




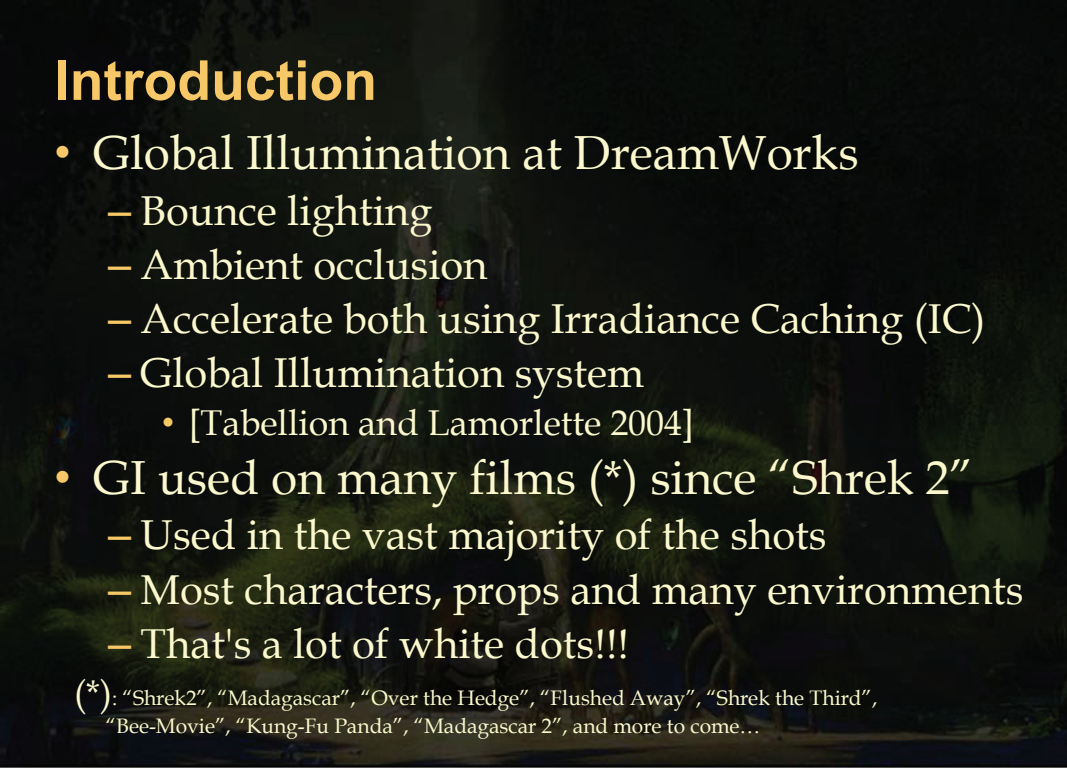
Irradiance Caching at DreamWorks

Eric Tabellion
et@pdi.com



 SIGGRAPH2008



 DREAMWORKS
ANIMATION SKG



Introduction

- Global Illumination at DreamWorks
 - Bounce lighting
 - Ambient occlusion
 - Accelerate both using Irradiance Caching (IC)
 - Global Illumination system
 - [Tabellion and Lamorlette 2004]
- GI used on many films (*) since “Shrek 2”
 - Used in the vast majority of the shots
 - Most characters, props and many environments
 - That's a lot of white dots!!!

(*) : “Shrek2”, “Madagascar”, “Over the Hedge”, “Flushed Away”, “Shrek the Third”, “Bee-Movie”, “Kung-Fu Panda”, “Madagascar 2”, and more to come...

Global Illumination at DreamWorks refers to rendering one bounce of diffuse interreflection, as well as ambient occlusion.

We use a proprietary renderer to render all our films. Many implementation details of this global illumination system are published in our 2004 Siggraph paper [Tabellion and Lamorlette 2004]. In our toolset, both bounce lighting and ambient occlusion are computed using irradiance caching to reduce the amount of raytracing and shading required.

GI has been used extensively at DreamWorks to light and render many animated films since “Shrek 2”. If you consider the list of films above and the number of frames per film, we are glad to have been using irradiance caching to render all these images!

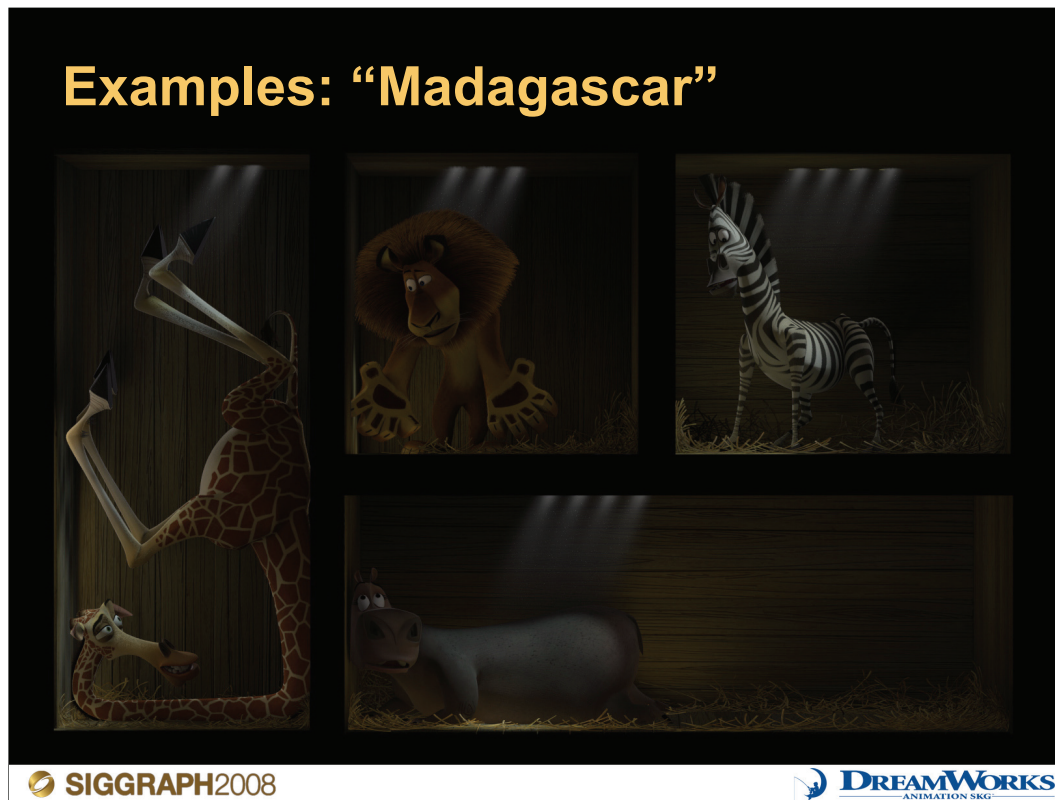
Examples: "Shrek 2"



 SIGGRAPH2008

 DREAMWORKS
ANIMATION SKG

Here are some example images from the movie "Shrek 2"...



... some more examples from movies released since then...

Examples: “Over the Hedge”



 SIGGRAPH2008

 DREAMWORKS
ANIMATION SKG

... some more examples from movies released since then...

Examples: “Bee Movie”



 SIGGRAPH2008

 DREAMWORKS
ANIMATION SKG

... some more examples from movies released since then...

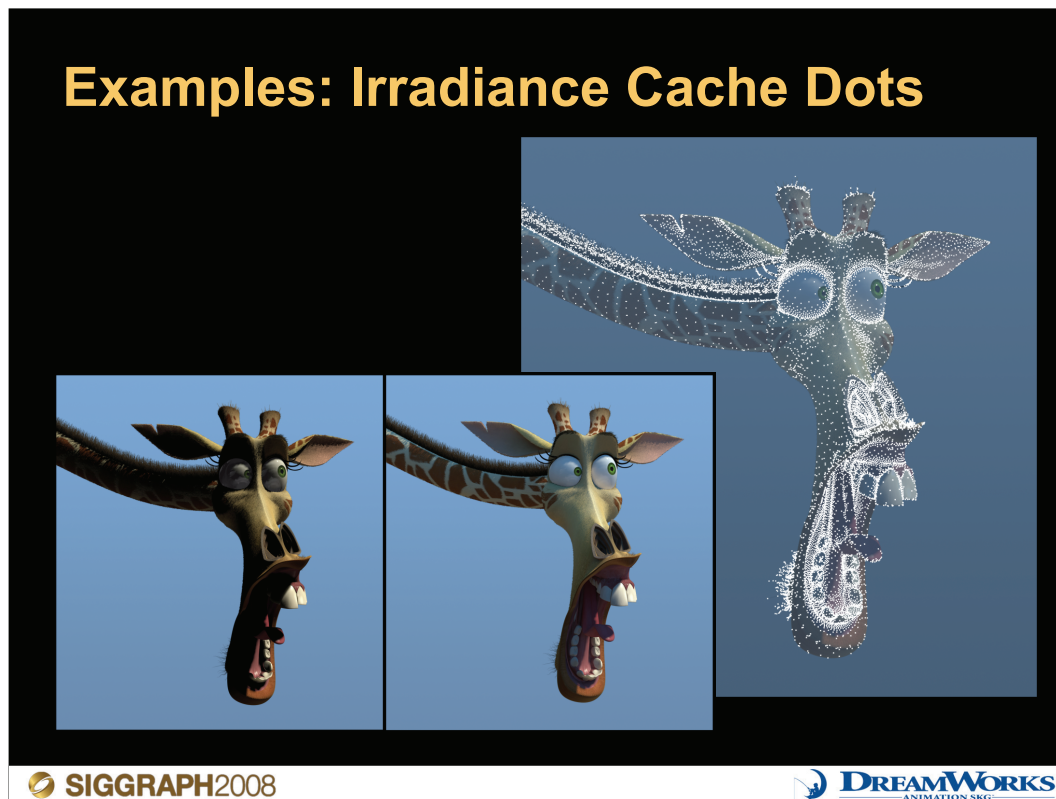
Examples: “Kung Fu Panda”



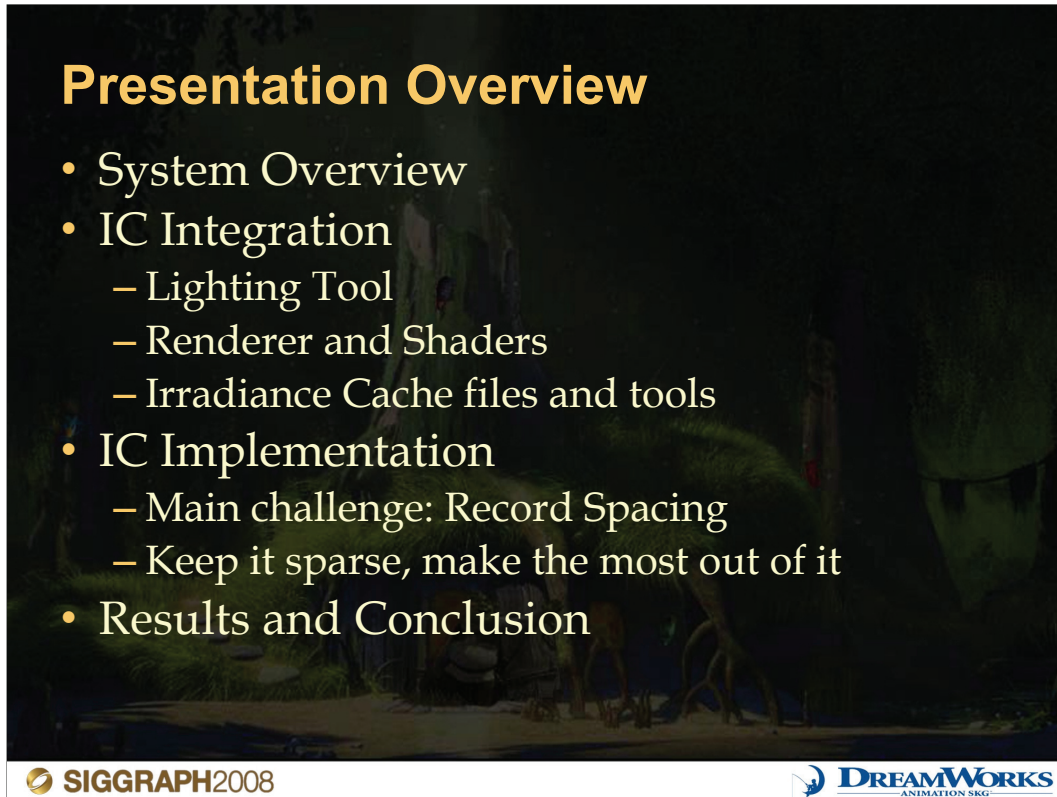
 SIGGRAPH2008

 DREAMWORKS
ANIMATION SKG

... and some more examples from the movie “Kung Fu Panda”. All these images use irradiance caching under the hood.




Here are image that show Melman with and without global illumination, as well as the corresponding irradiance cache.



Presentation Overview

- System Overview
- IC Integration
 - Lighting Tool
 - Renderer and Shaders
 - Irradiance Cache files and tools
- IC Implementation
 - Main challenge: Record Spacing
 - Keep it sparse, make the most out of it
- Results and Conclusion

In this section of the course, we will share some of our practical experience of implementing and using irradiance caching in a film production environment.

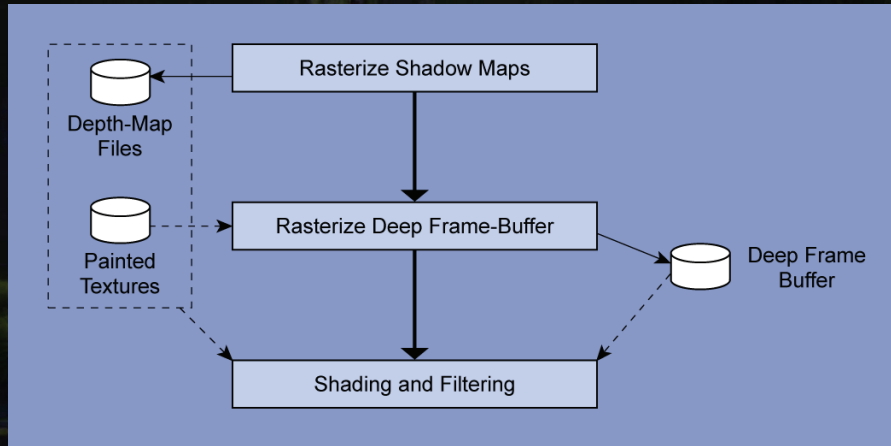
First we will review our global illumination system, and describe how irradiance caching fits within our rendering pipeline and how it integrates within a parallel production renderer. We will also talk and demo some of our tools to handle Irradiance Cache files.

We will then review some of the implementation details, covering the main challenge of Irradiance Caching: record spacing. We will show how specific solutions can help make IC up to an order of magnitude faster compared to final gathering at every pixel.



System Overview

- Micropolygon Scanline Renderer
- Streaming and Caching Architecture
- Deep Frame Buffer + Lighting Tool



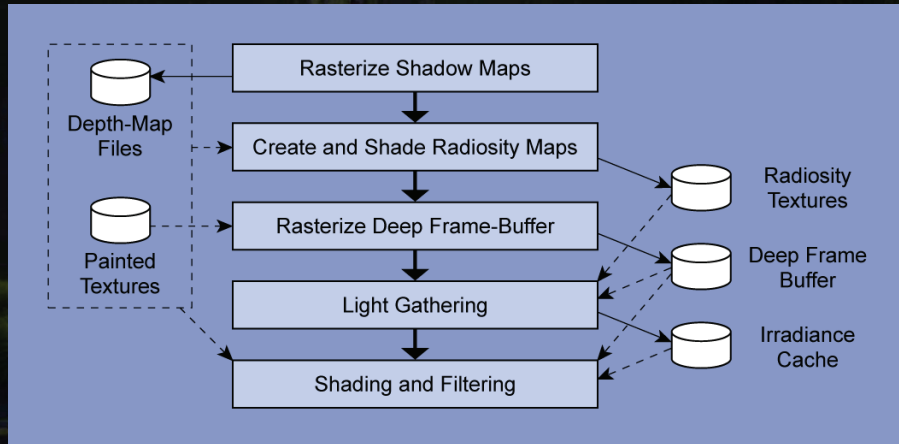
SIGGRAPH2008

DREAMWORKS
ANIMATION SKG

Our system is built around a proprietary micropolygon scanline renderer. It produces a deep frame buffer that can be used in an interactive lighting tool to shade an image multiple times while adjusting shader parameters. The deep frame buffer itself acts as a primary visibility rasterization cache and is also used when rendering images in batch for final quality renders.

Adding Global Illumination with IC

- One Bounce of Diffuse Interreflections
- Ambient Occlusion
- Ray Tracing Approach using Irradiance Caching

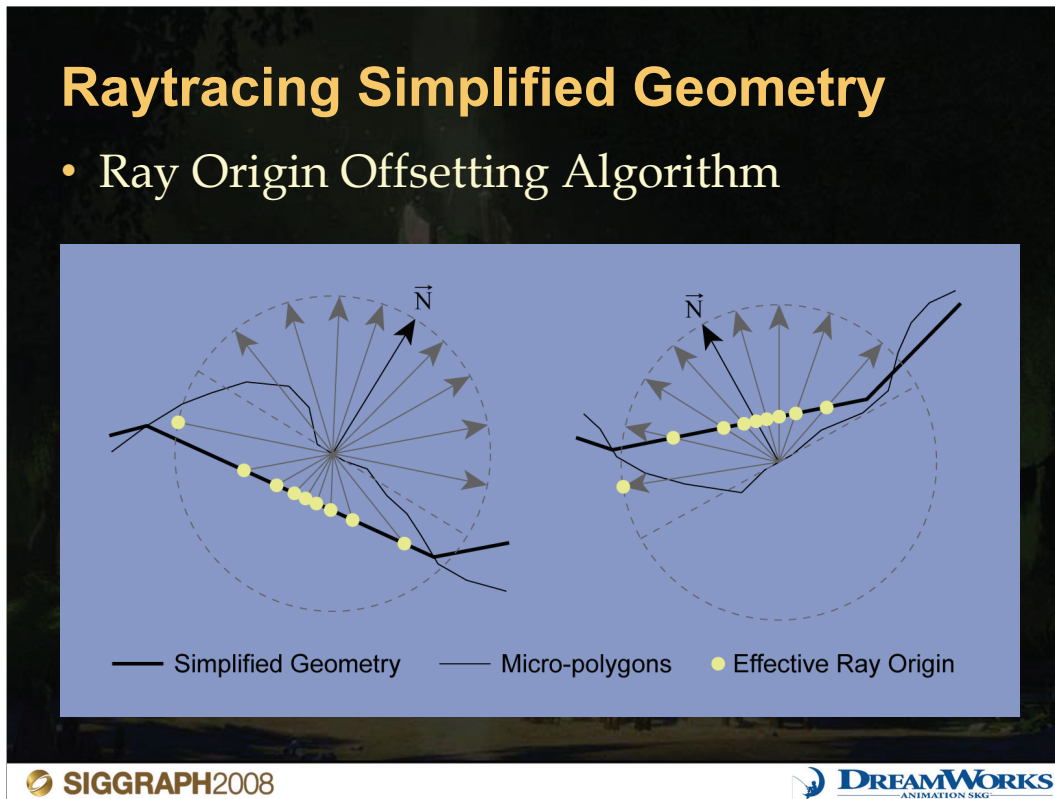


SIGGRAPH2008

DREAMWORKS
ANIMATION SKG

We enhanced our renderer with single-bounce global illumination capability by using a raytracing engine (also proprietary). We use it to perform final gathering calculations, as well as render reflections and refractions. To accelerate the expensive light gathering process, we implemented Irradiance Caching. The irradiance caching approach fits naturally into our rendering pipeline, which is using many other stages of caching. This strategy comes in handy when only a few steps need to be run again in the user's workflow to accomplish a task. Irradiance caching also helps by speeding up ambient occlusion calculations as we will see in later slides.

To further accelerate final gathering, the user has the ability to pre-compute a set of radiosity texture maps. These "baked" textures are called radiosity textures not because they have been computed using the radiosity algorithm, but rather because they contain a radiometric quantity known as radiosity. The textures essentially look like the result of direct lighting evaluations on purely diffuse textured surfaces. They are used to accelerate the shading of each gathering ray-intersection by replacing potentially complex shader network evaluations by a simple texture lookup. They can be thought of as the equivalent of a photon map, simplified to the context of a single bounce of diffuse interreflections.



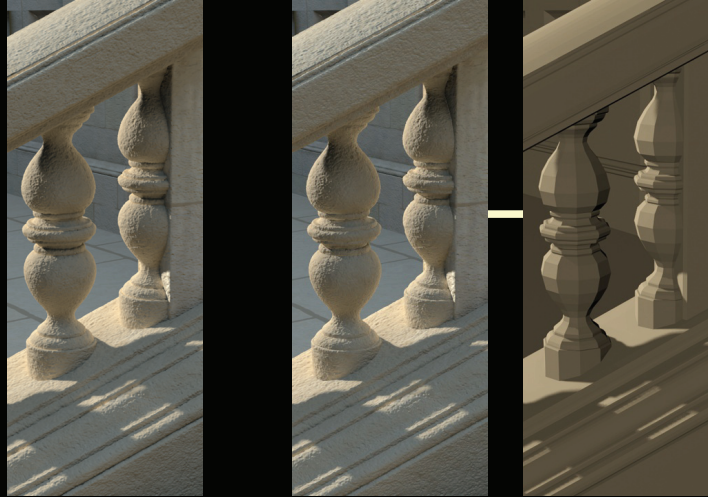
Here we illustrate how we integrate our raytracing engine within our micropolygon scanline renderer. When rays are cast, we do not actually attempt to intersect geometry finely tessellated down to pixel-size micropolygons - this would be indeed very expensive and use a lot of memory. Instead we tessellate geometry down to a simplified resolution and the raytracing engine only needs to deal with a much coarser set of polygons.

Since rays originate from positions on the micropolygon tessellation of the surface and can potentially intersect a coarser tessellation of the same surface, self intersection problems can happen. The image above illustrates cross-section examples of a surface tessellated both into micropolygons and into a coarse set of polygons. It also shows a few rays that originate from a micropolygon with directions distributed over the hemisphere. To prevent self intersection problems to happen, we use a ray origin offsetting algorithm. In this algorithm, we adjust the ray origin to the closest intersection before / after the ray origin along the direction of the ray, within a user-specified distance. The ray intersection returned as a result of the ray intersection query is the first intersection that follows the adjusted ray origin. Please see our paper [TL04] for more details.

Raytracing Simplified Geometry

Raytracing 2 million micropolygons

Raytracing 2 thousand polygons



 SIGGRAPH2008

 DREAMWORKS
ANIMATION SKG

Here is a comparison between our technique (center), and raytracing micropolygons (left). The image in the right shows the coarse tessellation used to perform ray intersections.



Let's go over a couple of examples to see our system in action.

Here is a simple example to illustrate direct and indirect illumination. In this image, direct illumination is coming from the left side of the image as indicated by the white arrow. Any surface area not exposed to the light directly is completely dark.



Once we let the direct light bounce on the geometry in the scene, we get this image. We call indirect illumination the secondary lighting that naturally fills up the dark areas and produces very soft-looking images. Notice also the color bleeding effects alongside the walls. You can compare by flipping back and forth between both slides.

One very nice feature bounce lighting provides for free, is the ability to control the lighting on a subject using distant off-screen bounce cards or reflectors. This offers a similar control to what a director of photography would expect on a live action set.

Key Lighting Only



This example from the movie "Shrek2" illustrates final gathering using irradiance caching. As in the previous example, this image contains direct lighting only.



Without using Irradiance Caching, gathering calculations happen for each pixel in the image where we want to compute indirect illumination. At each pixel we send several hundred rays from the surface seen through that pixel, with a cosine-weighted directional distribution over the hemisphere above it. The intersection of each ray is then shaded and the result of these evaluations are averaged together, producing an irradiance value.

This image illustrates a few of the rays cast to compute the irradiance for a single pixel on Shrek's arm. The image was intentionally rendered using the pre-computed radiosity textures that are used to shade each ray. Notice the overall blurry low-quality of the textures. These textures don't need to be pre-computed at very high resolution. In the background, there is also a textured sky-dome that rays can also intersect, as well as an off-screen ground plane.

Indirect Lighting Only



Here is the same image shaded only with indirect lighting, using the irradiance computed as described in the previous slide.

Key Lighting + Global Illumination



 SIGGRAPH2008

 DREAMWORKS
ANIMATION SKG

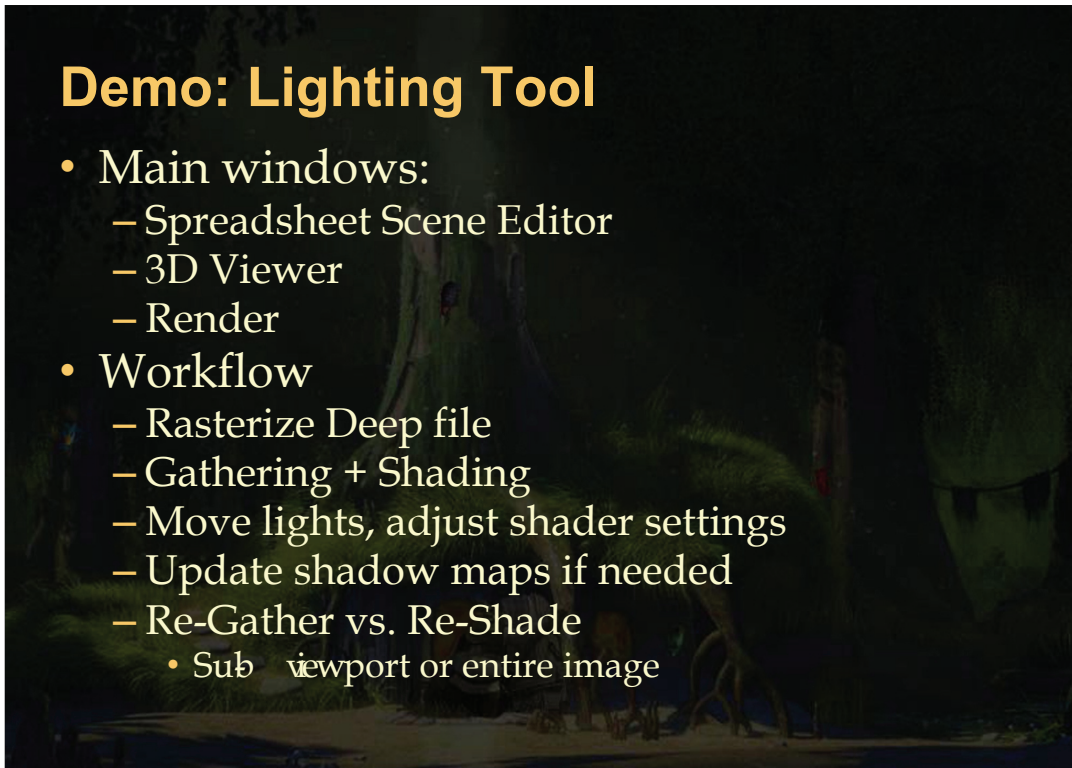
And again, adding the direct lighting back in the equation.



When we use irradiance caching, the expensive irradiance calculations are only performed at the “white dots” in this image and are cached for reuse. In between the white dots, we use the irradiance and its gradients from several neighbor cache records to smoothly interpolate the irradiance.



Notice the low density of the cache record distribution on surfaces that are smooth (belly, arms, hands, etc.). Also notice how dense the cache distribution can get in creases and on surfaces that are displacement-mapped in a way that produces high-frequency geometric detail (vest). We will talk in later slides how to remedy this problem, as well as what solution we use to deal with fur (eyebrows, medium-to-short fur and grass in general).





Demo: Lighting Tool

- Main windows:
 - Spreadsheet Scene Editor
 - 3D Viewer
 - Render
- Workflow
 - Rasterize Deep file
 - Gathering + Shading
 - Move lights, adjust shader settings
 - Update shadow maps if needed
 - Re-Gather vs. Re-Shade
 - Sub viewport or entire image

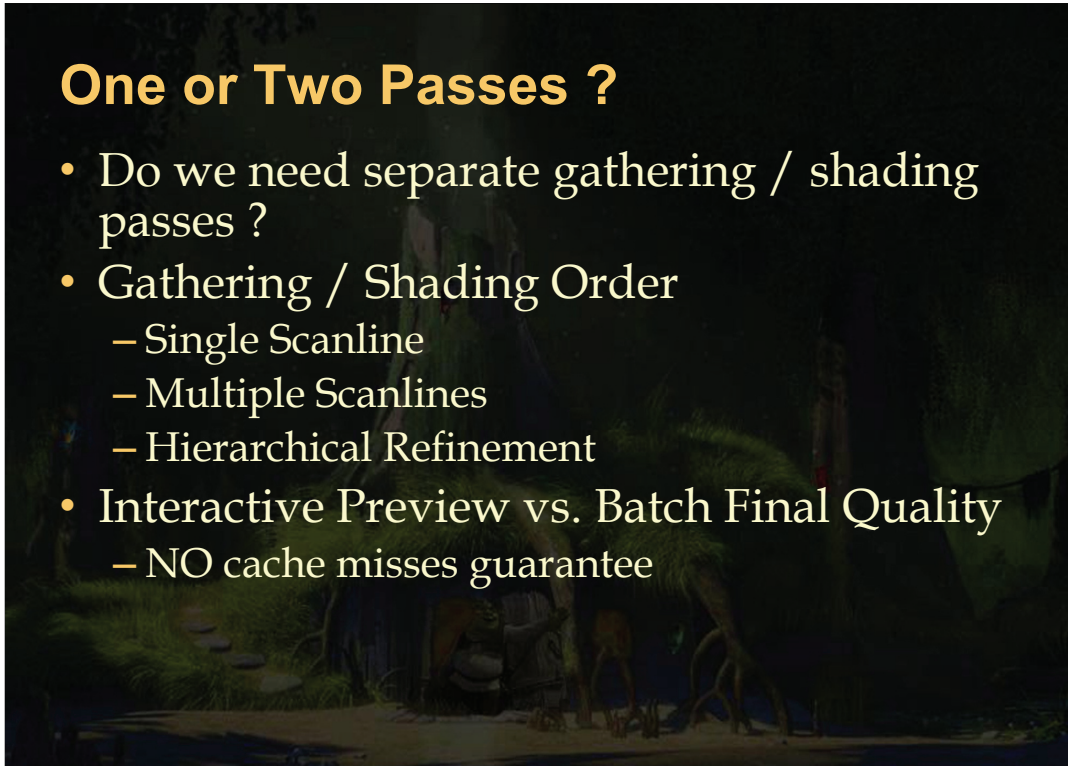
 

During the course we will show a video of our interactive lighting tool being used through a typical user workflow session.

The lighting tool allows the user to edit scene and shader parameters through a spreadsheet interface. A 3D viewer window lets the user handle light and camera placement as well as inspect geometry. A render window displays the rendered images and allows the user to do rendering tasks, such as rasterizing a deep file, updating shadow maps, and starting and stopping (re)gathering and/or (re)shading.



Irradiance caching works quite well within this workflow, specifically when the user wants to gather and re-shade many times specific sub-viewports and then finally update the entire image.

Explicitly requesting gathering triggers the irradiance cache to be flushed before the gathering/shading passes start. If the user only wants to re-shade, the pre-existing irradiance cache (if any) is used. Gathering may still happen if the user is shading a viewport that does not yet contain any/enough irradiance cache records.



One or Two Passes ?

- Do we need separate gathering / shading passes ?
- Gathering / Shading Order
 - Single Scanline
 - Multiple Scanlines
 - Hierarchical Refinement
- Interactive Preview vs. Batch Final Quality
 - NO cache misses guarantee

One question that arises with using Irradiance Caching, is whether a separate shading pass is needed after the gathering pass, or if both gathering and final shading can happen in the same pass. The answer mainly depends if the user is doing interactive preview work or if final quality rendering is required.

Another important factor is the gathering/shading order. When the gathering/shading calculations happen along a scanline (or multiple scanlines when rendering on a multi-core CPU), irradiance cache records are created as the scanline progresses. In this scheme, notice that cache records only affect interpolated irradiance for subsequent scanlines (and not scanlines that preceded their creation). This usually causes noticeable irradiance discontinuity artifacts, and impacts the artist's workflow. When rendering with scanlines, we found that a separate final shading pass is needed, even for preview work.

When rendering using hierarchical refinement however, the image is scanned multiple times across several sub-passes. The first sub-pass shades only every few pixels and every few scanlines yet displays the whole image with reduced detail in a pixelated fashion. Each subsequent sub-pass shades more and more pixels in between the subset of pixels shaded during previous sub-passes, thereby reducing pixelation and revealing more and more image detail. In this scheme, the same discontinuity problem happens within each sub-pass, but the visual artifacts are greatly reduced by the fact that we use many sub-passes. This makes it acceptable to use a single gathering/shading pass (made of many sub-passes) for preview work.

For final quality rendering, we have an important requirement: no cache misses and no cache record creation can happen during final shading. This would otherwise introduce irradiance discontinuities in the image plane as described above, which then translates into temporal flickering and popping in animations. We therefore use a separate gathering pass before the final shading pass.

Parallel Rendering

- Load Balancing
 - Image tile partition (a.k.a. Bucketing)
 - Threads render many tiles each
- Single Cache – Shared Memory
 - Read many times, write scarcely
- Multiple Caches – Thread Local Storage
 - One cache per thread
 - Merge caches when load-balancing changes
 - Becomes once cache per thread + one global cache
 - Loss of cache coherence across tiles

 SIGGRAPH2008

 DREAMWORKS
ANIMATION SKG

Our renderer can render images in parallel, thereby leveraging the power of multi-core CPUs. We use multiple threads of execution and a divide and conquer strategy. We actually use multi-processing and selective explicit shared memory (for several practical reasons) in a way that is very much similar to multi-threading. For the clarity of the explanation we will refer to multi-threading since most people are familiar with this parallel programming model.

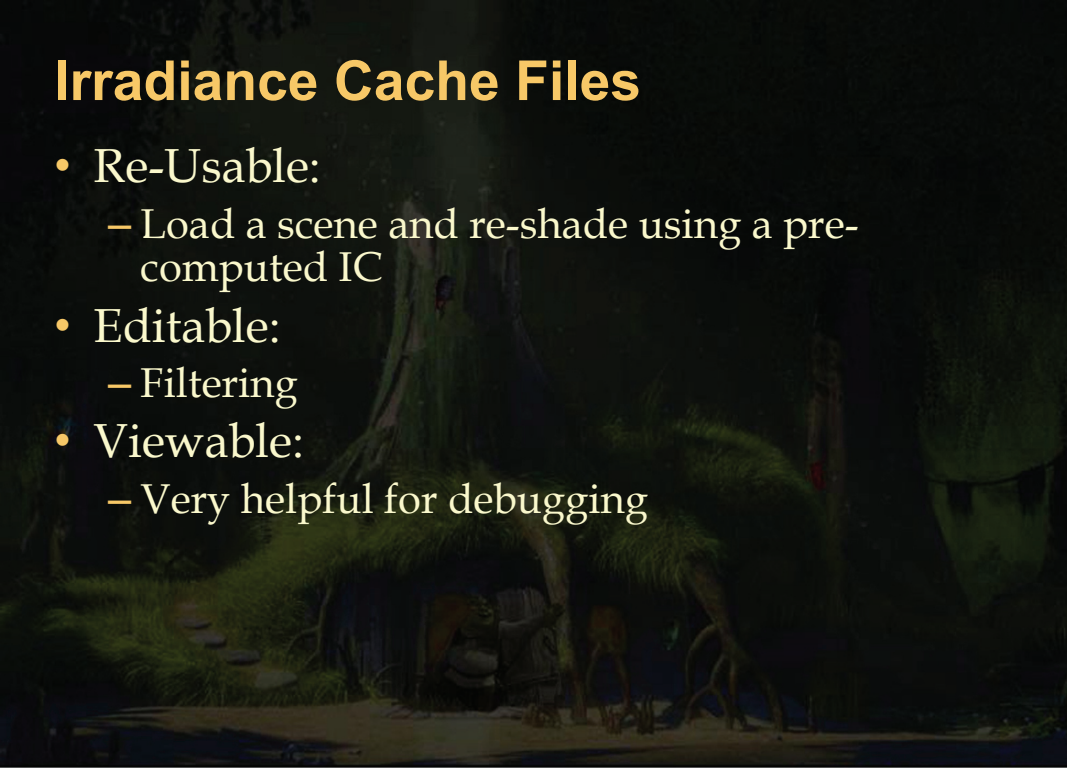
As is common practice, we partition the image into several shading tiles (or buckets), and each thread is responsible for rendering as many buckets as it can in a greedy fashion. The same parallel rendering main-loop is used when using irradiance caching. Notice that the irradiance cache becomes a read-write shared resource across threads, since irradiance cache records can contribute to shading multiple adjacent tiles handled by different threads. This resource sharing can be dealt with in several ways, using classic parallelization techniques.

The simplest approach is to use a single irradiance cache in shared memory and rely on a read-write lock resource sharing mechanism, since the cache will be read from many times and written to scarcely. This works well but places a small contention point and may impact scalability with growing number of CPU-cores and threads.

Another alternative is to use multiple irradiance caches. This may be desirable if we want to remove the contention point altogether, or necessary in cases where the parallel execution environment doesn't provide an efficient shared memory system. In this case each thread can manage its own separate irradiance cache, which ends up being populated with cache records that cover the tiles shaded by the thread.



When considering a multiple cache approach in the context of an interactive lighting tool, some difficulties arise. In such a tool, we might be asking our threads to shade different sub-viewports, and therefore the load-balancing might assign each thread with a different set of image tiles across successive reshades. When the load balancing changes, it becomes necessary to merge all of the threads' individual caches together into a global irradiance cache, which then becomes available to the threads for subsequent gathering/shading. Care must also be taken when merging thread local caches again and again, to avoid redundant duplication of cache records originating from a previous merge (as opposed to newly created cache records).

This approach doesn't come without its own inefficiencies. In a multiple cache scheme, cache records are only re-used within areas of the image shaded by a given thread. In essence, we do lose a bit of cache coherence across tile boundaries, and some extra work will be spent creating cache records on each side of tile boundaries. It is then crucial to choose a big-enough tile size in order to minimize this loss of cache coherence. Unfortunately this is contrary to using small enough tiles for the sake of good load balancing, and therefore also limits the scalability of the algorithm with growing number of CPU-cores.



Irradiance Cache Files

- Re-Usable:
 - Load a scene and re-shade using a pre-computed IC
- Editable:
 - Filtering
- Viewable:
 - Very helpful for debugging

During the course we will show a video of our interactive tools used to edit, and inspect irradiance cache files using a 3D viewer.

Being able to read and write irradiance cache files has been very valuable for various reasons. The most obvious one is the ability to re-use an irradiance cache created in a previous interactive lighting session. The lighters often open a shot and shade an image multiple times using a previously computed irradiance cache.

Irradiance cache files also make the irradiance cache editable, viewable, and even playable over time, which has been very useful for debugging purposes.

Integration with Shaders

- Surface shaders want Radiance
- IC provides Irradiance
- Other problems:
 - Decouple record spacing from Bump
 - Make it work with Non-diffuse BRDF's
- Solution:
 - Assumptions on light direction
 - $\vec{\omega}' =$ Dominant incoming light direction
 - Radiance($\vec{\omega}'$) = Irradiance:

$$L_i(\vec{p}, \vec{\omega}') = \frac{E(\vec{p}, \vec{n})}{|\vec{n} \cdot \vec{\omega}'|}$$

 SIGGRAPH2008

 DREAMWORKS
ANIMATION SKG

Integrating irradiance caching in a rendering system that uses programmable shaders needs to be considered. Most shading systems support surface shaders and light shaders. The latter are most often responsible for computing illumination, and the latter for applying a BRDF. Typically, light shaders provide surface shaders with sets of (radiance, direction) value pairs. Irradiance caching however provides irradiance quantities which are not associated to any specific incoming direction.

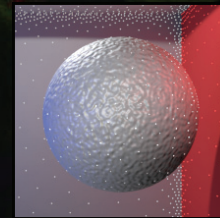
Furthermore we want irradiance caching record spacing to be independent of the bump-mapped normal variation, yet we want bump-mapped surfaces that are lit with irradiance caching to still look like they have some bump. Finally, even though irradiance caching assumes purely diffuse BRDF's in many ways, we also want glossy surfaces lit with irradiance caching to still look reasonably glossy.

To remedy / alleviate the three problems above, we use a simple assumption which is far from fool-proof but works well in practice. During the gathering process, we keep track of the dominant incoming light direction (also called “bent normal” in the literature) by simply computing an average of ray directions over the hemisphere, weighted by their contribution to the corresponding irradiance value. We also store this direction in the irradiance cache, and interpolate it similarly to how we interpolate the irradiance. Our assumption is then to assume that all of the lighting contributing to the cached irradiance comes from this direction. With this assumption it is then trivial to compute the corresponding radiance and feed this information to surface shaders, which then use the bumped normal in their BRDF evaluations.

Approximate Shading Model

$$L_o(\vec{p}, \vec{\omega}) = L_e(\vec{p}, \vec{\omega}) + \sum_k f_r(\vec{p}, \vec{\omega}'_k, \vec{\omega}) L_d(\vec{p}, \vec{\omega}'_k) |\vec{n} \cdot \vec{\omega}'_k| \\ + \sum_{\lambda=R,G,B} f_r(\vec{p}, \vec{\omega}'_\lambda, \vec{\omega}) L_{i\lambda}(\vec{p}, \vec{\omega}'_\lambda) |\vec{n} \cdot \vec{\omega}'_\lambda|$$

- Result:
 - Sparse record spacing
 - Interaction with complex BRDFs
 - Works well with Bump Mapping



 SIGGRAPH2008

 DREAMWORKS
ANIMATION SKG

Formally speaking, the formula above defines our approximate shading model based on the description of the previous slide. In this formula, we also decompose the incoming light direction calculations over three “wavelength” (the three RGB channels in fact), which provides three (radiance, direction) value pairs for each pixel (one for each wavelength). This color separation trick tends to increase chromaticity effects on bumped glossy surfaces, but may also cause color separation which is undesirable. It is not always used and can be dialed in/out in our system.

The result of this model is shown on a bumped glossy sphere within a cornell box. Notice the sparcity of the record spacing and the glossiness of the sphere appearance. The sphere here is only lit with indirect illumination using irradiance caching and the shading model we described.





Main Challenge: Good Record Spacing

- Record spacing flickers in animation
- Interpolated irradiance does not
 - Requires stable gradients
 - Rays per Record > Rays per Pixel
- IC still a win if:
 - Total rays with IC < Total rays without IC
- Challenge: keep spacing sparse
 - Even with high frequency detail: bump, displacement, fur, foliage, etc.
 - Use knowledge from the scene

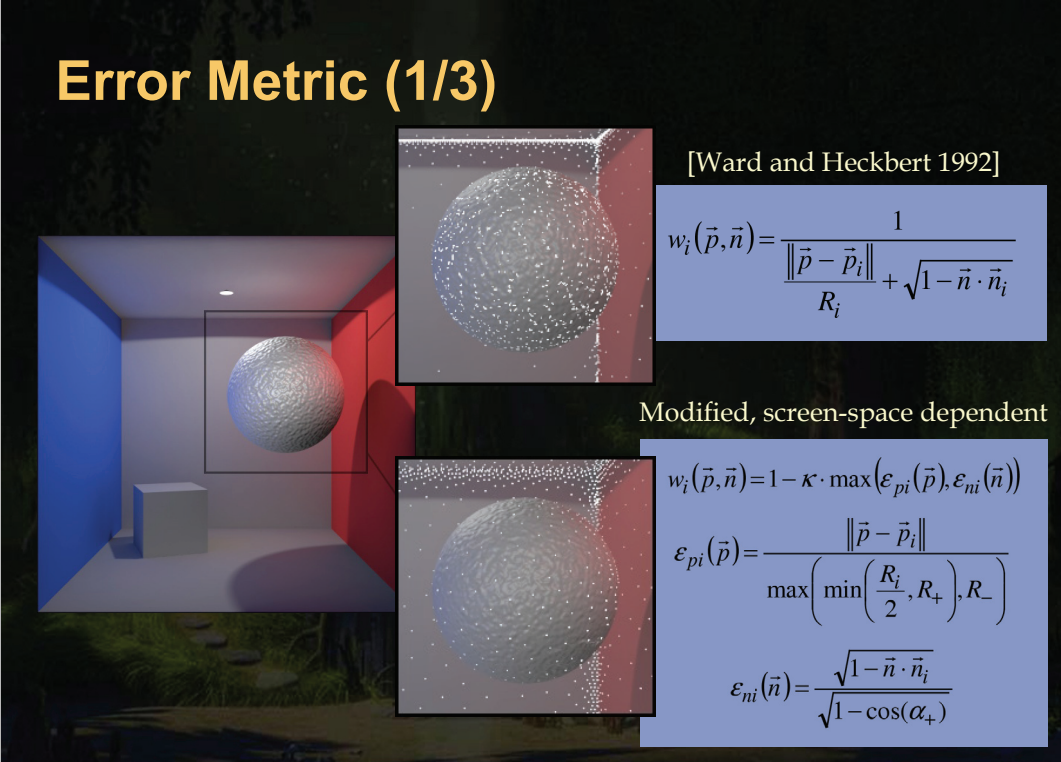
In this section we will talk about the main challenge in implementing irradiance caching: providing a good record spacing quality.

As we've seen, it is possible to render stable non-flickering animations using irradiance caching, even though the underlying cache record distribution changes as can be seen in playbacks of images rendered with irradiance cache dots. At first, it is in fact hard to imagine that a stable result is possible with such a flickering distribution.

Stable results require accurate irradiance values and stable gradients, and in turn may require more rays per cache record, than might be needed per pixel if we were not to use irradiance caching. Even though, using irradiance caching is still a win as long as the record spacing remains sparse enough to offset this cost.

In the next few slides we will talk about the few extra tricks we use to control the record spacing distribution, and show how we can use scene-knowledge to exploit underlying geometry smoothness even in cases with high frequency displacement maps or fur.

Error Metric (1/3)



[Ward and Heckbert 1992]



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

Modified, screen-space dependent

$$w_i(\vec{p}, \vec{n}) = 1 - \kappa \cdot \max(\epsilon_{pi}(\vec{p}), \epsilon_{ni}(\vec{n}))$$

$$\epsilon_{pi}(\vec{p}) = \frac{\|\vec{p} - \vec{p}_i\|}{\max\left(\min\left(\frac{R_i}{2}, R_+\right), R_-\right)}$$

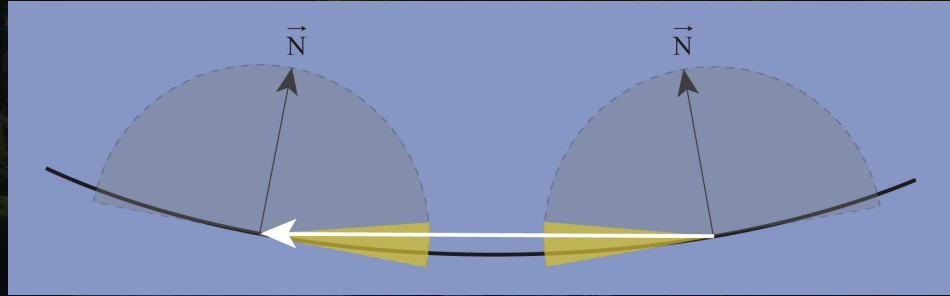
$$\epsilon_{ni}(\vec{n}) = \frac{\sqrt{1 - \vec{n} \cdot \vec{n}_i}}{\sqrt{1 - \cos(\alpha_+)}}$$

As described in our paper [TL04], we use a different error metric than the one proposed by [Ward and Heckbert 1992]. It allows us to control the record spacing with a screen-space metric which allows us to bound the minimum and maximum screen-space distance between cache records. As emphasized by previous speakers in this course, this has many advantages especially in scenes with wide depth range or when considering the world-space scale invariance of the metric. We also use a slightly more intuitive normal error metric, normalized once the normal deviation reaches a specific maximum angle.

Error Metric (2/3)

- Also use “Bump” trick
- Also use “Step” error metric
- Concave surfaces
 - Select ray hits for minimum distance



SIGGRAPH2008

DREAMWORKS
ANIMATION SRG

Like other implementations, we also use the bump mapping trick and the “step” error metric described previously in this course.

Additionally we carefully deal with concave surfaces to prevent unnecessarily dense record spacing. We add an extra test to select rays when computing the minimum distance R_i . We simply use a threshold on the angle between the ray direction and the surface normal both at the gathering point and at the ray intersection point. In the case we fall above the threshold for both angles, we do not discard the corresponding radiance value, but we discard the ray’s intersection distance from contributing to R_i . This helps achieve a coarse record spacing distribution on “slightly” concave surfaces. How slightly depends on the value chosen for the angle threshold.

Error Metric (3/3)

- Use Irradiance Gradient magnitude:
 - Gradient can explode numerically
 - Avoid artifact amplification in corner areas
 - Use Weber's Law:

$$k = \frac{(\hat{n}_i \times \hat{n}) \cdot \vec{\nabla}_r E_i + (\vec{P} - \vec{P}_i) \cdot \vec{\nabla}_t E_i}{E_i}$$

Another problem that happens frequently with irradiance caching, is the numerical explosion of the translation gradient vector's magnitude. This is the case when ray intersection distances become really small as in corner areas, as the formulation of the gradient requires dividing by the ray intersection distance.

We tried arbitrarily capping the magnitude of the gradient vectors, but this results in under or over estimation of the irradiance in corners. What we found works quite well is to limit the interpolation from cache records based on the magnitude of the gradient. We add an extra term to our error metric formula, and limit how far a cache record can be reused based on Weber's law. We simply threshold on the ratio between the irradiance delta coming from applying the gradient at a specific position and the sampled irradiance stored in the cache record.

Combining Metrics

- Combine error metrics: position, normal, step, gradient magnitude, etc.

$$w_i(\vec{p}, \vec{n}) = 1 - \kappa \cdot \max(\epsilon_{pi}(\vec{p}), \epsilon_{ni}(\vec{n}), \dots)$$

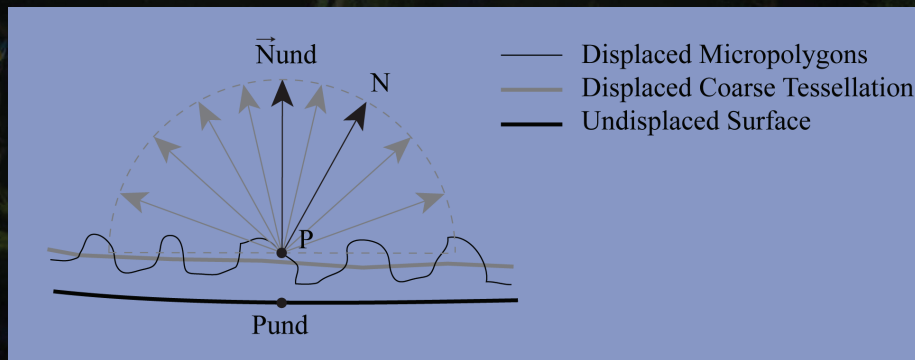
- Use MAX of the errors, instead of SUM
 - More relaxed
 - Easier to understand
- Cache misses happen:
 - MAX: At least one of the metrics must fail
 - SUM: Small errors in various metrics

To make the model a bit easier to understand, we use a combination of error metrics that are all normalized between 0 and 1. When gathering / shading a particular pixel, we consider a set of candidate cache records to interpolate from. When one of the error metrics reaches 1, the corresponding cache record is discarded. If all candidate cache records are discarded a cache miss happens.

Combining our error metrics that way provides a more relaxed approach, where at least one of the error metrics must fully fail in order to discard a cache record.

Displacement (1/4)

- Displacement Mapping generates detail
- Exploit the underlying smooth surface:
 - Use P and N_{und} for gathering
 - Use P_{und} , N_{und} for record spacing



 SIGGRAPH2008

 DREAMWORKS
ANIMATION SKG

We now describe how we can maintain a sparse record spacing distribution on surfaces that are displacement mapped.

Displacement mapping can produce surfaces with high frequency geometric detail and rapidly varying surface normal and even cause abrupt surface normal discontinuities. One possible solution is to exploit the smoothness of the underlying un-displaced surface:

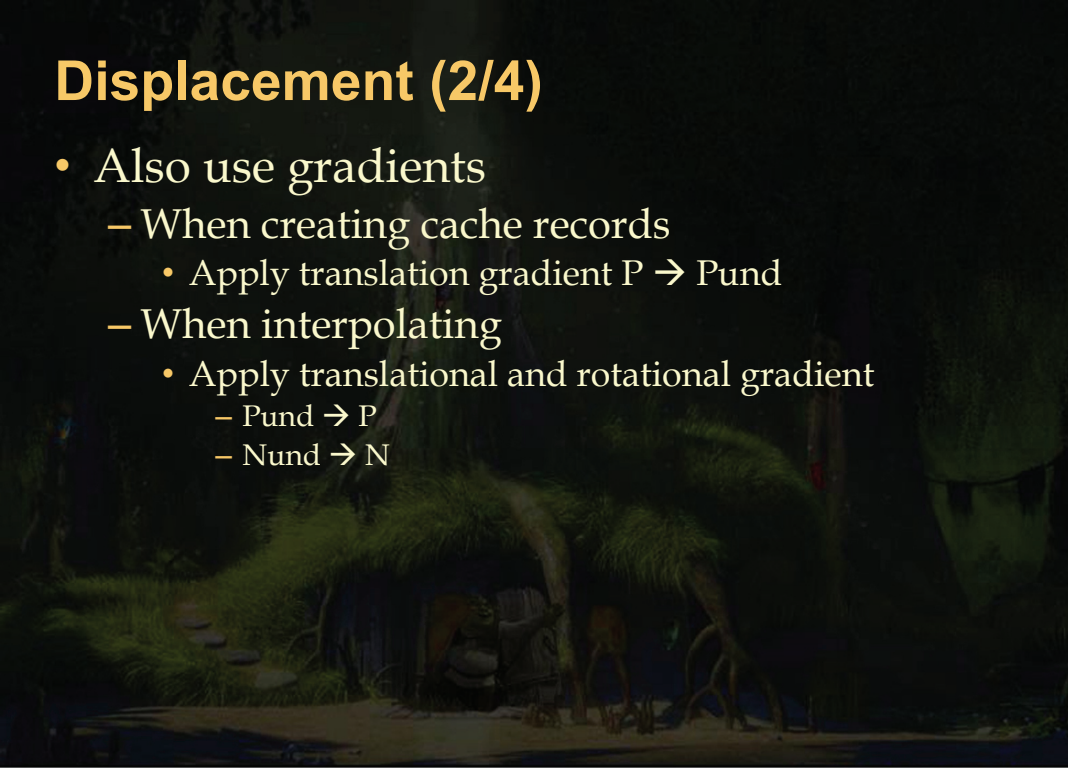
- Cache records are stored on the un-displaced surface,
- Cache lookups happen on the un-displaced surface,
- The irradiance interpolation should be careful to apply gradient vectors to extrapolate the irradiance to the displaced shading position and normal.



When sampling the irradiance however, we want to use the displaced position as ray origins to avoid obvious self-intersection problems. We also want the hemisphere of ray directions to be aligned on the un-displaced surface normal. This is so our irradiance gradient calculations align with the un-displaced smooth surface.

For this to be possible, we simply store in our deep file the un-displaced surface position and normal for each displaced micropolygon vertex.

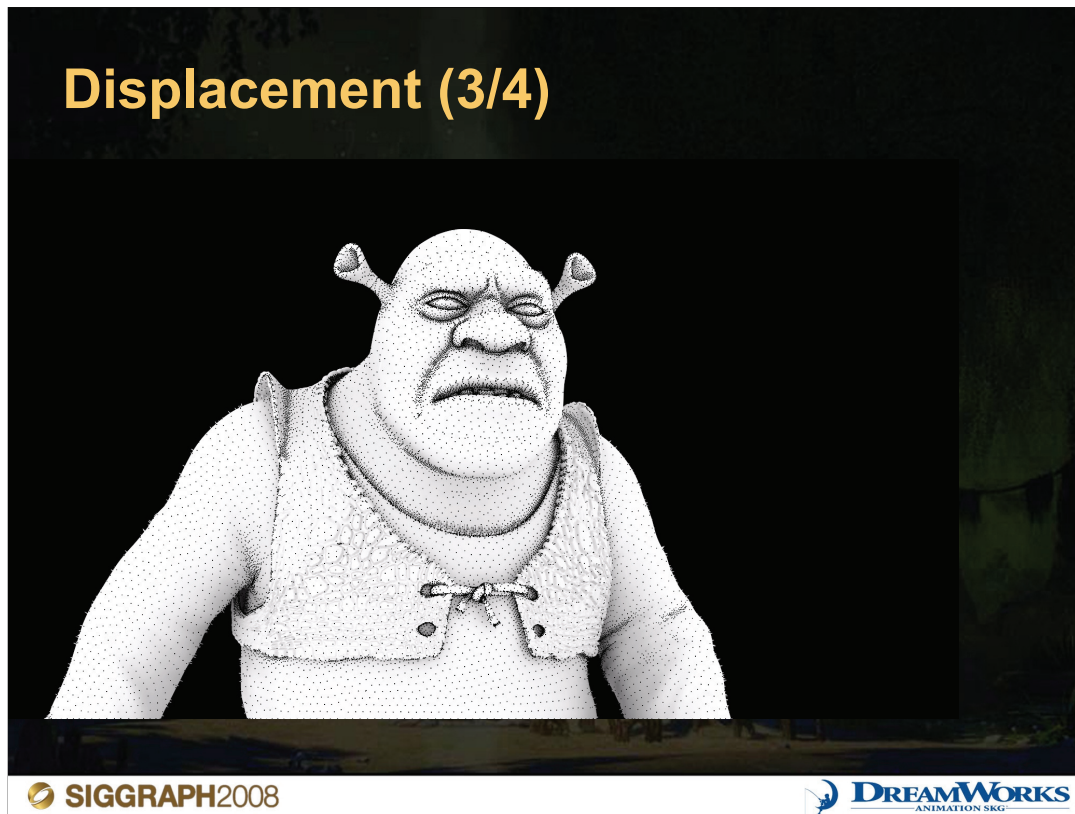
Displacement (2/4)

- Also use gradients
 - When creating cache records
 - Apply translation gradient $P \rightarrow P_{und}$
 - When interpolating
 - Apply translational and rotational gradient
 - $P_{und} \rightarrow P$
 - $N_{und} \rightarrow N$



In the case where displacement mapping can move surface vertices along any direction (not just along the surface normal), it may be beneficial for each new cache record to estimate the irradiance value at the cache record location on the un-displaced surface. For this purpose we can use the sampled irradiance value on the displaced surface and the translation irradiance gradients.



And here is what the record spacing looks like.



Here is the image you get...

Displacement (4/4)

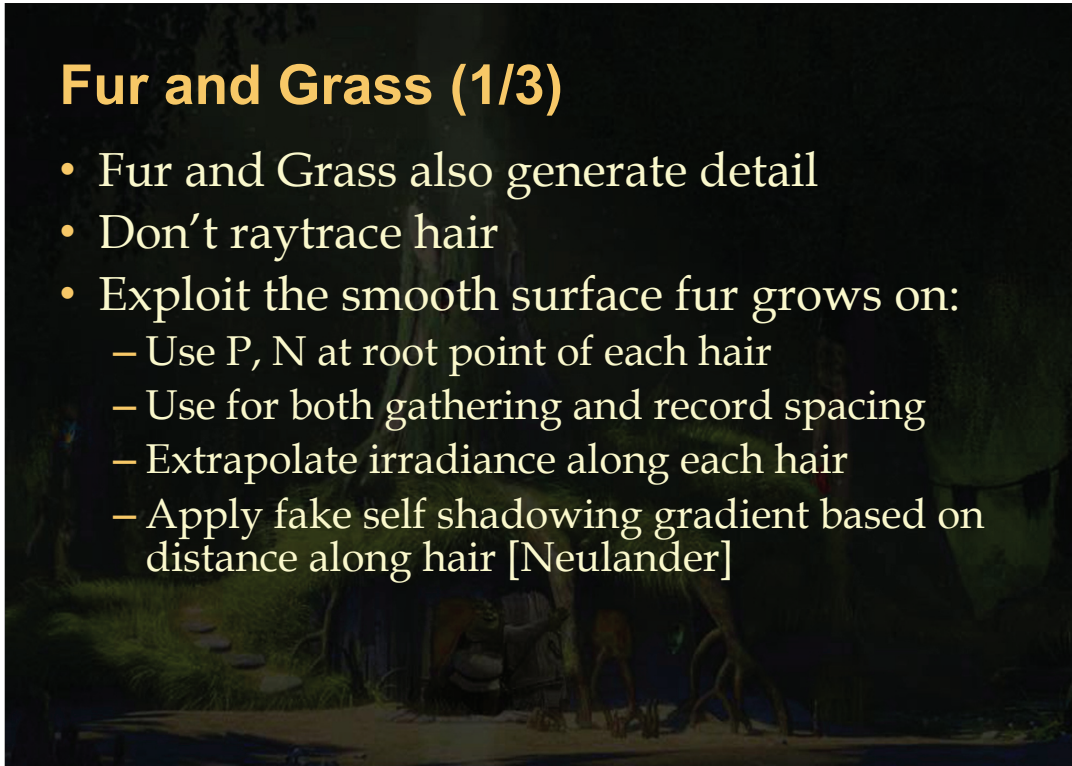
- Show video
- Results
 - Captures occlusion/bounce from surrounding objects onto small displaced detail
 - Misses self-occlusion from small displacements
 - Also from raytracing approximate geometry
 - Can be statically baked accurately





During the course we will show an animation with some results. As we can see in the images and video, we can achieve a sparse record spacing on displaced surfaces, and still capture occlusion / bounce from surrounding objects onto small displaced detail. When shrek's arm occludes its vest, it reveals small displaced surface detail.

We do however miss self-occlusion or self-bounce from small displacements with this technique. This is also partially the fact that we raytrace approximate geometry. In the case of ambient occlusion, this effect is mostly static and can be pre-computed accurately, stored in uv texture maps and easily combined to the final result.



Fur and Grass (1/3)

- Fur and Grass also generate detail
- Don't raytrace hair
- Exploit the smooth surface fur grows on:
 - Use P, N at root point of each hair
 - Use for both gathering and record spacing
 - Extrapolate irradiance along each hair
 - Apply fake self shadowing gradient based on distance along hair [Neulander]

Characters and objects covered in fur and grassy ground planes are more and more common and pose challenging rendering problems. Raytracing hair is one of them, and in our system we typically avoid explicitly doing it: hair rarely directly contributes to occlusion or bouncing light onto itself or onto other surrounding surfaces and is sometimes replaced by lower resolution proxy geometry to achieve the desired effect.

However, we definitely want hair to receive bounce lighting or occlusion from neighboring surfaces. We therefore need to perform gathering calculations from hair fragments and we can accelerate this process using irradiance caching for medium-to-short hair and grass. Here again we can exploit the smoothness of the underlying surface that fur grows on (called the “growing surface” below):

- Cache records are stored on the growing surface,
- Cache lookups happen on the growing surface for each hair fragment at the root of the corresponding hair,
- Irradiance is sampled at the root point of hairs when cache misses happen,
- Hair fragments don't define a surface themselves and don't provide a very meaningful normal. Therefore we don't apply any gradient when extrapolating the irradiance along each hair. However we can apply fake self-shadowing gradient based on the distance along the hair. There are many ways to do this and a good reference can be found here [Neulander].

For this to be possible, we simply store in our deep file the growing surface position and normal for each hair.

Fur and Grass (2/3)



 SIGGRAPH2008

 DREAMWORKS
ANIMATION SKG

This technique works quite well in practice for medium to short fur and captures occlusion / bounce from surrounding objects onto the hair. Unfortunately, there doesn't seem to be a generic coherence we can exploit similarly with longer hair. Also this technique doesn't capture fur self-occlusion although alternate techniques exist and are widely used for this purpose.

Fur and Grass (2/3)



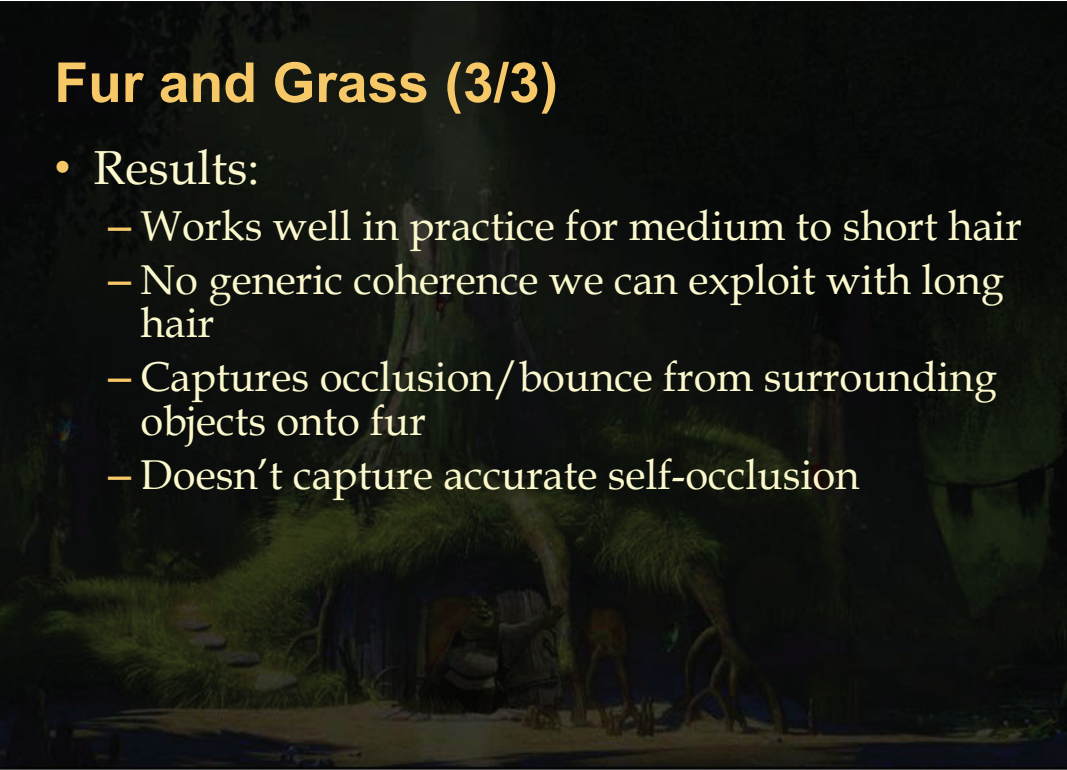
 SIGGRAPH2008



 DREAMWORKS
ANIMATION SKG

This technique works quite well in practice for medium to short fur and captures occlusion / bounce from surrounding objects onto the hair. Unfortunately, there doesn't seem to be a generic coherence we can exploit similarly with longer hair. Also this technique doesn't capture fur self-occlusion although alternate techniques exist and are widely used for this purpose.

Fur and Grass (3/3)

- Results:
 - Works well in practice for medium to short hair
 - No generic coherence we can exploit with long hair
 - Captures occlusion/bounce from surrounding objects onto fur
 - Doesn't capture accurate self-occlusion



This technique works quite well in practice for medium to short fur and captures occlusion / bounce from surrounding objects onto the hair. Unfortunately, there doesn't seem to be a generic coherence we can exploit similarly with longer hair. Also this technique doesn't capture fur self-occlusion although alternate techniques exist and are widely used for this purpose.

Render Statistics

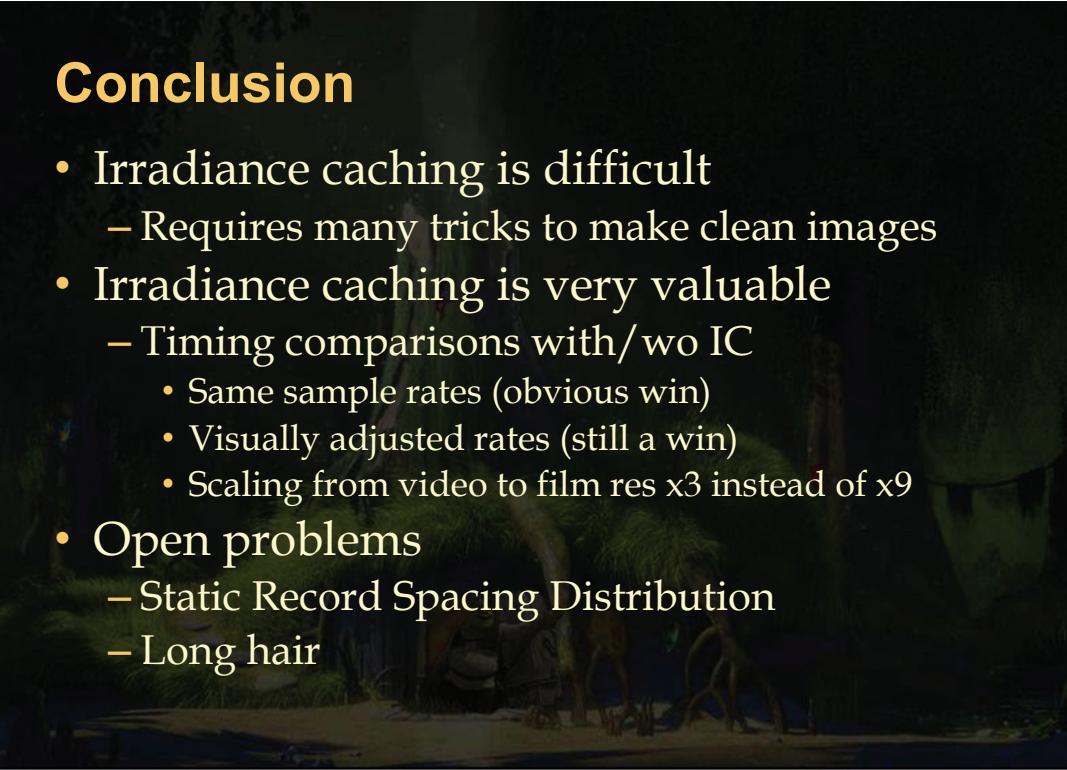
	Gathering	Shading	Total	Rays / Record	Cache Records	Rays / MP
Shrek AO - 1/3 res – wo IC	6:39	-	6:39	-	-	250
Shrek AO - 1/3 res	0:58	0:13	1:11	453	8282	22
Shrek AO - Film res	2:33	0:54	3:27	453	19294	7
Melman - 1/3 res	2:59	1:21	4:20	453	8559	18
Melman - Film res	6:44	4:03	10:47	453	16799	9
Shifu - 1/3 res	6:01	3:01	9:02	1000	17924	34
Shifu - Film res	15:48	9:32	25:20	1000	38187	22

- Timing comparisons with/wo IC:
 - Same sample rates (obvious win)
 - Visually adjusted rates (still a win)
- Excellent scaling wrt. image resolution





Here are some render times for the various image examples shown. They were all achieved on a AMD Opteron 2.14 GHz processor workstation running on a single core.

We also provide a comparison with/without using irradiance caching. In this comparison, we visually adjusted the number of rays per pixel to achieve a similarly noise-free image than when using irradiance caching.



Conclusion

- Irradiance caching is difficult
 - Requires many tricks to make clean images
- Irradiance caching is very valuable
 - Timing comparisons with/wo IC
 - Same sample rates (obvious win)
 - Visually adjusted rates (still a win)
 - Scaling from video to film res x3 instead of x9
- Open problems
 - Static Record Spacing Distribution
 - Long hair

As we have seen throughout this course, irradiance caching is not an easy topic and requires many tricks to work well and produce clean images. It is however a very valuable technique, as it provides many benefits in the context of an interactive lighting tool and pipeline. It also accelerates gathering calculations significantly as can be shown in our render time results above.

Direction for future work includes exploring static record spacing distributions on animated deforming characters, as well as finding ways to extract some form of coherence from long hair.

Acknowledgements

- Course Organisers
- Janne Kontkanen
- Jin Liou
- Mark Edwards
- Everyone at DreamWorks

