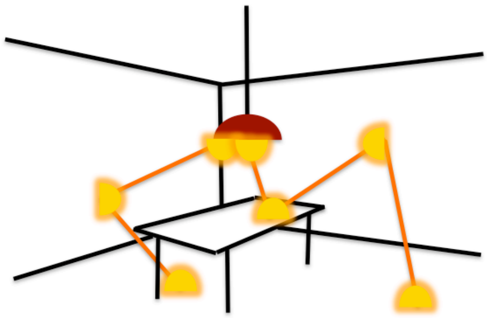
To compute the **indirect illumination at some surface location**, we gather **light from all VPLs** .

However, we still need to **take dynamic visibility into account**. For instance, this path is blocked.

The easiest idea is to use **shadow maps**, even though that is expensive.

Creating VPLs

- High-level idea

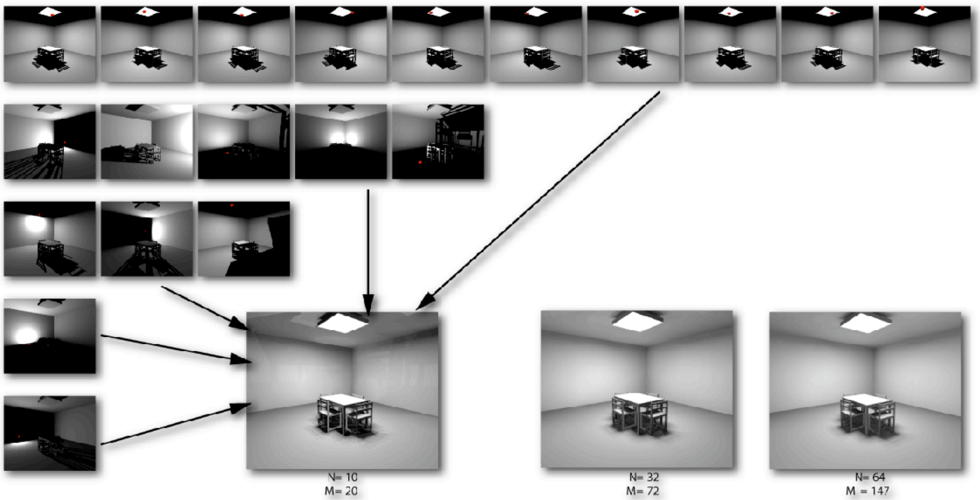


  - Shoot photons from light source
  - Follow them through scene
  - At each hit point, create VPL
    (VPL = point light with cosine distribution)
  - Russian roulette to end path

Derivation is intricate, this is the high-level idea.

VPL Distribution

- VPLs are distributed according to $\bar{\rho}^i$
  ($\bar{\rho}$ = av. reflectance, i = bounce)



Sample direct lighting (L_0), one-bounce indirect (L_1), two-bounce indirect (L_2), etc…

Sum up contributions.

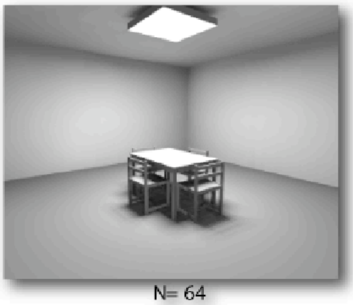
Make sure VPLs are distributed according to average reflectance (raised to the bounce number).

This ensures that there are more VPLs for the direct lighting, and then fewer for the first bounce, and then even fewer for the second bounce, etc.

Rendering with VPLs

- Render scene
  - Illuminated by each VPL, including shadows
    - E.g., use shadow maps or volumes

- Accumulate results:



N= 64
M = 147

Pseudo-Code

```
void InstantRadiosity(int N, double ρ̄)
{
    double w, Start; int End, Reflections = 0;
    Color L; Point y; Vector ω⃗;

    Start = End = N;

    while(End > 0)
    {
        Start *= ρ̄;

        for(int i = (int) Start; i < End; i++)
        {
            // Select starting point on light source
            y = y0(Φ2(i), Φ3(i));
            L = Le(y) * supp Le;
            w = N;

            // trace reflections
            for(int j = 0; j <= Reflections; j++)
            {
                glRenderShadowedScene(N/⌊w⌋ L, y);
                glAccum(GL_ACCUM, 1/N);
                // diffuse scattering
                ω⃗ = ω⃗d(Φb_{2j+2}(i), Φb_{2j+3}(i));
                //trace ray from y into direction ω⃗
                y = h(y, ω⃗);
                // Attenuate and compensate
                L *= fd(y);
                w *= ρ̄;
            }
        }

        Reflections++;
        End = (int) Start;
    }

    glAccum(GL_RETURN, 1.0);
}
```

- First, render paths that immediately end on the light source
- Then, render paths that are reflected once, etc.

Pseudo-code for IR.

DERIVATION

Rendering Equation

- Defined as:

$$L(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_r) + \int_\Omega f(\mathbf{x},\omega_i,\omega_r)L(\mathbf{y},-\omega_i)\cos\theta_i \, \mathrm{d}\omega_i$$

- Using operator R:

$$(\mathbf{R}L)(\mathbf{x},\omega) = \int_\Omega f(\mathbf{x},\omega_i,\omega)L(\mathbf{y},-\omega_i)\cos\theta_i \, \mathrm{d}\omega_i$$

- Equation becomes:

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

Let's go back to the rendering equation and derive Instant Radiosity (IR).

Rendering Equation

- Solving for L:

$$L = L_e + \mathbf{R}L$$
$$(1-\mathbf{R})L = L_e$$
$$L = (1-\mathbf{R})^{-1} L_e$$
$$L = (1+\mathbf{R}+\mathbf{R}^2+\mathbf{R}^3+\ldots)L_e$$

- Radiance towards eye =
  - direct light from light source
  - plus light reflected once,
  - plus light reflected twice, …

Using the operator notation, we know that radiance towards the eye = …

Instant Radiosity

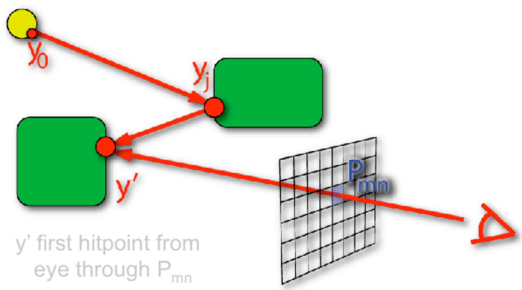
- Assume BRDF is diffuse

$$f(\mathbf{x}) = f(\mathbf{x}, \omega_i, \omega_r) = \frac{k_d}{\pi}$$

- Rewrite rendering equation with *explicit* sampling of ***all*** possible paths (with length j=0, j=1, j=2, …)

IR assumes diffuse BRDFs and explicitly samples all possible paths.

Instant Radiosity



- Integrate over pixel P_{mn}
- Sum over all paths with length j
- Integrate over all paths Ω^j (ΩxΩx…) with length j
- Integrate over light source S_e (positions y_0)

$$L(m,n) = \frac{1}{|P_{mn}|}\int_{P_{mn}} L_e(y', P - y') \; + \frac{1}{|P_{mn}|}\int_{P_{mn}} \sum_{j=0}^{\infty}\int_{\Omega^j}\int_{S_e} p_j(y_0,\omega_0,\ldots,\omega_j)\cdot V(y_j,y')f_d(y')\frac{\cos\theta_j\cos\theta'}{|y_j - y'|^2}\,dy_0 d\omega_0 \ldots d\omega_j dP$$

First integral part of sum: all light emitted from y' through Pixel

Second part of sum:

-Integrate of pixel

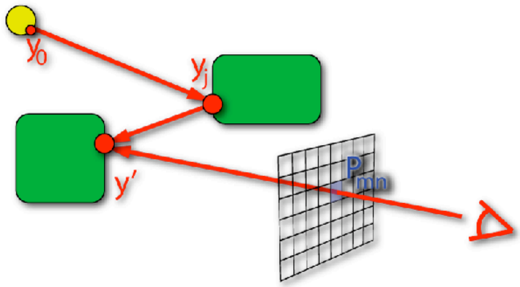
-Sum over all path lengths j

-Integrate over all paths of length j, p_j(…) is assumed to be _valid_ paths!

-Integrate over all light source (starting) positions

-Note that: V() is the visibility of between y_j and y', i.e., y_j is a VPL and y' are locations visible in screen-space.

Instant Radiosity



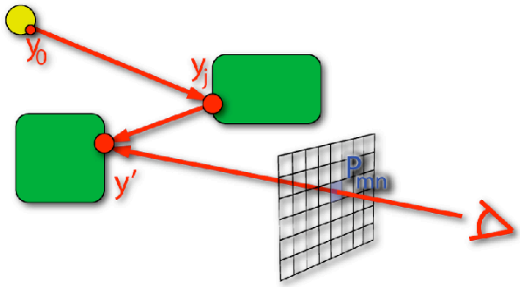
- Radiance after j reflections

$$p_j(y_0,\omega_0,\ldots,\omega_j) := L_e(y_0)\prod_{l=1}^{j}\left(\cos\theta_{l-1} f_d(y_l)\right)$$

[Assumes valid paths (all y_l are mutually visib.)]

The definition of p_j(): essentially the radiance after j reflections (assuming valid paths).

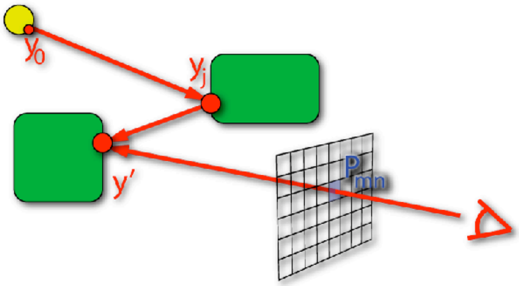
Instant Radiosity



- Now, we importance sample those paths
  - Create paths of different lengths (y_0 to y_j)
  - All y_j act as virtual point lights

Now sample those paths.

Instant Radiosity



- Sampling the paths
  - Start with random point y_0 on light source
  - Follow them (using IS) through the scene
  - Every hitpoint y_j becomes a VPL

When and how to end a path?

- We start with *N* point lights on the area light
  - Reflect all *N* of them? Paths never end then…
  - Russian Roulette?
- Assume surfaces are not far from average reflectivity $\overline{\rho}$
  - Enables use of fractional absorption
  - Of the initial *N*: $\overline{\rho}N$ get reflected (1st bounce)
  - 2nd bounce: $\overline{\rho}^2 N$ get reflected, etc…
  - Average path length: $\frac{1}{1-\overline{\rho}}N$

Instead of individually deciding which paths to continue, use fractional absorption based on the average reflectivity of the scene.

DERIVATION END

Incremental Updates

- Original paper proposes incremental updates for real-time speeds
  - Keep last N VPLs (and the rendering for each)
  - Replace oldest one with a new VPL
  - Accumulate

⛔ Has problems for dynamic scenes: illumination lags behind

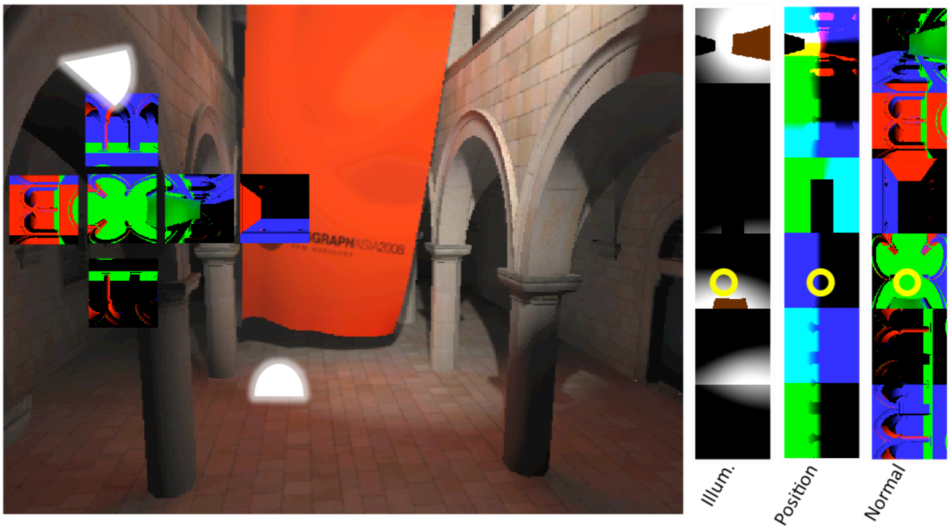
Practical Considerations

- How many total VPLs needed?
  - Artifact-free results:
    several hundred VPLs
  - Temporal coherent results:
    about 1000 VPLs

(just rule of thumb!)

GPU-Based VPL Generation

- Original VPL creation requires ray-tracer
- For single bounce (point light), can be done on GPU



Classic Instant Radiosity requires a ray-tracer to follow photons through the scene.

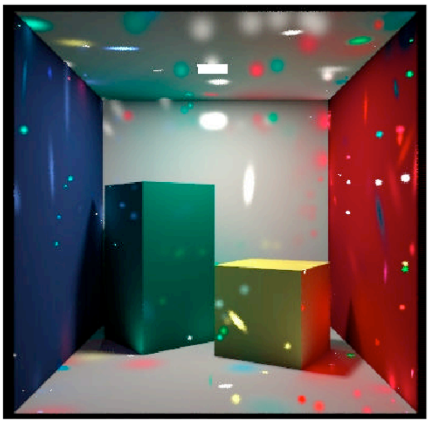
However, for a single bounce this can be easily done on the GPU.
To this end, the scene is rendered as seen **from the light source** into an **omnidirectional map**.
In particular, **positions**, **normals** and **direct lighting** are rendered.

Then all textures are sampled in parallel at a number of **random points**, which are **importance sampled according** to the **brightness** of the direct illumination.
For instance, this gives you this VPL here.

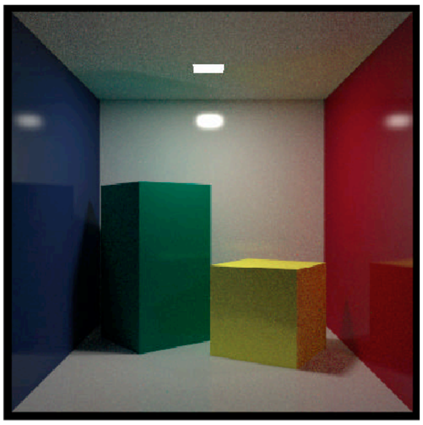
This is essentially the first step of reflective shadow mapping, what was shown before.

Instant Radiosity Problems

- Difficult to extend to specular surfaces



Instant Radiosity



Path Tracing

VPLs become visible for highly specular surfaces.

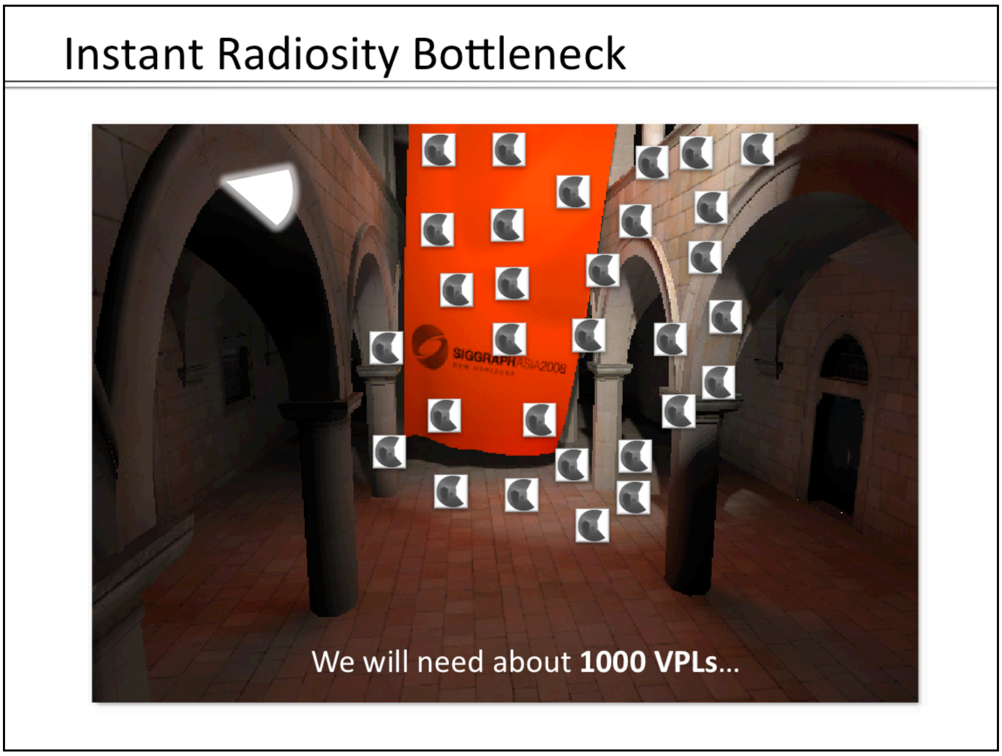
Summary

- Instant Radiosity
  - Approximate indirect illumination with VPLs
  - Accumulate contributions from VPLs

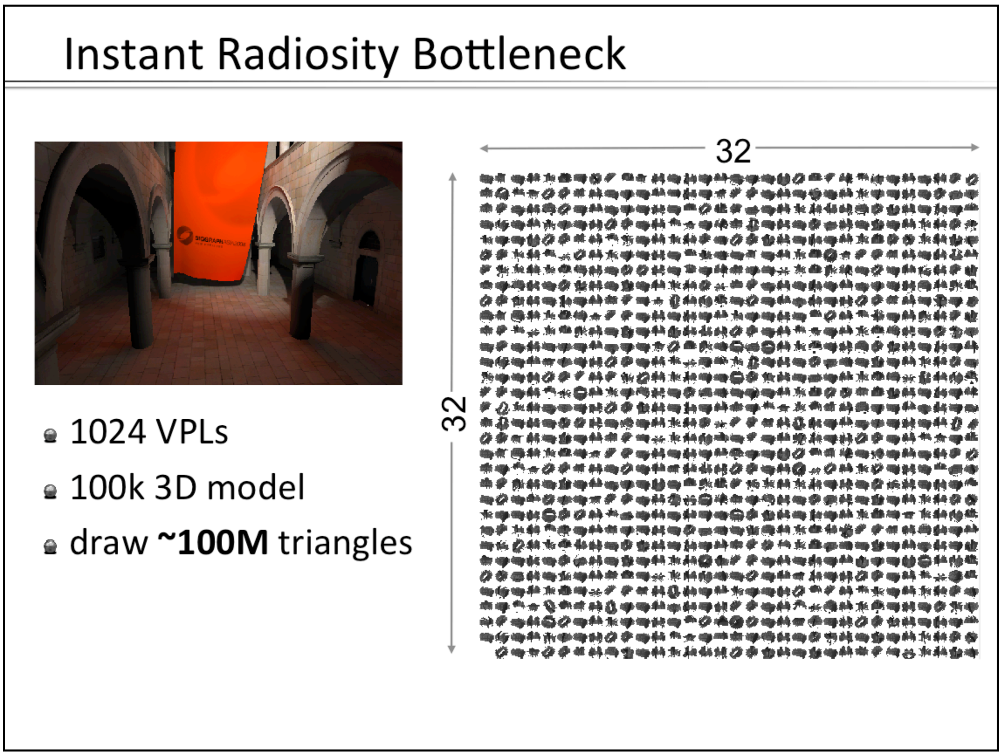
- (+) Easy to implement on GPUs
- (−) Assumes diffuse scenes (or very low-glossy)
- (±) Fast only for small scenes (due to the need for creating many shadow maps/volumes)

Overview

- Instant Radiosity
- Incremental Instant Radiosity [Laine et al. 2007]
- Imperfect Shadow Maps



In practice, IR needs to render a large number of shadow maps, which is very costly.



In fact, the **shadow map generation** is the **bottleneck**:

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

This means drawing **100 million triangles** to fill the 1000 shadow maps .

Incremental Instant Radiosity

- Goal: real-time indirect illumination
  - Essentially static static (plus dynamic receivers)
  - Single bounce
  - Allow light to move

Incremental Instant Radiosity allows semi-dynamic scenes with moving lights.

Incremental Instant Radiosity

The Recipe:

Old ingredients

- Instant radiosity with single bounce
- Interleaved sampling
- Paraboloid shadow mapping

New ingredients

- **Reuse of VPLs**

The ingredients are known but one: reuse of VPLs for moving lights.

VPL Reuse

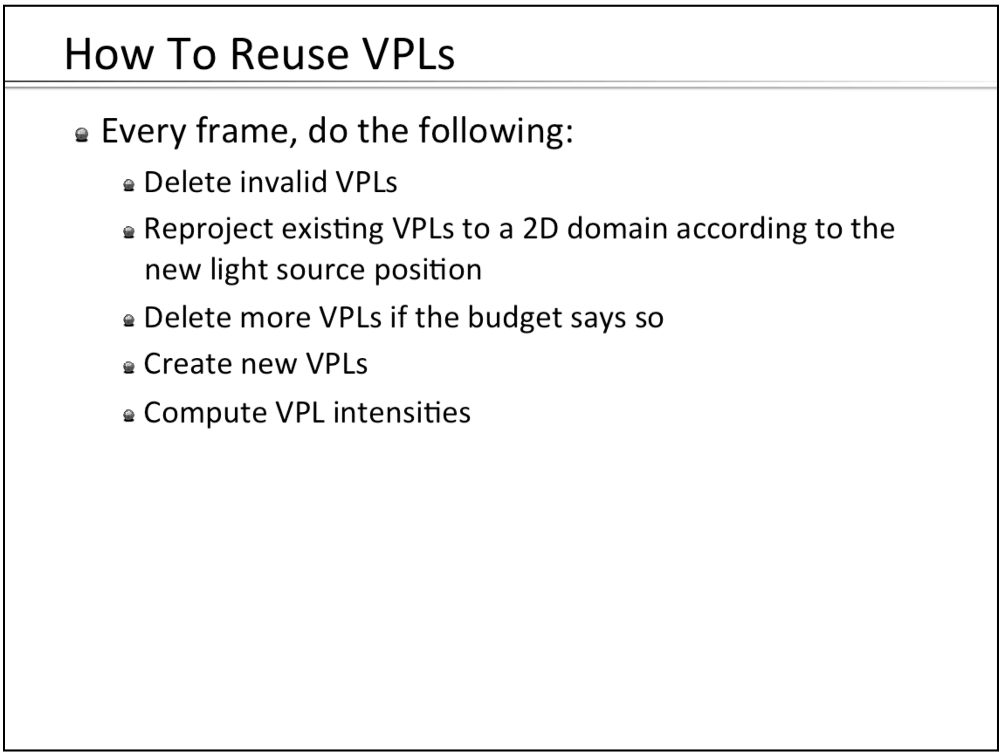
- Reuse VPLs from previous frame
  - Generate as few new VPLs as possible
  - Stay within budget, e.g. 4-8 new VPLs/frame

+ Benefit: Can reuse shadow maps!

! Disclaimer: Scene needs to be static

§ Note: Illumination *does not* lag behind

The main idea is to reuse VPLs from previous frames (assuming static geometry).



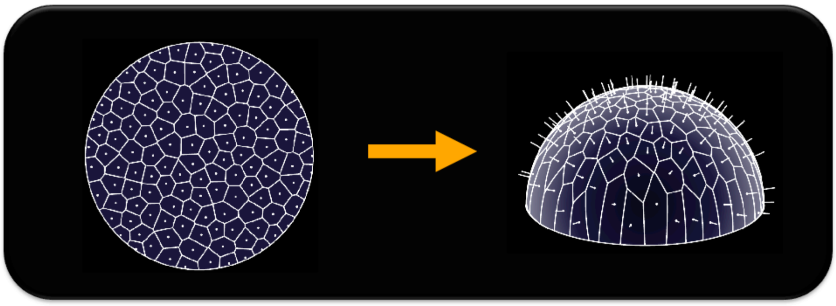
The basic algorithm is simple:

2D Domain for VPLs

- Let's concentrate on 180° cosine-falloff spot lights as main light source for now
- Nusselt analog

  Uniform distribution in unit disc

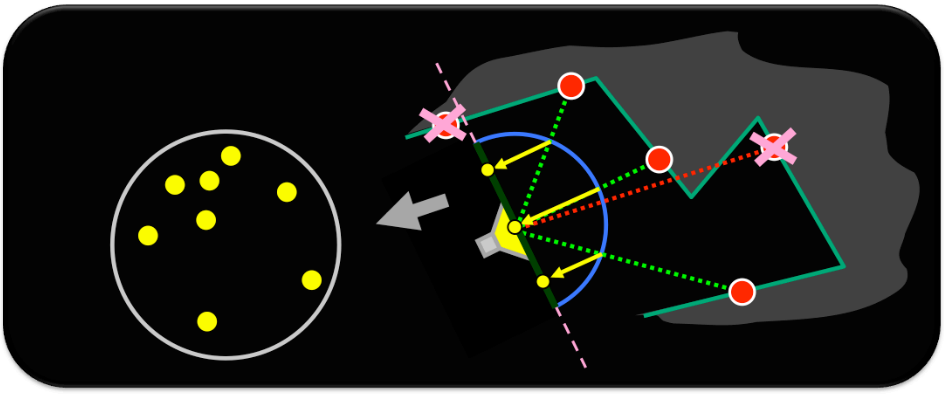
  = Cosine-weighted directional distribution



Let's assume for now that our main light source is a hemispherical spot-light with a cosine-fall off.

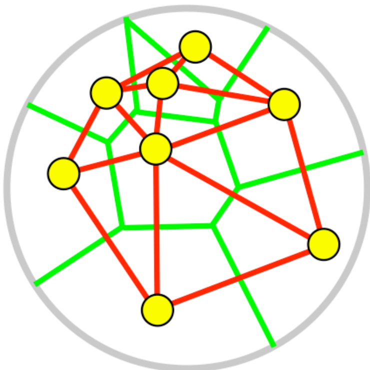
Reprojecting VPLs

- So we have VPLs from previous frame
- Discard ones behind the spot light
- Discard ones behind obstacles
- Reproject the rest



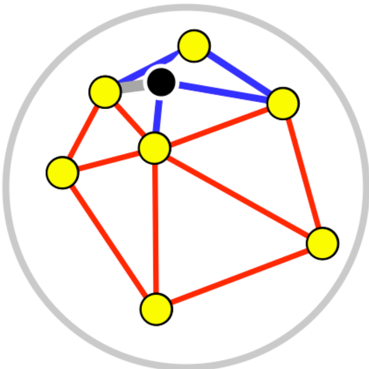
Spatial Data Structures

- Compute Voronoi diagram and Delaunay triangulation for the VPL point set



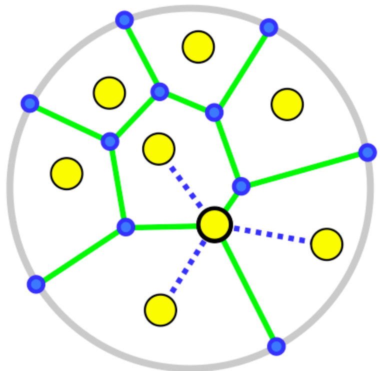
Deleting VPLs

- Greedily choose the ”worst” VPL
  - = The one with shortest Delaunay edges



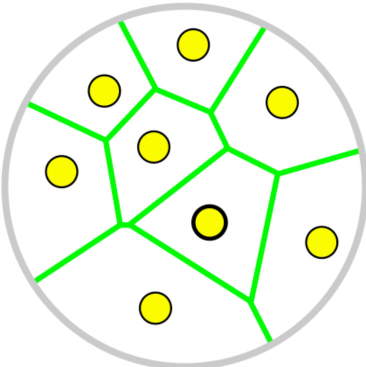
Generating New VPLs

- Greedily choose the ”best” spot
  - = The one with longest distance to existing VPLs
- Re-project VPL into scene (use sh. map from light)
- Create shadow map for new VPL



Computing VPL Intensities

- Since our distribution may be nonuniform, weight each VPL according to Voronoi area



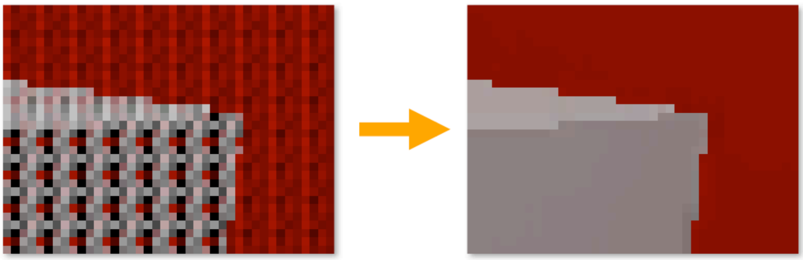
Omnidirectional Lights

- Perform all 2D domain actions on the surface of unit sphere

Interleaved Sampling (G-Buffer)

- Accumulating from hundreds of VPLs expensive

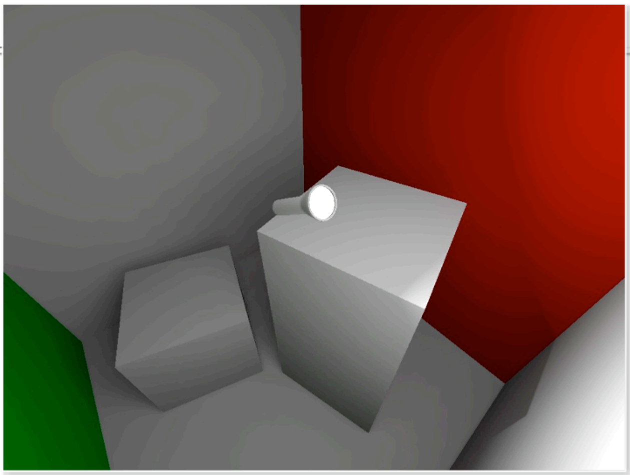
- Interleaved Sampling:
  - Reduces the number of shadow map lookups per pixel
  - For each pixel, use a subset of all VPLs
  - Apply geometry-aware filtering



Results

- 256 VPLs in all scenes
- Budget: 4-8 new VPLs per frame
- GeForce 8800 GTX

Cornell

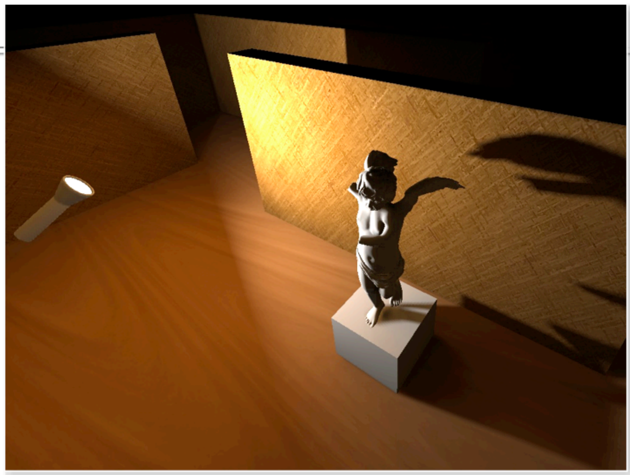


Triangles:

original 32
tessellated 4.4k

Resolution	Time (ms)	FPS
1024×7680	13.9	65.1
1600×1200	26.8	29.7

Maze



Triangles:

original 55k
tessellated 63k

Resolution	Time (ms)	FPS
1024×7680	15.6	49.2
1600×1200	28.6	28.5

Sibenik

Triangles:

original 80k
tessellated 109k

Resolution	Time (ms)	FPS
1024×7680	17.0	48.6
1600×1200	30.1	25.9

Discussion

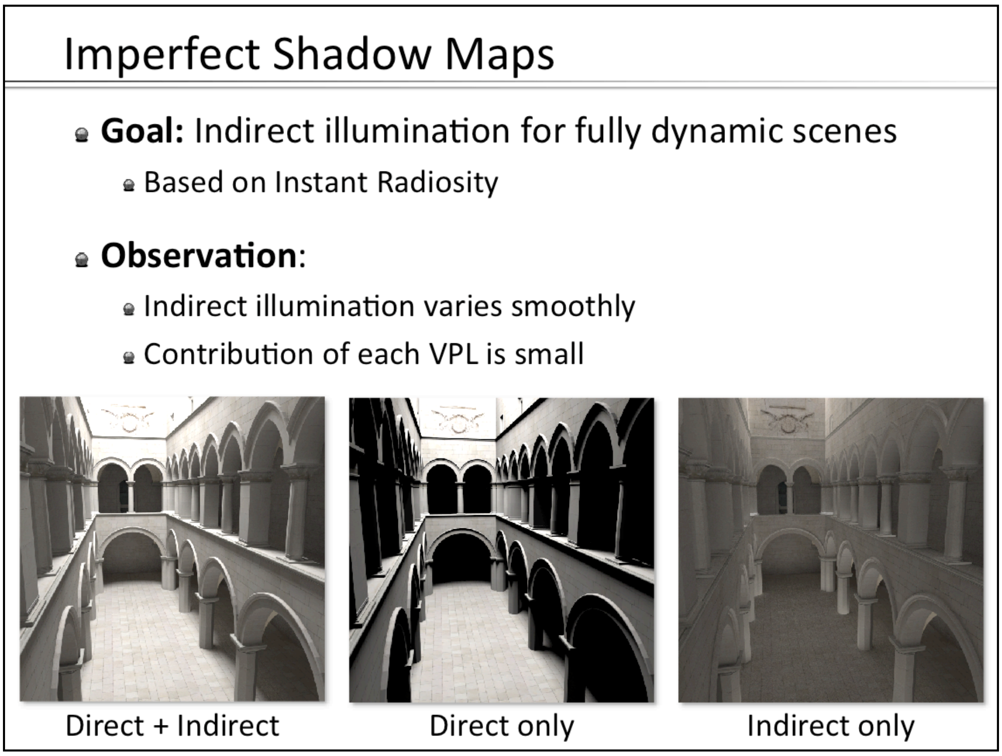
- Not full GI
  - Well, one *could* use entire light paths, but that would lead to many faint VPLs
- Diffuse surfaces only
  - Slightly glossy should work OK
  - Truly glossy won't work (same as Instant Radiosity)
- Not view-dependent
  - Distributing VPLs should be based on visual importance
- Dynamic scenes non-trivial
  - The shadows are wrong for less than a second when the scene changes...

Strengths

- No precomputation
- Dynamic objects can receive indirect light
- Real-time performance
- Simplicity
- No temporal aliasing (VPLs are consistent)

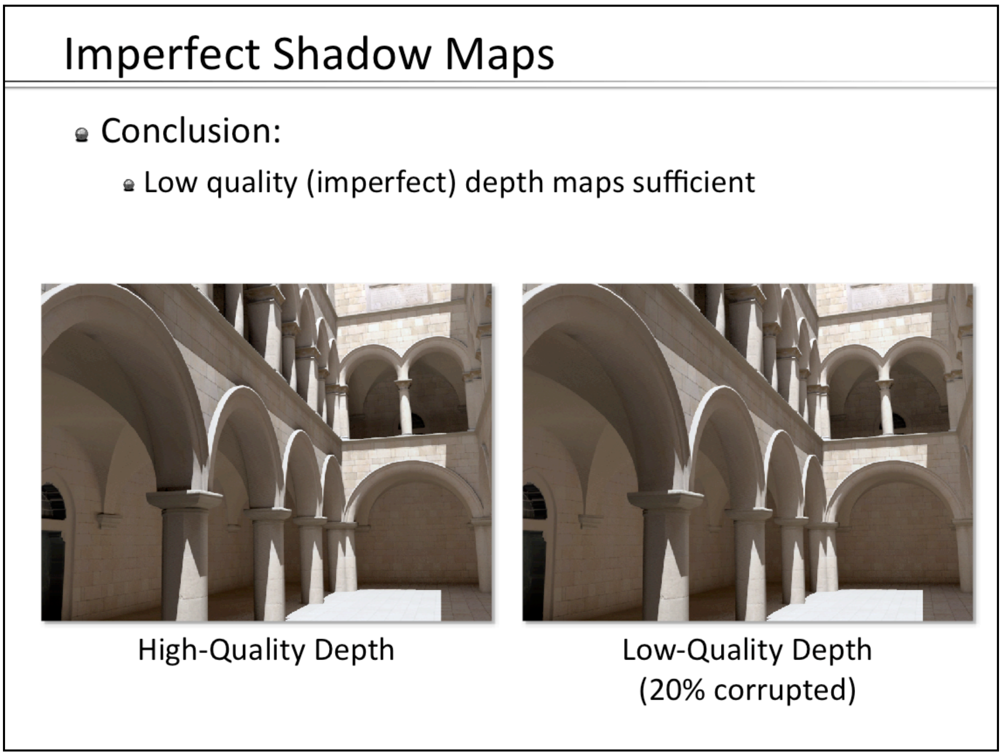
Overview

- Instant Radiosity
- Incremental Instant Radiosity
- Imperfect Shadow Maps [Ritschel et al. 2008]



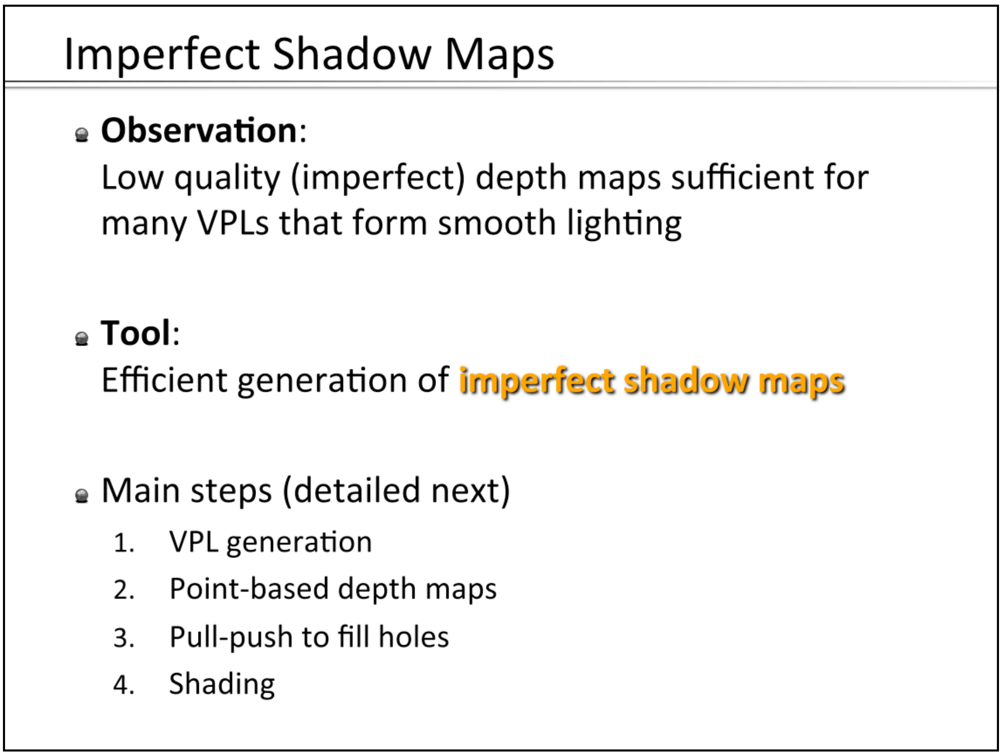
Imperfect shadow maps are based on **two key observations**.

1. indirect lighting **varies smoothly** in most scenes.
2. the **individual contribution** of each VPL is **small**.



Which leads to the conclusion that it is sufficient to use many **low quality depth maps to determine visibility in indirect illumination**, as errors tend to average out.

Here you can see an example, where **using low-quality depth** maps does not impact the final rendering much.

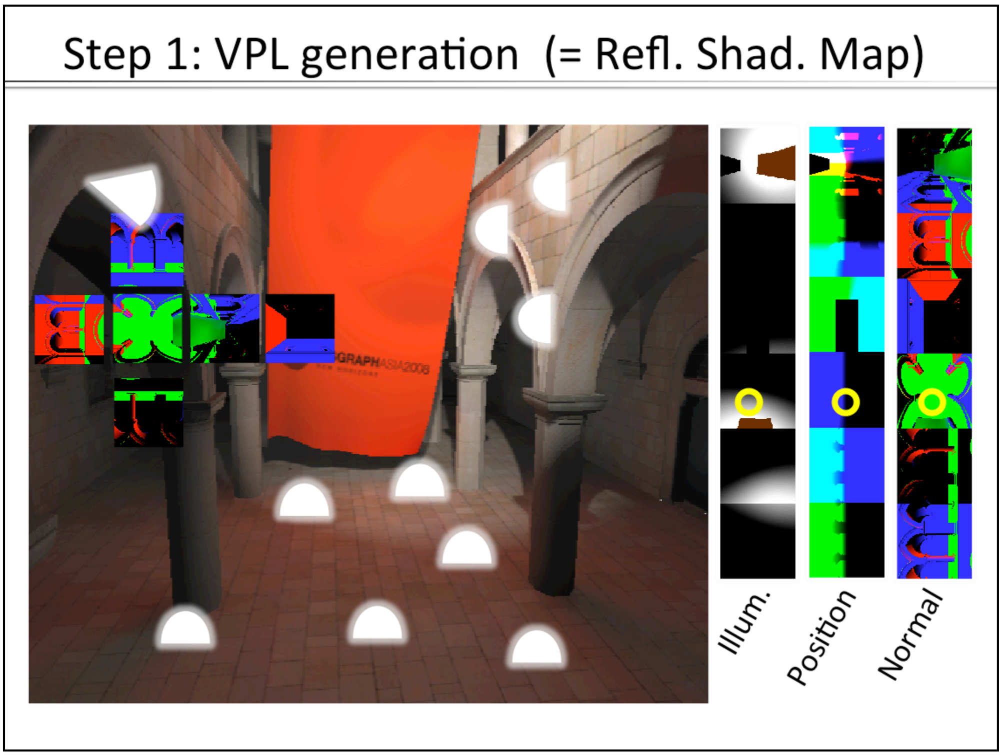


The main ingredient of ISM is to allow **imperfection** when creating a depth map, which enables a much more efficient generation.

The algorithm consists of **4 steps**:

1. VPL generation,
2. Point-based depth map generation
3. A pull-push operation to fill holes from point rendering
4. Shading

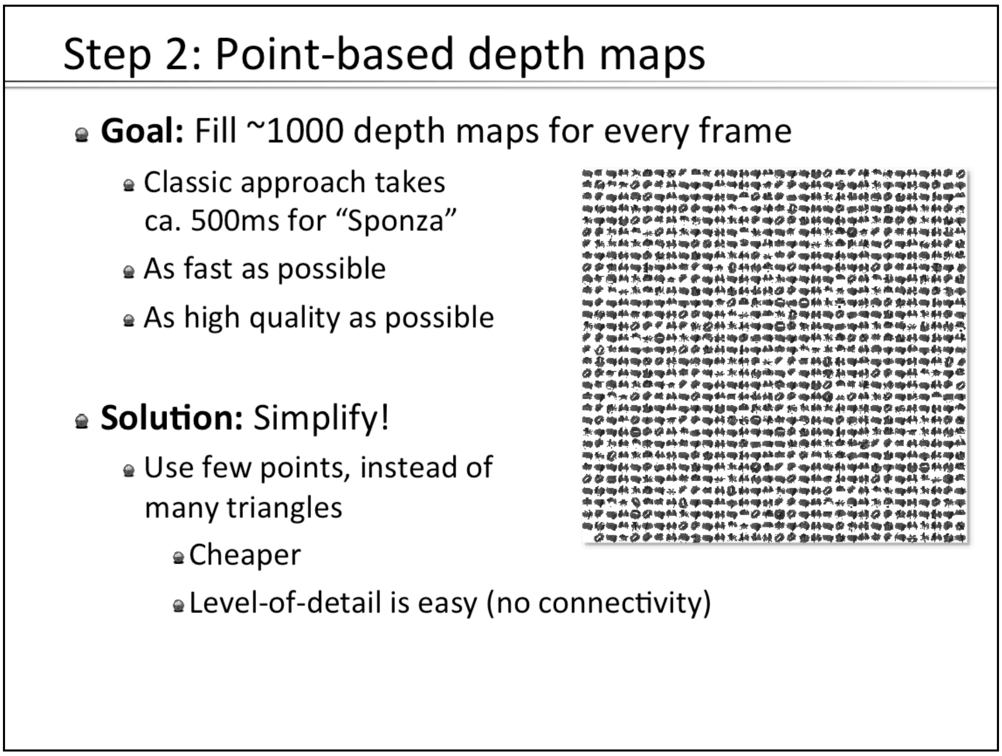
I will detail all four steps now.



There **should be no VPLs** where there is **no direct light** and there **should be VPLs** where there **is direct light** .

To achieve this, the scene is rendered as seen **from the light source** into an **omnidirectional map**.
In particular, **positions**, **normals** and **direct lighting** are rendered.

Then all textures are sampled in parallel at a number of **random points**, which are **importance sampled according** to the **brightness** of the direct illumination.
For instance, this gives you this VPL here.



Recall, that our goal is to **generate as many depth maps as possible**.

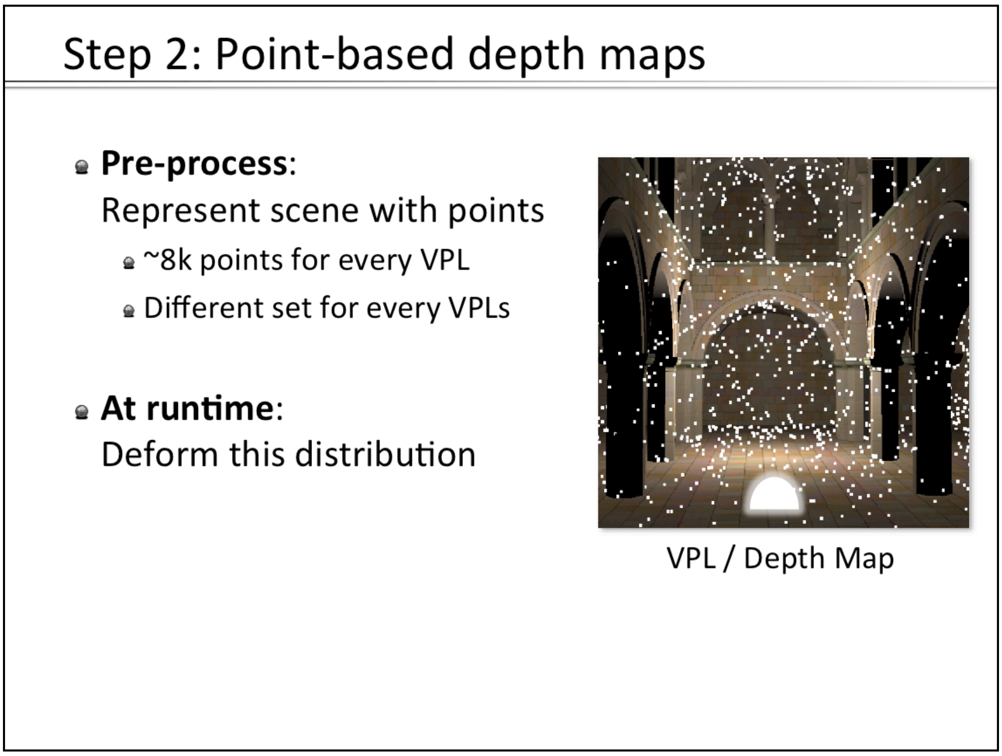
Using classic depth maps for this, takes around **half a second** for the Sponza scene.

We want it much **faster**, but as **high-quality** as possible.

We will do this by **simplification**.

We will draw **a small number of points** instead of **a large number of tris**, which is much **cheaper**.

Also **LOD** for points **is very simpler**, because they **don't require connectivity**.

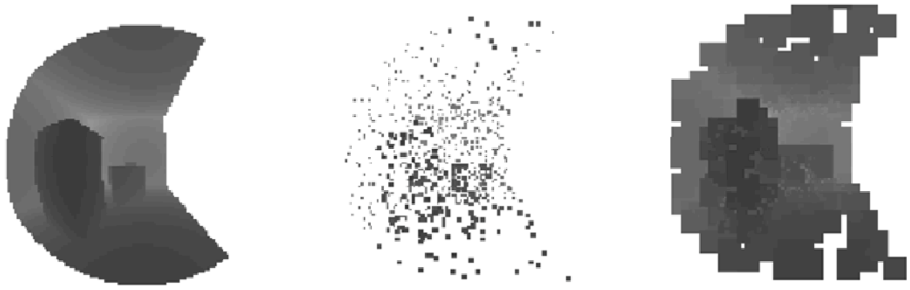


At startup, we **approximate** the surfaces with a set of points.
Each VPL has it’s **own different set of points**; typically, we use **8k points** per VPL.

At run-time we **deform** this distribution according to **surface deformations**.

The image to the right visualizes the point set for a single VPL, and as you can see it’s quite sparse.

Step 3: Pull-Push



Classic
Triangles

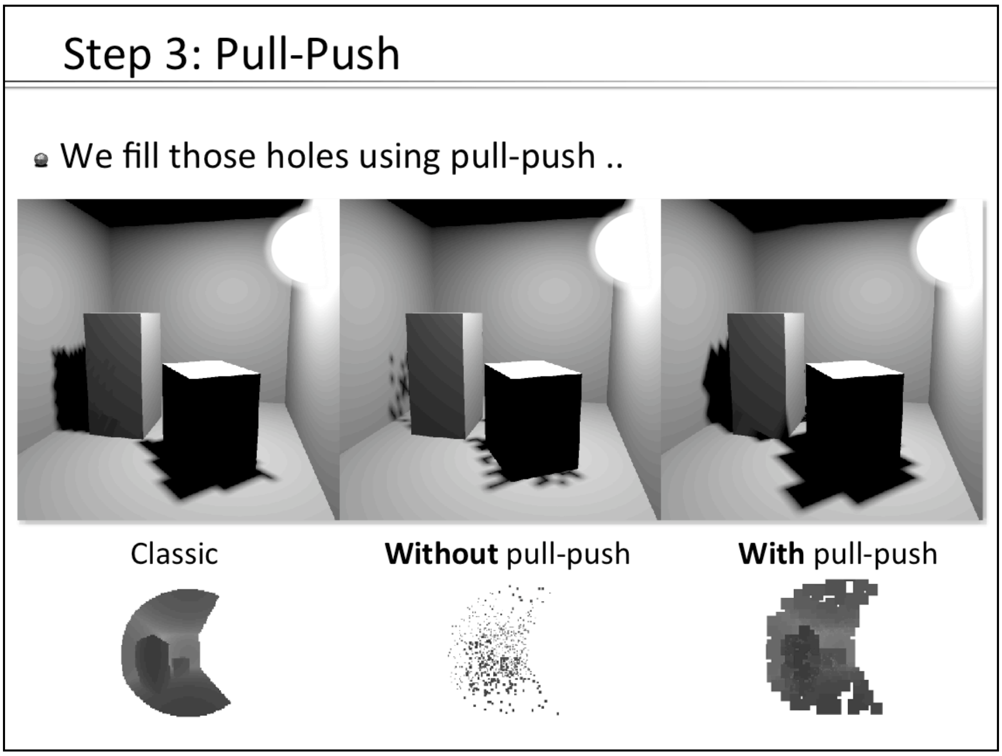
Imperfect
Smaller points
Fewer points

Imperfect
Pull-Push
Hole Filling

Here’s a **classic** shadow map with **triangles** compared to an **imperfect** shadow map with **points**. There are quite a few holes.

Using a process called pull-push, we **fill holes**, and then the maps are quite similar**.**

Pull-push is essentially a **hierarchical** method to fill holes, essentially averaging nearby depth values.



Here we show the imperfect shadow of an individual VPL.

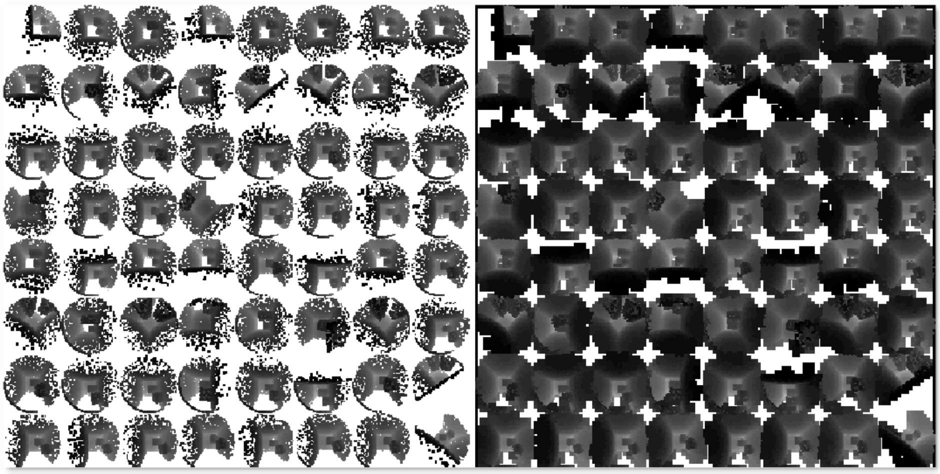
A depth map without pull-push will have **light leaks**, that are **fixed** by pull-push.

At least mostly, of course, there are still **some errors** in the depth maps.

However, since we accumulate the result of many VPLs, and the errors are uncorrelated, they tend to average out.

Step 3: Pull-Push

- .. on all depth maps in parallel.

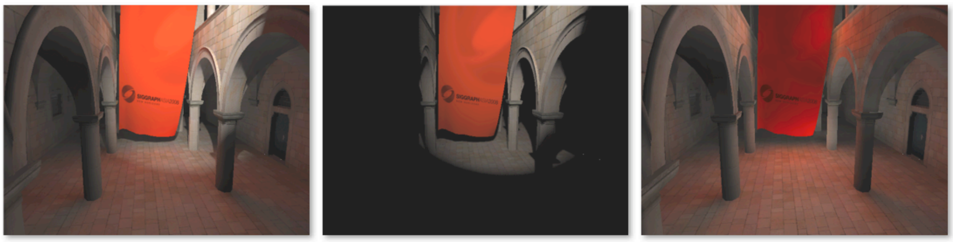


Without pull-push **With** pull-push

We can do this pull-push step on **all depth maps in parallel**, as we work in texture space.

Step 4: Shading

- Separate direct and indirect
  - Render separately



Direct + Indirect | Direct only | Indirect only

- *Direct*: standard techniques
- *Indirect*: gather from VPLs with lookup into ISM

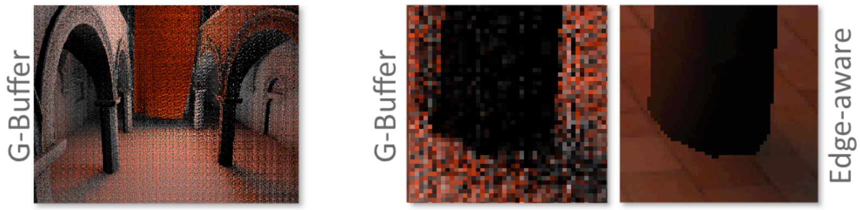
The **rendering of direct and indirect lighting is separated**, like in most current methods.

For the **direct** illumination, we use **standard methods**.

For the **indirect**, we **accumulate light** from all VPLs, with a **visibility lookup** into the ISMs.

Step 4: Shading

- Indirect: Interleaved sampling, geometry-aware blur (same as incremental instant radiosity)



Interleaved sampling is used for the indirect illumination,

i.e. not every pixel is lit with every VPL, but only, say, 16 random ones out of 1024 VPLs for every pixel (in order to save computation).

Doing so, will result in noise, that is blurred away using a **geometry-aware blur filter**.

Results: Quality (*ISM, 11 fps*)

Time for some results. Our method is used to render this at 11 FPS, and it looks quite nice.

You can see **color bleeding** as well as **indirect shadows**.

But how does it compare to a reference solution?

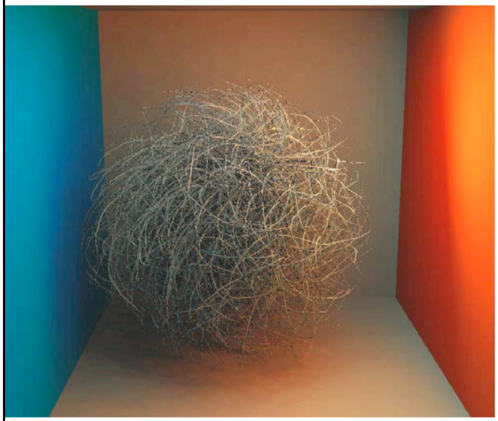
Results: Quality (*Reference, hours*)

This is a Monte Carlo reference rendering.

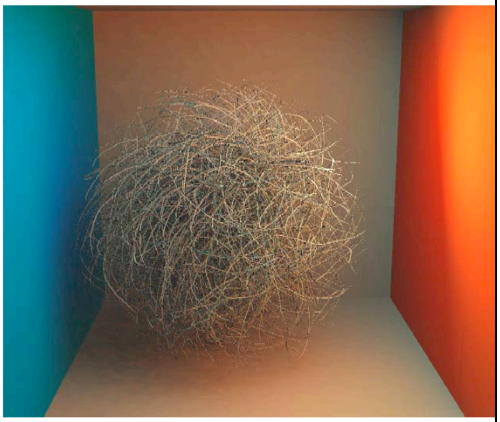
There are some differences. But remember this is an **extreme case**, where **indirect illumination dominates**.

In more "**normal**" scenes (**without spot lights**), the differences are then almost **indistinguishable**.

Results: Quality



Reference *(hours)*



Imperfect Shadow Maps *(15Hz)*

Results: Performance

- “Christo’s Sponza”
  - 70k faces, dynamic
  - 1024 VPLs
  - 256x256 depth maps
  - 8k points each

- Breakdown
  - 7 ms VPL generation
  - 44 ms ISM
  - 8 ms Pull-push
  - 15 ms Rendering
  - 4 ms Blur (G-Buffer)
  - 11 ms Direct light

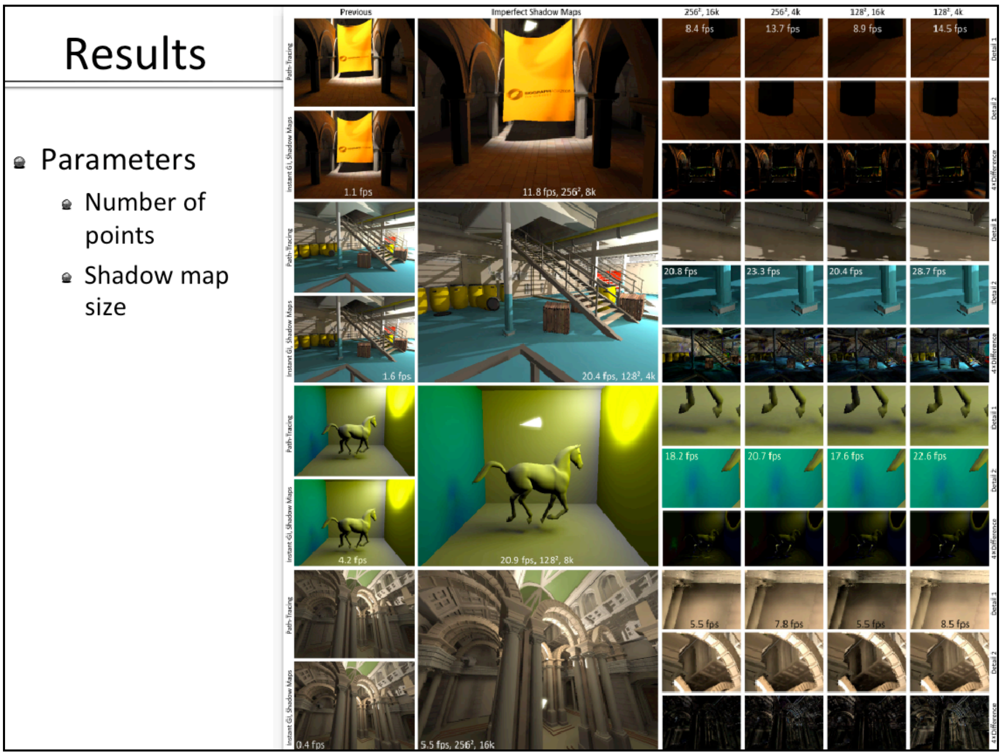
- Total
  - 89 ms frame time
  - 11 frames / s

Here is a **performance breakdown** of the „Christo’s Sponza“ scene.

It has **70k triangles** and was rendered **using 1024 VPLs**, with a shadow map of **256x256** each. **8k points** are splatted into each individual depth map.

The ISMs are generated in 44 ms. Generating classic shadow maps instead, takes around **10 to 15 times longer** for this step, because 70k triangles would be drawn per depth map.

This results is more than **11 fps**, on a Geforce 8800 GTX, around **ten times** faster than normal instant radiosity with classic shadow maps.



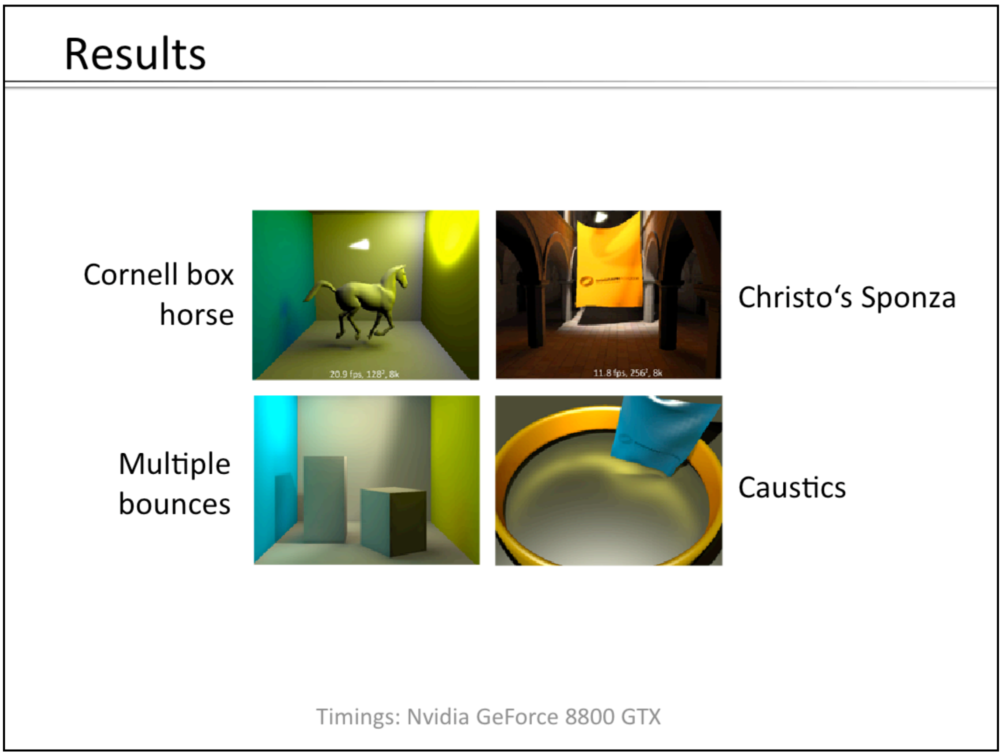
There are **two** main parameters that can be **tweaked**: the **number of points** and the **shadow map size**.

We have experimented with these, and the findings are not surprising:

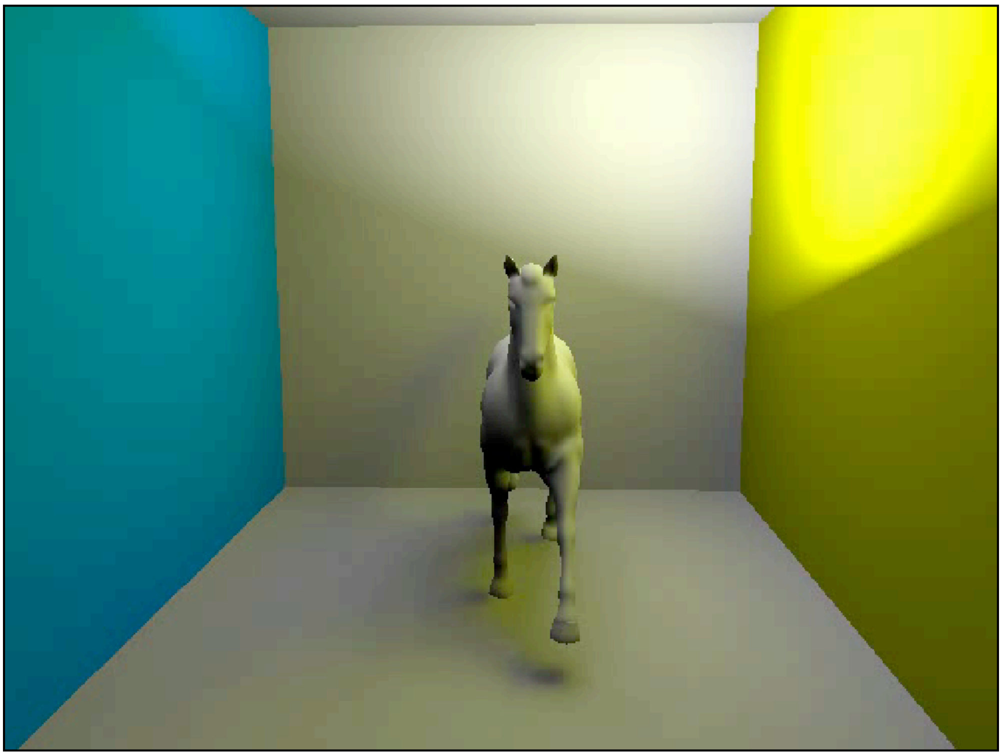
More points per VPL yields **higher quality**, and

larger shadow maps are **better**, if there are sufficient points available.

In general **128^2 or 256^2** shadow maps with **8k points** yields good results.



Some ISM results, which range from diffuse bounces in a Cornell box to complex scenes, including multiple bounces, arbitrary local area lights, natural illumination to caustics.



In this example, running at 20 fps, we placed **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**.

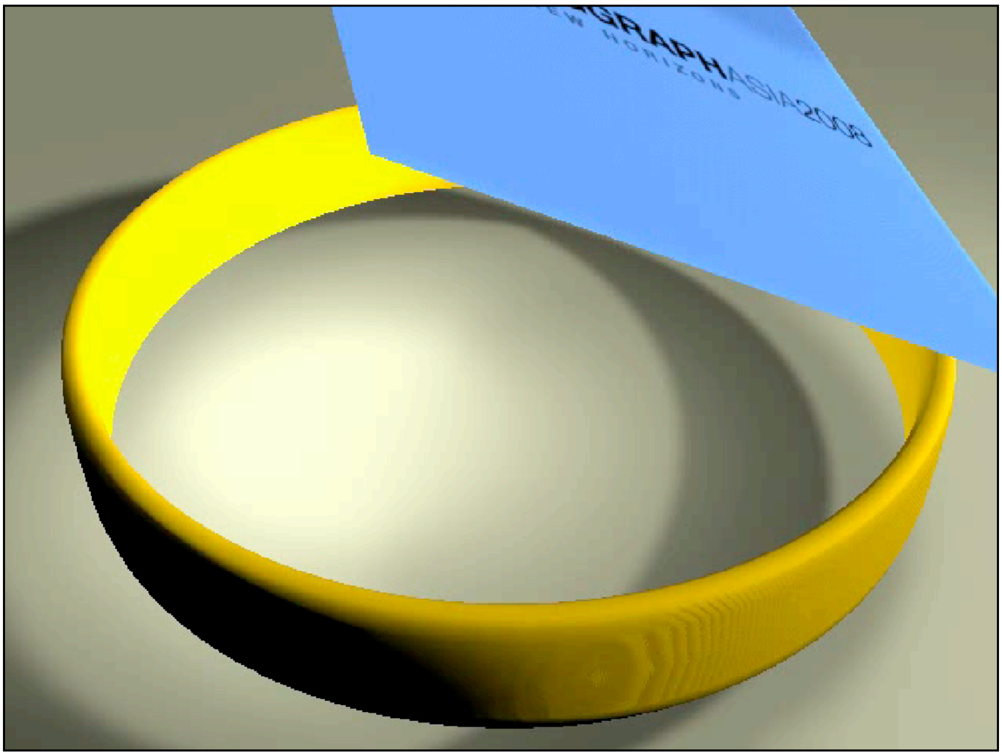
Despite the fundamental changes, in indirect lighting there is **no flickering**.

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

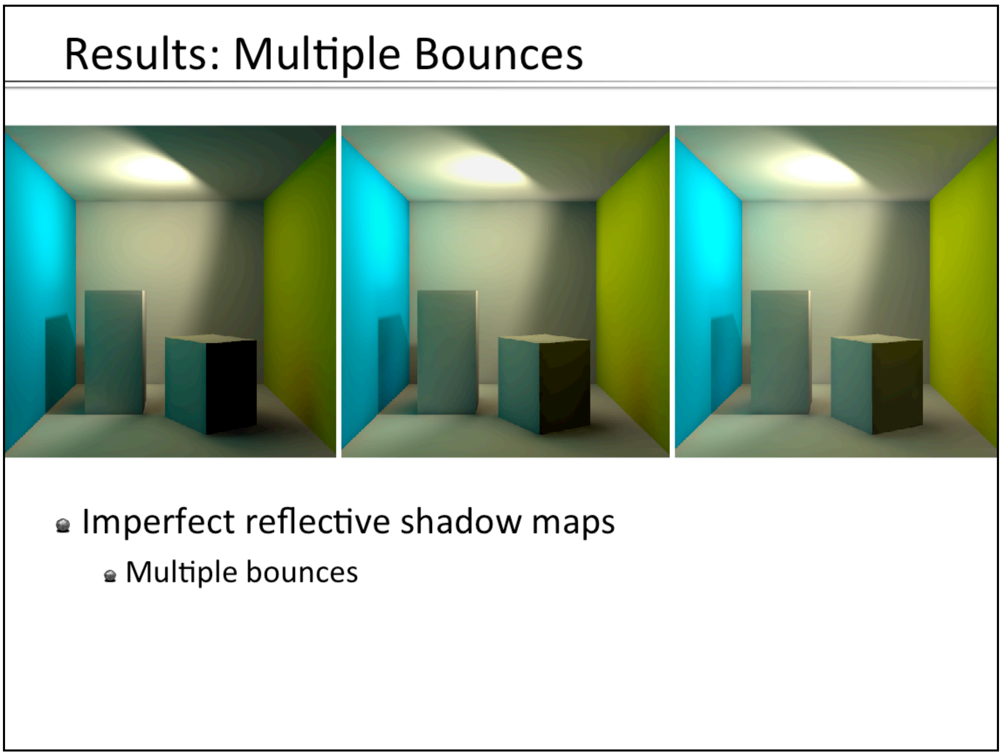
To achieve sufficient **temporal coherence**, we need **1024 VPLs** in this example.

Note, how the bounced light color **changes drastically** when the cloth is moving. Also note, the **indirect shadow** from the columns.

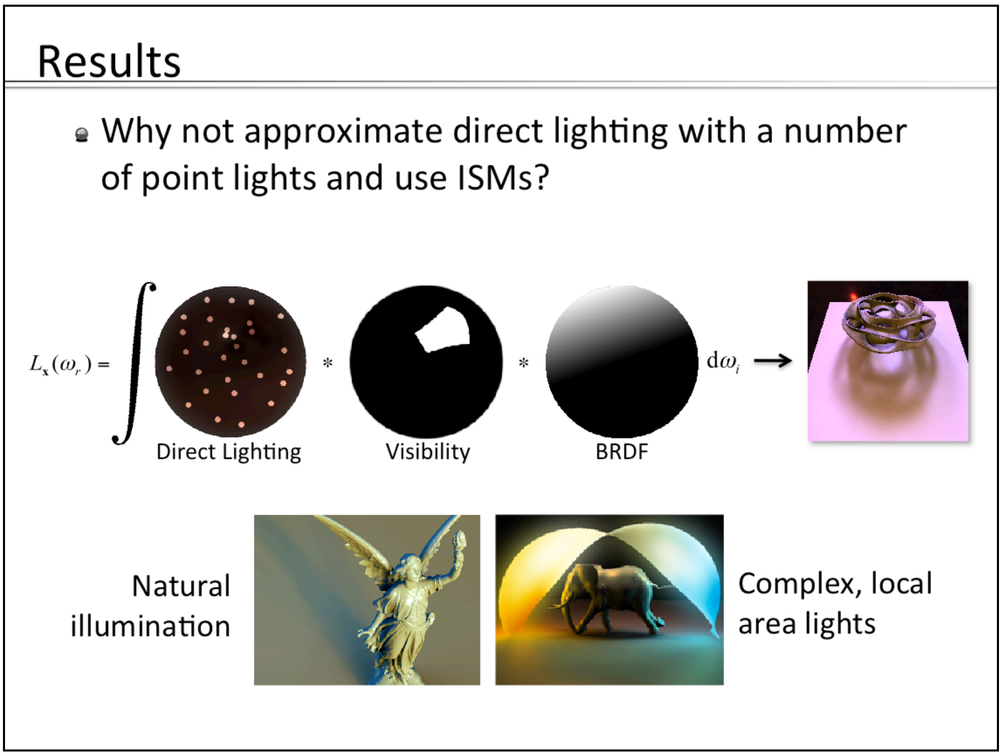
No other method can do this - all **previous real-time** methods were **essentially limited** to **static scenes**.



Finally, we are **not limited to diffuse materials**.
Here, we have a gold ring, casting a **caustic at 15 fps** with full **indirect visibility.**

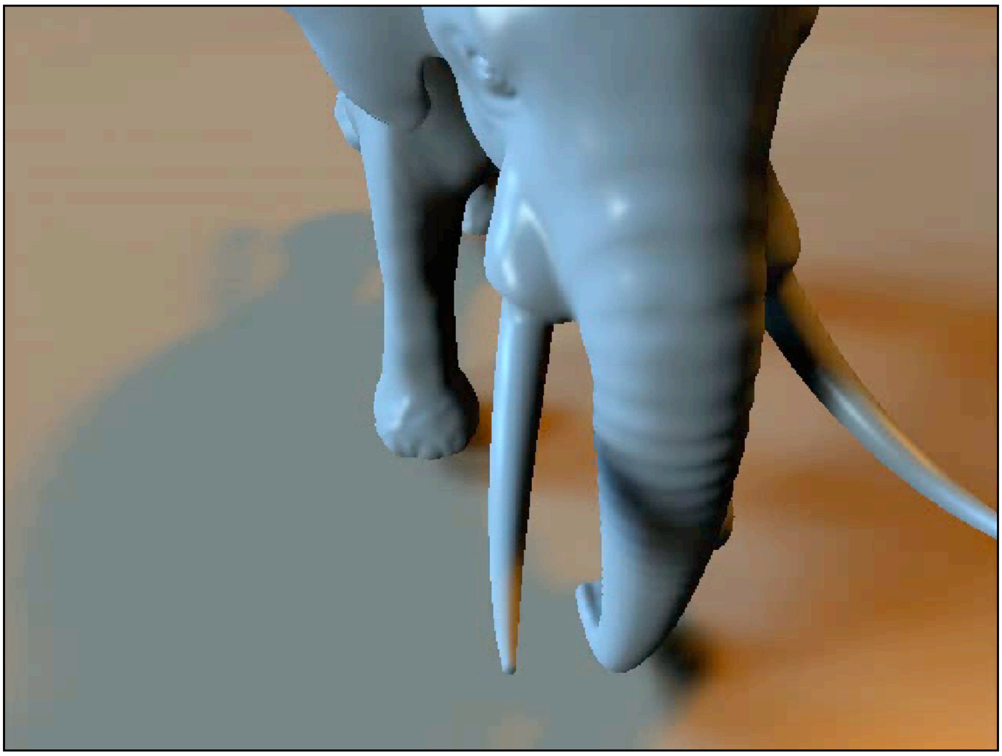


Imperfect reflective shadow map, generalize all this to additional bounces.

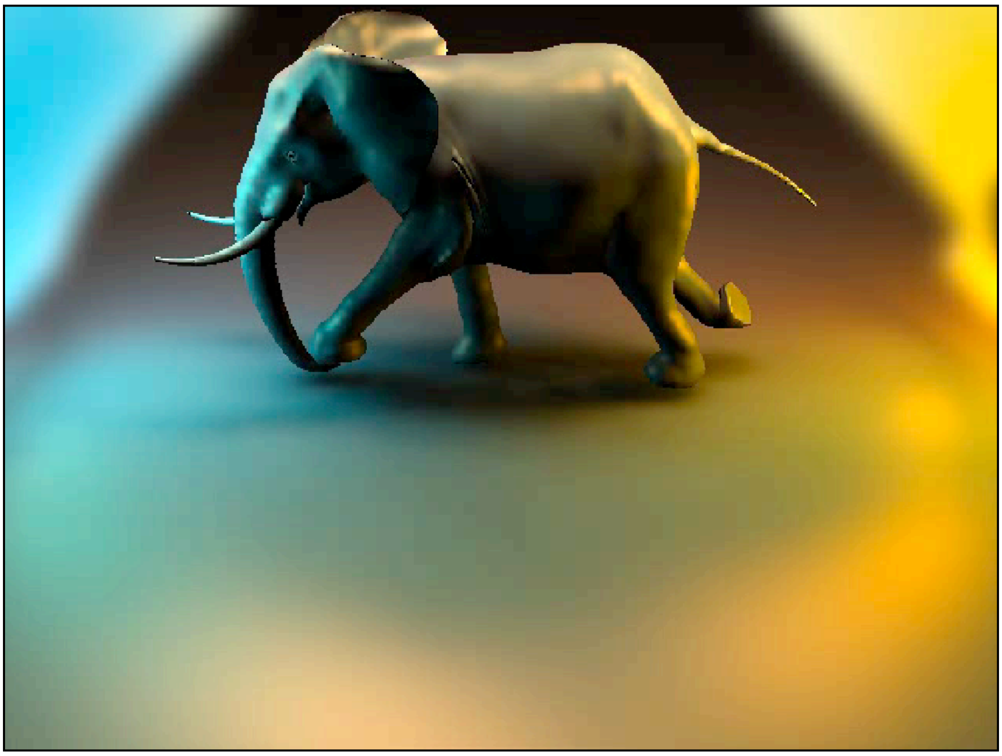


Why couldn’t we use the **same idea** for direct lighting and **approximate** it with a **number of point lights**.

In fact, that is easily possible. Let’s look at two results.



In this example, animals using **direct natural illumination** are rendered using ISMs. Note the shadows and the glossy highlights.



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

Again, here the area light is **approximated** with many **point lights** and use **ISMs for rendering**.

Notice the **glossy** reflections on the **floor**. This is even **difficult for offline** rendering methods.

There is **no other method** that can do this at this speed.

This example runs at **15fps**.

Imperfect Shadow Maps

- ⊕ Exploit simplified indirect visibility
- ⊕ Very fast global illumination
- ⊕ Can be used for direct illumination as well

- ⊖ Diffuse (low-glossy) only
- ⊖ Not (directly) scalable to very large scenes
- ⊖ Parameters need to be set

In summary, **visibility for global illumination effects can be drastically simplified** and **a simple, practical method to exploit this on current graphics hardware** was shown.

Note that in the limit, i.e. with enough points, our method yields correct results.

It can also be used for direct illumination, where it's not quite as fast as the first technique, but more flexible.

There are some limitations, ISMs could be more **scalable** to very large scenes (like an office building).

Also, parameters are currently set by hand, an automated method for setting the parameters is not available.

Schedule

8:30 – 8:40 Introduction (Kautz)

- Motivation
- Problem Statement
- Definitions (Rendering Equation, Neumann Series, …)
- Direct Illumination vs. Indirect Illumination

8:40 – 9:10 Screen Space Techniques (Dachsbacher)

- Screen-Space Ambient Occlusion (SSAO)
- Extending SSAO to Indirect Illumination
- Reflective Shadow Maps
- (Multiresolution) Splatting of Indirect Illumination
- Examples, Results, Limitations

9:10 – 9:40 Virtual Point Lights (Kautz)

- Instant Radiosity
- Incremental Instant Radiosity
- Imperfect Shadow Maps
- Examples, Results, Limitations

9:40 – 10:05 Hierarchical Finite Elements (Dachsbacher)

- Dynamic Ambient Occlusion for Indirect Illumination
- Implicit Visibility
- Anti-Radiance
- Examples, Results, Limitations

10:05 – 10:15 Conclusions/Summary (Kautz)

* Comparison of Presented Techniques
* Q&A