

Global Illumination Across Industries

*Ray Tracing vs. Point-Based GI
for Animated Films*

Eric Tabellion
et@pdi.com

 SIGGRAPH2010

 **DREAMWORKS**
ANIMATION SKG

Introduction

- Global Illumination (GI) usage at DreamWorks Animation
 - Bounce lighting
 - Ambient occlusion
 - HDR Environment map lighting (IBL)
 - Used on many films since “Shrek 2”
 - Used in a variety of shots
 - Various characters, props and environments
- Global Illumination system
 - Using Ray Tracing and Irradiance Caching (IC)
 - [Tabellion and Lamorlette 2004]
 - [Krivanek et al. 2008]
 - Using point-based GI
 - [Christensen 2008]

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

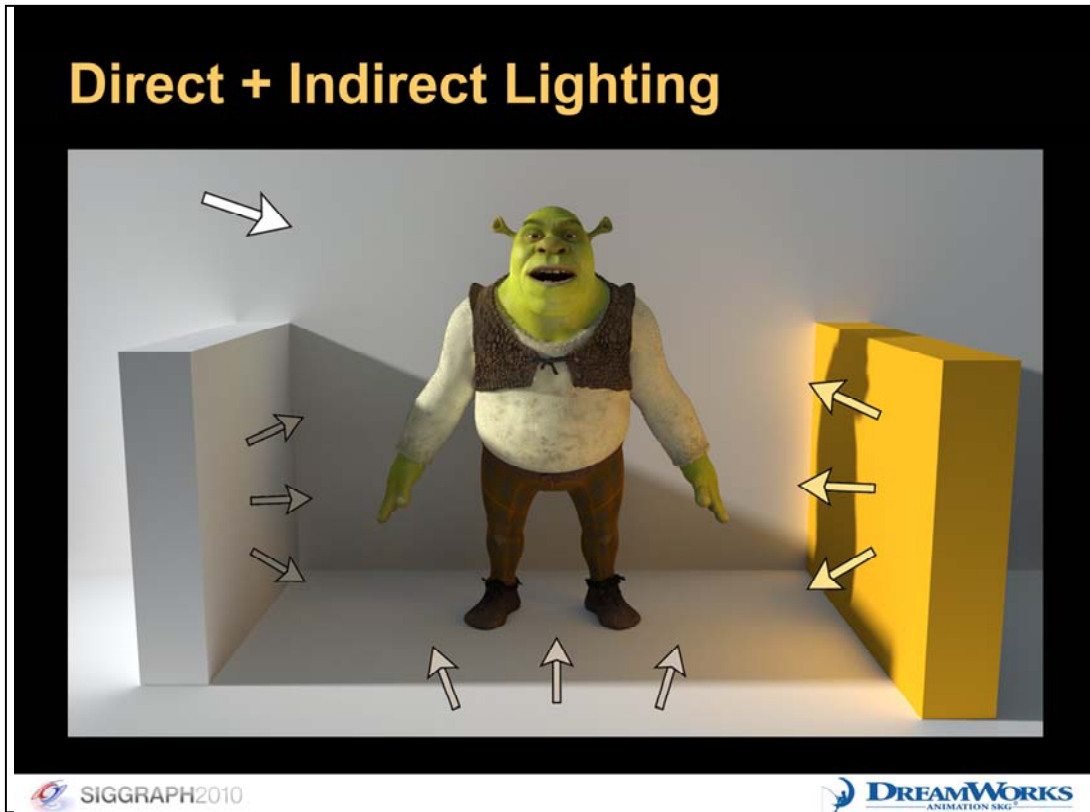
Global Illumination at DreamWorks Animation consists in simulating one bounce of diffuse interreflection, as well as ambient occlusion and environment-map image-based lighting.

We use a proprietary renderer to render all our films. GI has been used extensively at DreamWorks Animation to light and render many animated films since “Shrek 2”. Many implementation details of our global illumination system are published in our 2004 Siggraph paper [Tabellion and Lamorlette 2004] as well as in our more recent course notes [Krivanek et al. 2008].

First we review the GI requirements for animated film production. We then briefly describe our ray-tracing based system, and describe how irradiance caching fits within our rendering pipeline and how it integrates within a production renderer. We will show how specific solutions can help make IC up to an order of magnitude faster compared to final gathering at every pixel. We then describe the specifics of our point-based global illumination (PBGI) system, which is based on [Christensen 2008], and finally compare the strengths and weaknesses of both systems.



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 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.

Example: “How To Train Your Dragon ”



SIGGRAPH2010

DREAMWORKS
ANIMATION SKG

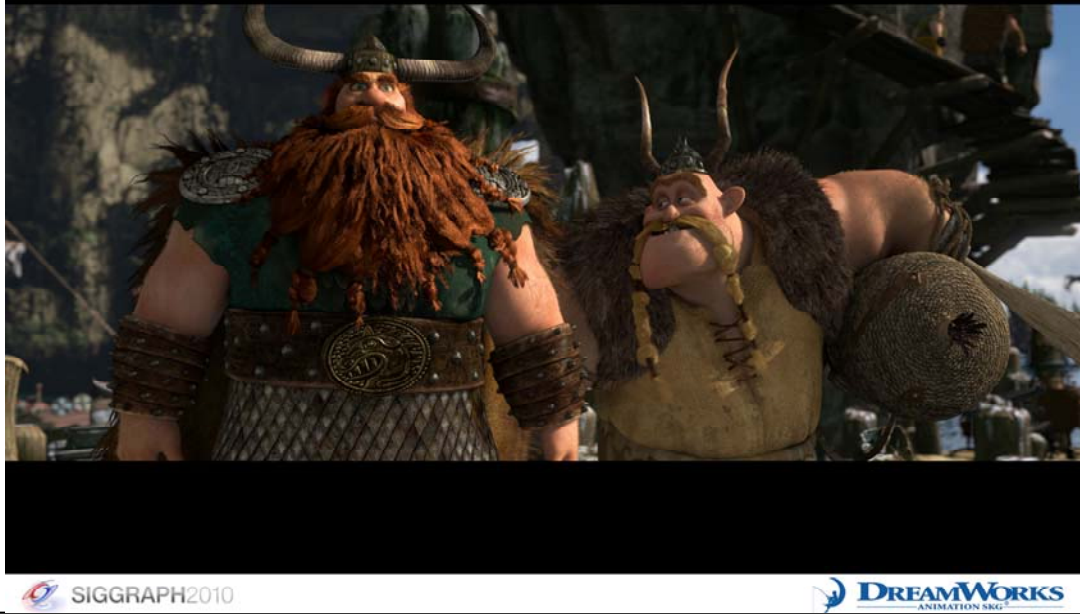
Here are some recent examples from images rendered with GI, from the movie “How To Train Your Dragon”.

Lighting credit:

Left image: Dan Levy, Melva Young, Mark Edwards

Right image: Jung Jin Song, Michel Kinfoussia

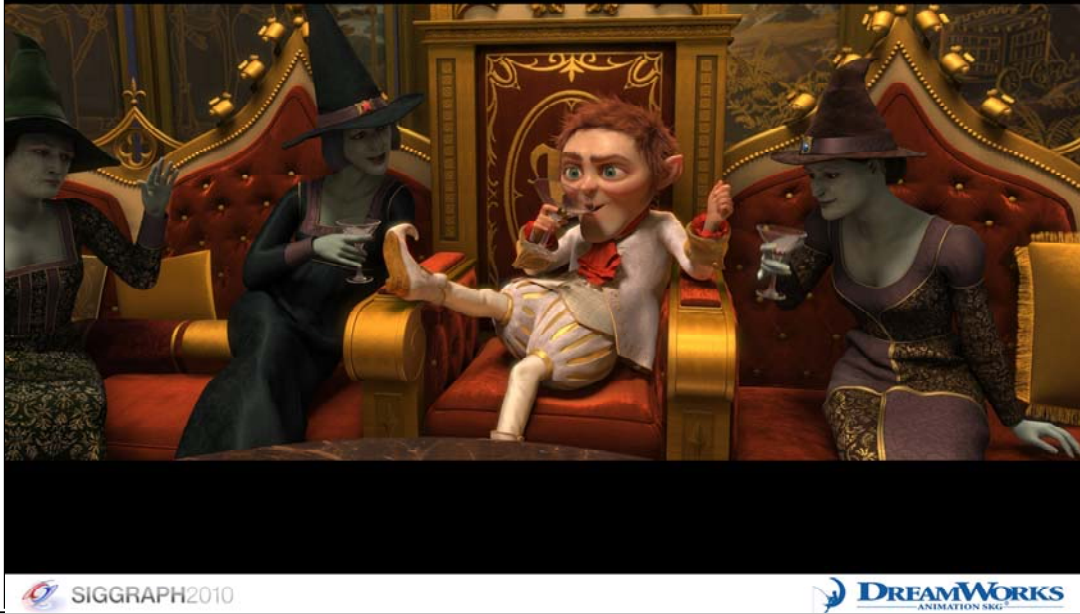
Example: “How To Train Your Dragon ”



Here are some recent examples of images rendered with GI, from the movie “How To Train Your Dragon ”.

Lighting credit: Liang-Yuan Wang, Mike McNeill

Example: “Shrek Forever After”



Here are some recent examples of images rendered with GI, from the movie “Shrek Forever After”.

Lighting credit : Benjamin Venancie, Archie Donato, Betsy Nofsinger

GI in Animated Film Production

- Requirements:
 - High Quality
 - No noise, buzzing or popping
 - Artistic control
 - Offer physically correct starting point
 - Let the user tweak shaders further
 - Good shader integration
 - Speedy interaction
 - Add bounce cards to control lighting
 - Apply localized filters to GI
 - Scene complexity
 - Scene complexity is unbounded (more or less tuned)

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

The requirements for GI in animated film production are quite drastic.

The resulting images have to be noise-free. There should be no buzzing or popping of any kind in the resulting animations. The tools should offer a good physically correct starting point for the indirect illumination. The tools also need to be very flexible. Lighters should be able to address all art-direction comments from their supervisors or from the director and remain in control of the image creation process. There must be a quick feedback loop when making adjustments, otherwise lighters shy away from using GI in their shots, and fall back to alternate lighting techniques (using many fake lights).

In our system, GI is rendered through a global bounce light shader. Its parameters affect the overall GI simulation, and provide global color correction controls that apply to the indirect illumination. Our toolset also contains bounce filter light shaders which can further control the indirect illumination coming from the main shader using a set of spatial and angular falloffs. There are also additional parameters in our surface shaders that control how much each surface bounces indirect light. The lighters can also add off-screen bounce cards or include a distant HDR environment map in the lighting simulation.

The scene complexity is usually not bounded in film production. The scene geometry, texture and other associated data created by many departments funnels down in the hands of lighters who need to get many shots lit and rendered. The scenes are not necessarily tuned for rendering performance or memory efficiency.

GI in Animated Film Production

- Multi-department Impact:
 - Modelling & Animation
 - Geometry with bad contact / penetrations
 - Surfacing
 - Account for GI as a lighting scenario
 - Define bouncing characteristics
 - FX
 - Effects do illuminate and occlude
 - Lighting
 - GI not always intuitive
 - Light is coming from everywhere
 - GI produces simpler lighting rigs
 - Propagates well to other shots / sequences

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

Using GI in film production has a non negligible impact on many departments.

Models with parts that don't join properly or interpenetrate each other are not acceptable as global illumination will tend to reveal these issues. Surfacing needs to sometimes define the bouncing characteristics of surfaces, and all surfacing must be tested with plausible indirect illumination scenarios. Particle or volumetric effects (i.e. fire and explosions) created by the FX department often should cast indirect light onto the neighboring hard surfaces or occlude indirect light passing-by.

For GI-beginners and sometimes more advanced users too, it is not always intuitive how to control a particular lighting effect, since light can be coming from all possible directions. On the positive side, using GI produces much simpler lighting rigs that require very few lights. The rigs tend to propagate very well to other shots within a sequence or across sequences, and provide an excellent lighting continuity across shots. This tends to be much harder with many "local / manual" indirect lights, especially as characters move around within an environment.

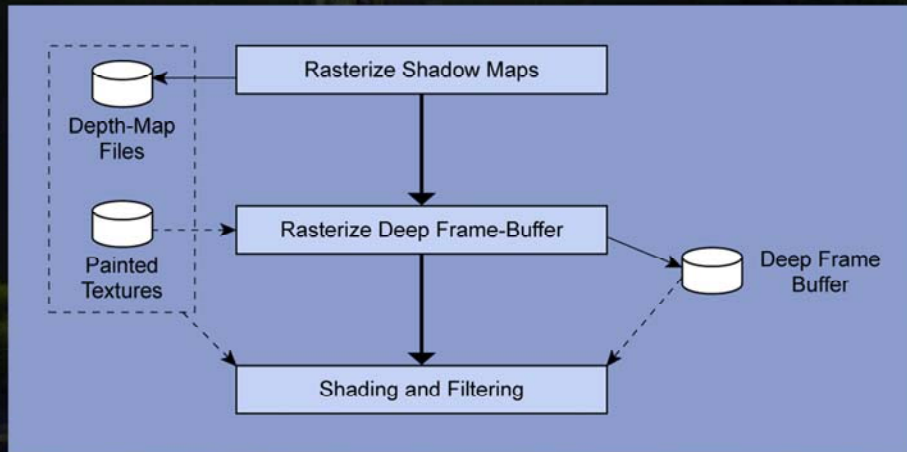
Ray-Tracing-Based GI

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

System Overview

- Micropolygon Deep Frame Buffer Renderer
- Use Ray Tracing and Irradiance Caching
- Lighting Tool for Interactive lighting



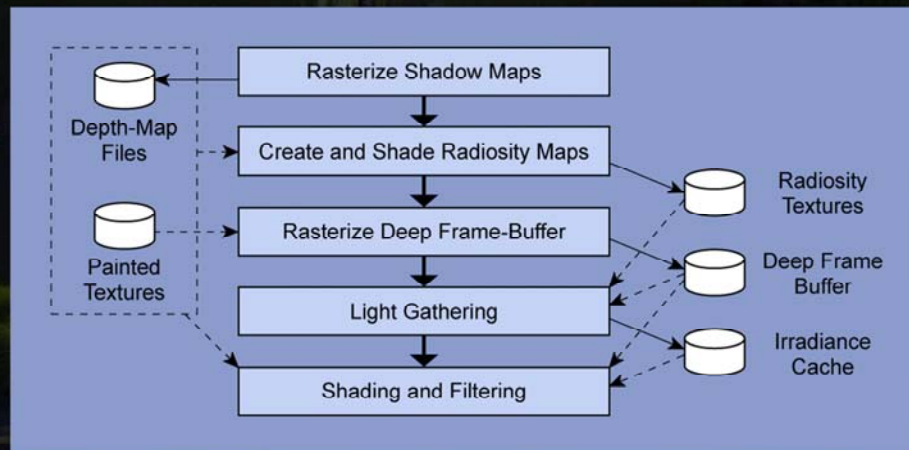
SIGGRAPH2010

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.

System Overview

- Micropolygon Deep Frame Buffer Renderer
- Use Ray Tracing and Irradiance Caching
- Lighting Tool for Interactive lighting



SIGGRAPH2010

DREAMWORKS
ANIMATION SKG

We enhanced our renderer with single-bounce global illumination capability by using a ray-tracing 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.

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 shadowed direct lighting 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 inter-reflections.

It is important to notice that using radiosity textures is optional in our system and shaders can be run instead at each ray intersection. In practice radiosity maps are always used for batch final quality rendering and are almost never used during daily interactive lighting workflow. In our lighting tool, the user can move lights and immediately re-run the "Gathering" and "Shading" of the image (or of small regions of the image). The lighters don't have to wait while re-baking the radiosity textures, which would otherwise slow down the feedback loop.

The lighters can also decide to only "re-shade" the image to fine-tune the position of a direct light. This avoids incurring the extra cost of re-gathering for small minor adjustments. Irradiance caching also provides a convenient way to accelerate this workflow.

Direct Lighting Only



Credit: "Shrek 2"

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

This example from the movie "Shrek 2" illustrates final gathering using irradiance caching. This image contains shadowed direct lighting only.



Without using Irradiance Caching, gathering calculations would 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.

Radiosity maps pros/cons:

- Shaded with direct illumination only
- Sets a limit to a single bounce
- Doesn't flicker in animation like photon maps
- Each ray radiance estimate is very cheap (a direct texture lookup vs. a kd-tree nearest neighbor search)

Indirect Lighting Only



Credit: "Shrek 2"

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

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

Indirect + Direct Lighting



Credit: "Shrek 2"

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

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

Geometry 2-Levels of Detail 1/3

- Primary visibility: rasterize micropolygons
- Secondary visibility
 - Raytrace coarser tessellation and no/coarse fur
 - Greatly increases renderable scene complexity
 - Use “smart bias”



SIGGRAPH2010

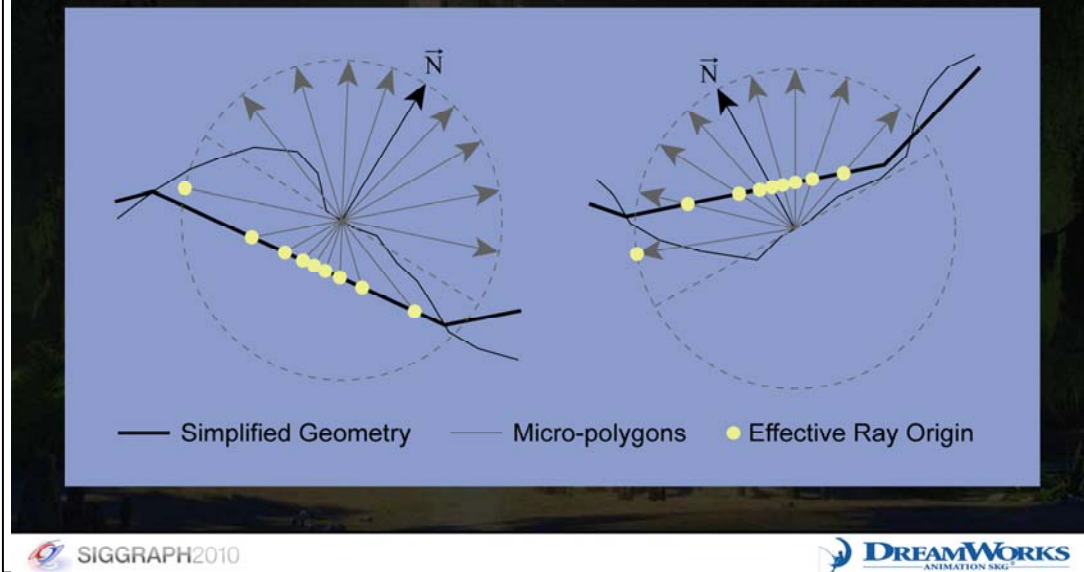
DREAMWORKS
ANIMATION SKG

We describe here one of the main ways in which we deal with geometric complexity. When rays are cast, we do not attempt to intersect the geometry that is finely tessellated down to pixel-size micropolygons - this would be very expensive and use a lot of memory. Instead we tessellate geometry using a coarser simplified resolution and the raytracing engine only needs to deal with a much smaller set of polygons. This greatly increases the complexity of the scenes that can be rendered with a given memory footprint, and also helps increase ray-tracing efficiency.

To avoid artifacts due to the offset between the two geometries, we use a “smart bias” algorithm described in the next slides.

Geometry 2-Levels of Detail 2/3

- Ray Origin Offsetting Algorithm



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. The full algorithm is described in [Tabellion and Lamorlette 2004].

Geometry 2-Levels of Detail 3/3

Raytracing 2 million
micropolygons



Raytracing 2 thousand
polygons



 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

Here is a comparison between a reference image that was rendered while raytracing the micropolygons. The image in the center was rendered with our technique while raytracing the coarser polygon tessellation illustrated in the image on the right.

Geometry Multiresolution LOD

- Multiple coarser levels of detail
 - [Christensen 2003]
- Use even coarser tessellations
- Use for larger ray footprints
- Problems:
 - Based on tessellation LOD only
 - Often need less polys than primitives

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

It has been shown in [Christensen 2003] that it is possible to use even coarser and coarser tessellations of the geometry to intersect rays with larger ray footprints. This approach is also based on using only coarser tessellation and is not able to provide very coarse tessellations with much fewer polygons than there are primitives (fur, foliage, many small overlapping surfaces, etc.), which would be ideal for large ray footprints.

This problem is one of the main limitations of ray-tracing based GI in very complex scenes and is addressed by the clustering approach used in point-based GI, as discussed in later slides.

Irradiance Caching (IC)



When we use irradiance caching, the expensive GI 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.

Irradiance Caching (IC)



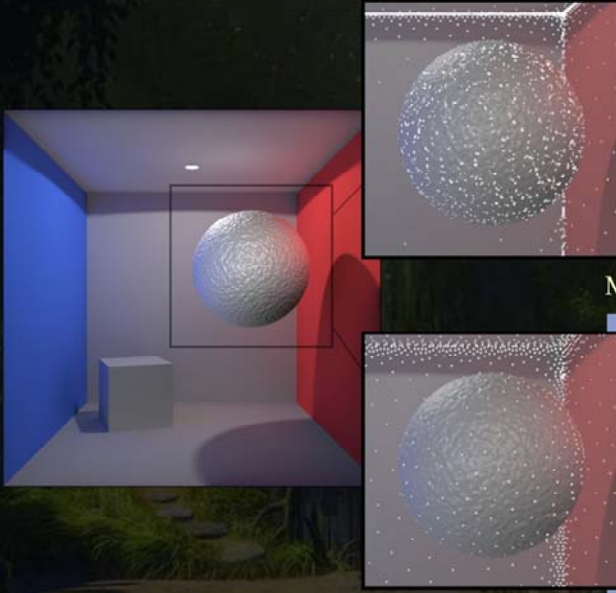
Credit: "Madagascar: Escape 2 Africa"

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

Here are is an example of Melman with and without global illumination, as well as the corresponding irradiance cache.

IC Error Metric



[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

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

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

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

SIGGRAPH2010 DREAMWORKS ANIMATION SKG

As described in our paper [Tabellion and Lamorlette 2004], 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. 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.

The error metric is also enhanced to add or remove constraints. In [Krivanek et al. 2008] we describe how we can modify the error metric further to maintain a sparse record spacing distribution on surfaces that have bump, are displacement mapped, or even objects with fur.

IC Record Spacing

- Record spacing flickers
- 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

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

The main challenge in implementing irradiance caching is providing a good record spacing quality.

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 flickering distributions.

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 using irradiance caching. Using irradiance caching pays off as long as the record spacing remains sparse enough to offset this cost.



Here is the record spacing distribution we can achieve, even on displaced objects like Shrek's vest.



Here is the result of bounce lighting + environment lighting applied to the displaced un-textured surfaces...

IC with Displacement



Credit: "Shrek The Third"

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

Which is what was used to render this final image.

IC with Fur / Grass



Credit: "Kung Fu Panda"

SIGGRAPH2010

DREAMWORKS
ANIMATION SKG

Similarly it is possible to keep the record spacing distribution sparse and on the surface of the object growing the fur as in this example. The irradiance is then used to illuminate the fur growing from that surface.

IC with Fur / Grass



Credit: "Kung Fu Panda"

SIGGRAPH2010

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. We will see in later slides how we use point-based ambient occlusion to "bake" the fur self-occlusion, which is then used in this type of ray tracing-based bounce lighting render.

Irradiance Caching Performance

	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



SIGGRAPH2010

DREAMWORKS ANIMATION SKG

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

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

Notice the right-most column of the table, which indicates the average number of indirect rays used to compute indirect illumination for each micropolygon during final quality rendering. It is the result of a relatively high number of rays per cache records, amortized over many pixel-sized micropolygons.

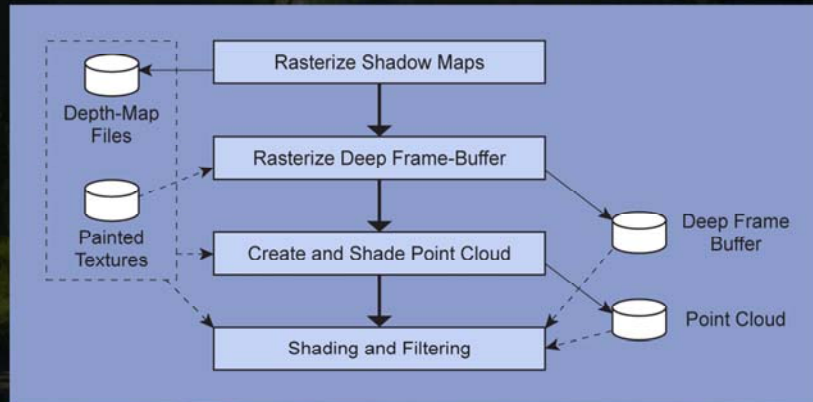
Point-Based GI

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

Point-Based GI (PBGI)

- Implementation basics [Christensen 2008]
- Same integration as with Ray Tracing
- Easy to switch Ray Tracing / Points



SIGGRAPH2010

DREAMWORKS
ANIMATION SIG

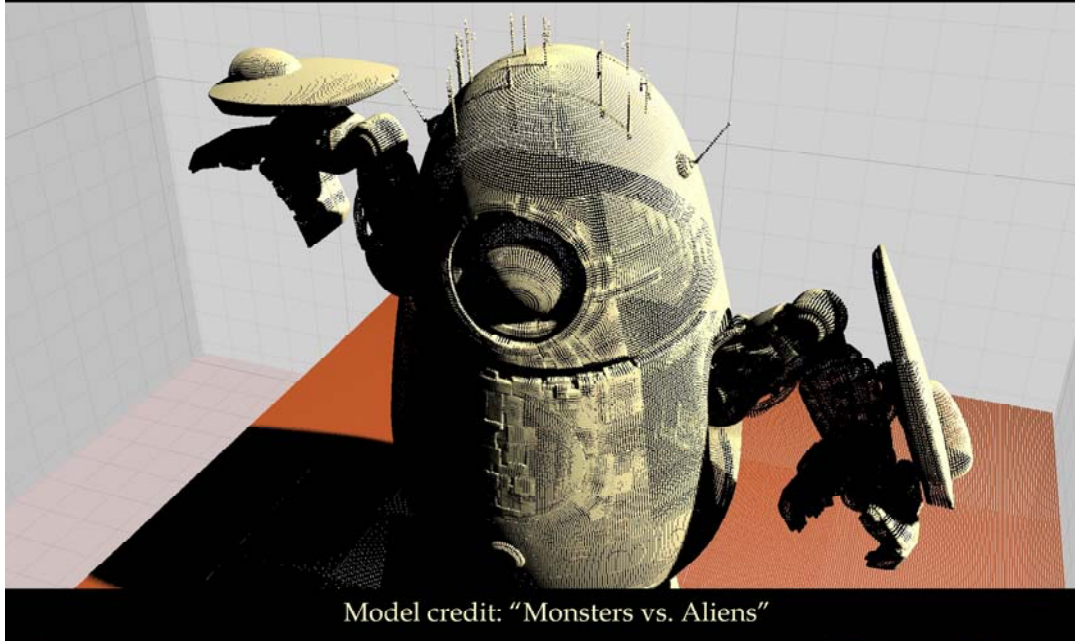
We also enhanced our renderer with a Point-Based GI system (PBGI), following some of the implementation basics described in [Christensen 2008].

We use the same light shader integration as our raytracing-based system, which makes it easy to switch between ray-tracing and point-based global illumination.

The dense point cloud represents the lit geometry that affects indirect illumination. It is enhanced before shading starts, by a clustered representation of the point cloud at many coarser levels of detail. The clustering can aggregate points that originate from different primitives / objects and is therefore much more powerful at reducing complexity for very coarse representations than a tessellation-based approach. This is the main advantage of point-based over raytracing-based GI.

Being also the analog to the radiosity maps used in our ray-tracing system, the point cloud must be re-shaded every time the lighters move a light. The point cloud data is not optional like the radiosity maps are and re-shading the whole point cloud can seriously slow down the interactive workflow.

Point-Based GI (PBGI)

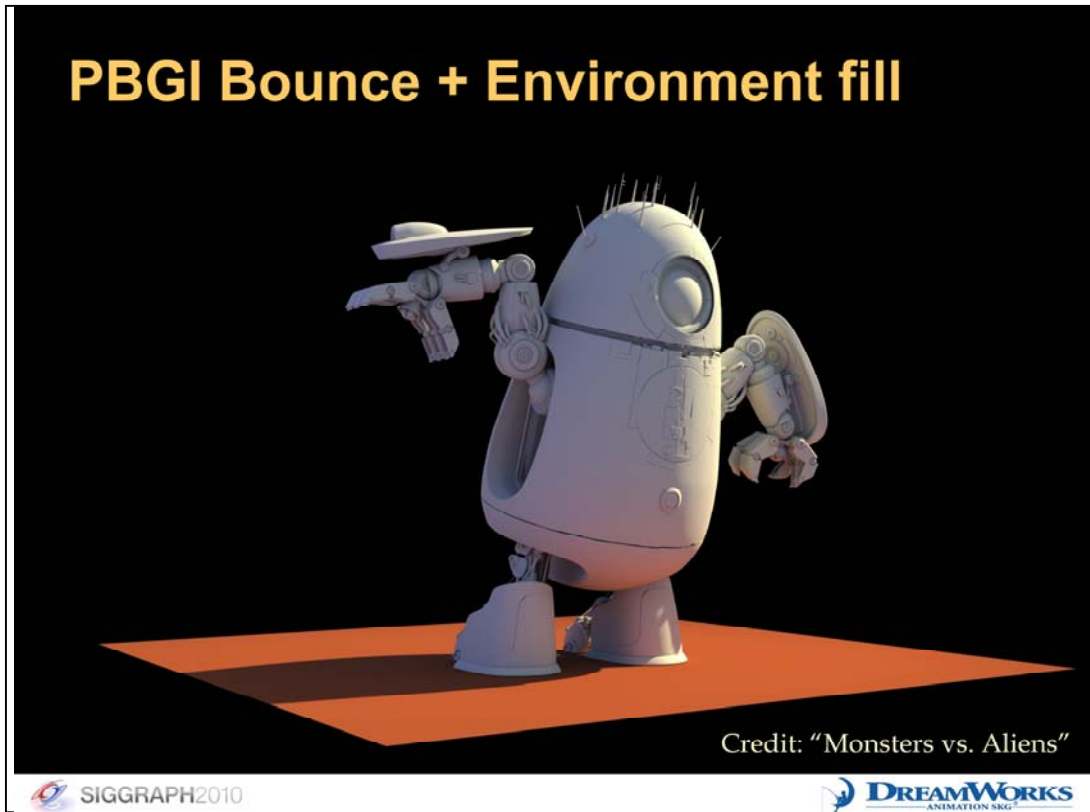


Model credit: "Monsters vs. Aliens"

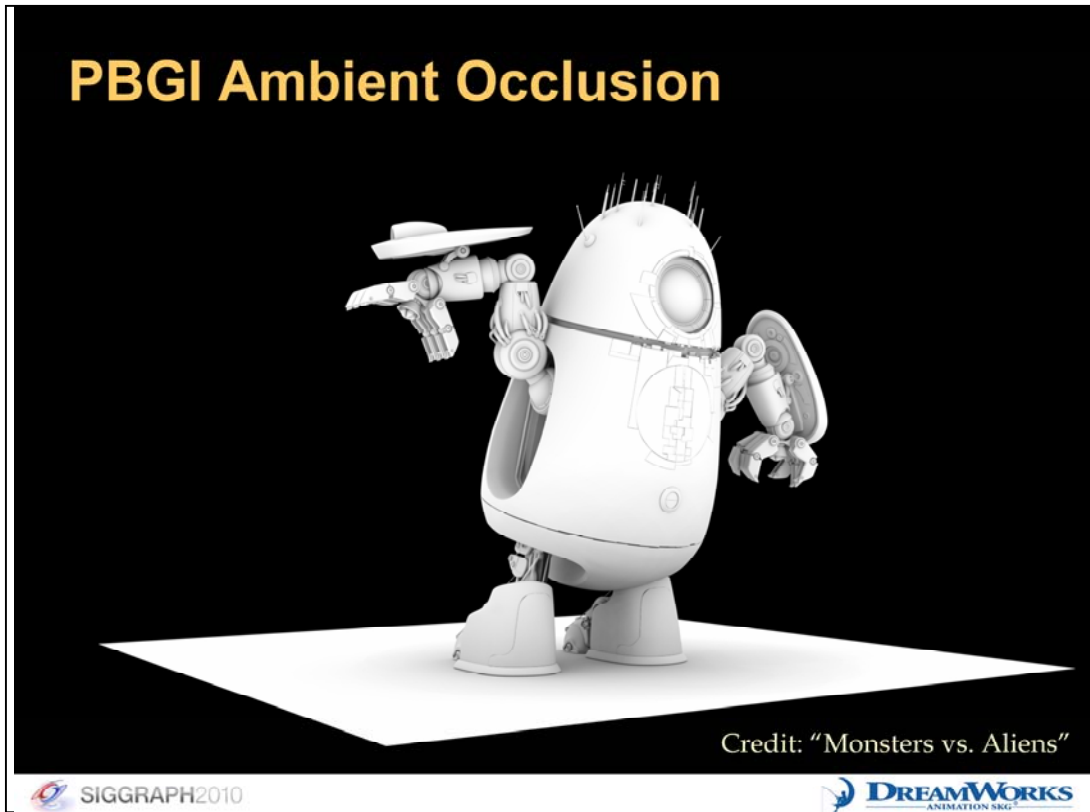
SIGGRAPH2010

DREAMWORKS
ANIMATION SKG

Here is an example of a dense point cloud for a robot model. Since we process dense points over the surface of the geometry, the technique is well suited to adding fine surface detail with displacement maps.



Here is an example of PBGI bounce lighting, using also a distant environment map. The cost of adding the environment to the point-based simulation is virtually free as described in [Christensen 2008]. It is also possible to render large textured area lighting with soft shadows, by generating lit points on the surface of the area light.



Here is an example of ambient occlusion rendered with the same point cloud data. By adjusting the surface normal and spread of directions that we use to compute the ambient occlusion integral, we can easily render directional ambient occlusion or glossy reflection occlusion passes.

Point Generation 1/5

- Use micropolygon “split & dice” tessellator
- Generate one point per micropolygon
- Need points everywhere
- Include FX point clouds
 - Fire, explosions, etc.
- Optimal point count for maximum LOD
 - In-view: Pixel-size points
 - Out-of-view: Much fewer / bigger points
- Change tessellation
 - Disable culling
 - Cannot use perspective projection
 - Work in camera space

SIGGRAPH2010

DREAMWORKS
ANIMATION SKG

Generating point clouds appropriate for PBGI with good density poses interesting challenges.

Like in [Christensen 2008], we use the camera-adaptive micropolygon tessellator of our renderer and generate one point per micropolygon. This allows us to achieve, if desired, a one-to-one mapping between micropolygons and points.

Our solution however, is able to generate points everywhere in the scene: inside and outside the camera view-frustum. This removes the need for manually placing an alternate distant camera that encompasses the entire scene. It also produces points in-view that are pixel-sized considering the size of the pixels from the actual rendering camera (and not considering the alternate distant camera). Our solution also produces much fewer and bigger points outside the camera view-frustum and ensures that occlusion or indirect light coming from off-screen surfaces is still faithfully represented.

Inside the camera frustum, using pixel-sized points is necessary to model indirect-lighting and surface self-occlusion with pixel-size surface detail. In practice, using points approximately 1.5 times the size of a pixel produces roughly half the point count. It also maintains enough detail and doesn't cause self-occlusion artifacts.

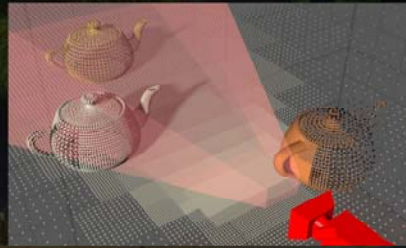
Outside of the camera frustum, we want to achieve an optimal point count and respect the LOD “max-solid angle” metric that will be used to pick appropriate distant point clusters. This means we need to generate points that are no smaller than the size of a valid cluster that would be used to illuminate any point inside the camera frustum. In order to generate micropolygons / points outside the view frustum at all, we are also required to disable view-frustum culling from the REYES “split and dice” tessellation algorithm. In turn, this means the tessellator can no longer use the perspective projection to project objects onto pixels and evaluate their size. The perspective divide would otherwise cause numerical issues and produce points with infinite density for surfaces close to the camera focal plane – also called the eye plane. Working in projected / screen space also causes “point holes” on surfaces at grazing angle, which can help reduce the point count but can cause very undesirable light / occlusion leaks.

To address these problems, we modify tessellation to work with surfaces in camera space.

Point Generation 2/5

- Solid angle to surface area
- Surface in-view:
 - d = Distance from focal point
 - ω = “center pixel” solid angle
- Solve desired point area
 - Ignore surface orientation
 - $A = \omega * (d*d)$
- Compute surface size
- Derive dice rate

$$\omega = \frac{A \cdot \cos \theta}{d^2}$$



SIGGRAPH2010

DREAMWORKS
ANIMATION SKG

To compute the size (and density) of surface elements that are in-view, we first compute the distance:

$$d = \text{MAX}(\text{distance}(\text{camera focal point, surface element}), \text{near plane distance})$$

Using the solid angle of the “center pixel”, and making the approximation that all pixels have the same solid angle, we can solve for the desired pixel-size point area. To avoid point holes at grazing angles, we can ignore surface orientation by ignoring the cosine term from the solid angle formula: $\text{solidAngle} = \text{area} / d^2$. Using the desired point area and an estimate of the surface size, it is then trivial to derive a corresponding dice rate that will produce pixel-size points / micropolygons.

To compute the size (and density) of surface elements that are out-of-view, we apply the same reasoning using the distance to the closest point on the camera frustum, and the “max-solid angle” used later in the LOD cut picking algorithm. Computing the closest point on the camera frustum can be done very efficiently for many successive surface elements that are spatially coherent. An amortized constant-time algorithm is described in [Mirtich 1998] and can be special-cased for a camera frustum that forms a convex polyhedra.

The resulting point density is illustrated in the image on top for the simple scene rendered in the image on the bottom. The top image intentionally does not show the size of each point in order to better represent the overall point density. Notice that the tessellator’s “split and dice” algorithm adapts discretely to the variations in desired point size / density. Surface patches in-view are tessellated into almost pixel-size points. Patches out-of-view are tessellated with fewer and fewer points based on distance to the camera frustum, with a rate of decay that is proportional to the LOD cut picking metric. This metric guarantees a faithful maximum possible level of detail outside of the camera frustum with an optimal number of points and consequently does not introduce any flickering in animations.

Point Generation 3/5

- Surface Out-of-view:
 - d' = Distance to closest point in-view
= Closest point on view frustum polyhedra [Mirtich 1998]
 - ω' = “max-solid angle” LOD quality
- Solve desired point area
 - Ignore surface orientation
 - $A' = \omega' * (d' * d')$
- Compute surface size
- Derive dice rate



SIGGRAPH2010

DREAMWORKS
ANIMATION SKG

To compute the size (and density) of surface elements that are in-view, we first compute the distance:

$$d = \text{MAX}(\text{distance}(\text{camera focal point, surface element}), \text{near plane distance})$$

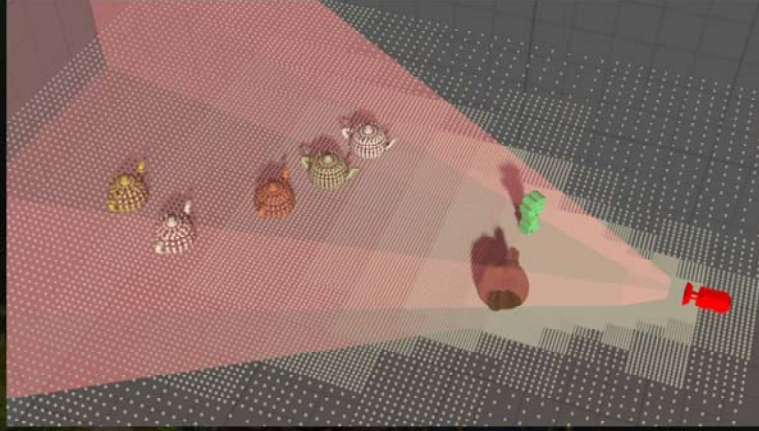
Using the solid angle of the “center pixel”, and making the approximation that all pixels have the same solid angle, we can solve for the desired pixel-size point area. To avoid point holes at grazing angles, we can ignore surface orientation by ignoring the cosine term from the solid angle formula: $\text{solidAngle} = \text{area} / d^2$. Using the desired point area and an estimate of the surface size, it is then trivial to derive a corresponding dice rate that will produce pixel-size points / micropolygons.

To compute the size (and density) of surface elements that are out-of-view, we apply the same reasoning using the distance to the closest point on the camera frustum, and the “max-solid angle” used later in the LOD cut picking algorithm. Computing the closest point on the camera frustum can be done very efficiently for many successive surface elements that are spatially coherent. An amortized constant-time algorithm is described in [Mirtich 1998] and can be special-cased for a camera frustum that forms a convex polyhedra.

The resulting point density is illustrated in the image on top for the simple scene rendered in the image on the bottom. The top image intentionally does not show the size of each point in order to better represent the overall point density. Notice that the tessellator’s “split and dice” algorithm adapts discretely to the variations in desired point size / density. Surface patches in-view are tessellated into almost pixel-size points. Patches out-of-view are tessellated with fewer and fewer points based on distance to the camera frustum, with a rate of decay that is proportional to the LOD cut picking metric. This metric guarantees a faithful maximum possible level of detail outside of the camera frustum with an optimal number of points and consequently does not introduce any flickering in animations.

Point Generation 4/5

- Show Video!



SIGGRAPH2010

DREAMWORKS
ANIMATION SKG

During the course we show animations of a camera-traveling scene rendered with PBGI and the point generation technique described in the previous slides. We also show the point distributions generated with this technique animated over time. This slide shows the point distribution for two different frames. The camera is in solid red and the camera frustum is semi-transparent red.

Point Generation 5/5

- Do it twice for stereo cameras
- No need to stitch
- Need less points than primitives ?
 - Apply “Russian-roulette”
- Modified geometry generators
 - Fur, Grass, Tree
 - Generate less primitives
 - Generate larger points
- FX particles
 - Fire, explosions, etc.

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

To support stereo cameras, the algorithm must be applied for each eye since the camera frustum of a stereo camera may be non-convex (due to stereo convergence).

Since we are generating points, we don't need to generate water-tight tessellated surfaces and any operations related to stitching, if any, can be disabled during tessellation. This may avoid unnecessary tessellation constraints required to enforce stitching, and help reduce further the point count.

Our point generation technique will produce at least one point per primitive. In cases where less points than primitives are desired, it is possible to apply “russian roulette” and kill points probabilistically (ex: small primitives really far from the camera). In our system we also modify our geometry generators to output less primitives and larger points according to user settings (ex: fur, grass, foliage).

In our pipeline, the effects animators also generate point cloud animations containing bright fire or explosion particles. These are used to light the scene geometry and greatly help integrate fire effects dynamic illumination within the overall scene lighting.

Point Clustering

- Fast Spherical Harmonics point projection
 - Point = oriented and scaled “cosine lobe”
 - Radially symmetric around Z axis
 - Use Zonal Harmonics rotation [Sloan et al. 2005]
- Use cluster average position
 - More accurate than octree-cell center
 - Less clusters “above the horizon”
 - Smaller cuts

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

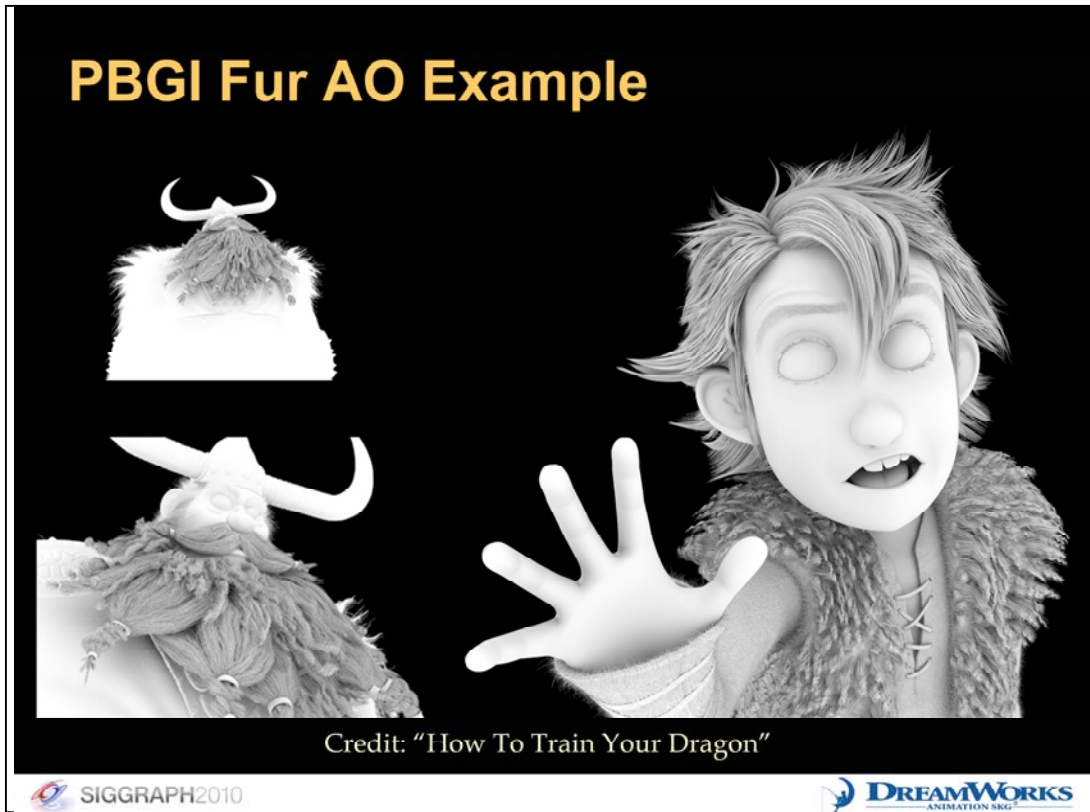
Point clustering works by sorting points into an octree and projecting each point's directional representation into spherical harmonics (SH) bases. The projection can be expensive with many points and further slow down the lighter's workflow. Fortunately, each point is represented as a scaled and oriented “cosine lobe”, which is a radially symmetric function around the point normal. This function can therefore be projected into SH with just a few multiplications by using Zonal Harmonics rotation, as described in [Sloan et al. 2005].

In our system we also keep track per cluster of the average position of the underlying points. This is more accurate than considering the octree-cell center, and can produce smaller cuts when shading convex objects as less clusters will bleed “above the horizon”.



Here is an example from “Shrek forever after” rendered using PBGI. During the course we show a video with this shot in animation, as well as the animated point distribution that was used to render this shot.

Lighting credit: Pablo Valle, William Arias



PBGI was used extensively on "How To Train Your Dragon" to bake fur ambient-occlusion. The baked occlusion is then used to compensate for the lack of fur occlusion when using ray-tracing based bounce lighting, when fur-less character geometry is ray-traced. This pipeline adds a lot of detail in all the characters hair, beards and furs and leverages the strengths of both GI systems. During the course we will show a few shots from the film that illustrate the technique.

PBGI Foliage AO Example



Credit: "How To Train Your Dragon"

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

The technique can also be extended to foliage. This is an example of PBGI ambient-occlusion on a complex large-scale forest scene.

PBGI Fire Lighting Example



Credit: "How To Train Your Dragon"

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

This is an example of dynamic fire particles, lighting the scene geometry. During the course we will show the shot in animation with the fire particles falling down and producing a very consistent lighting.

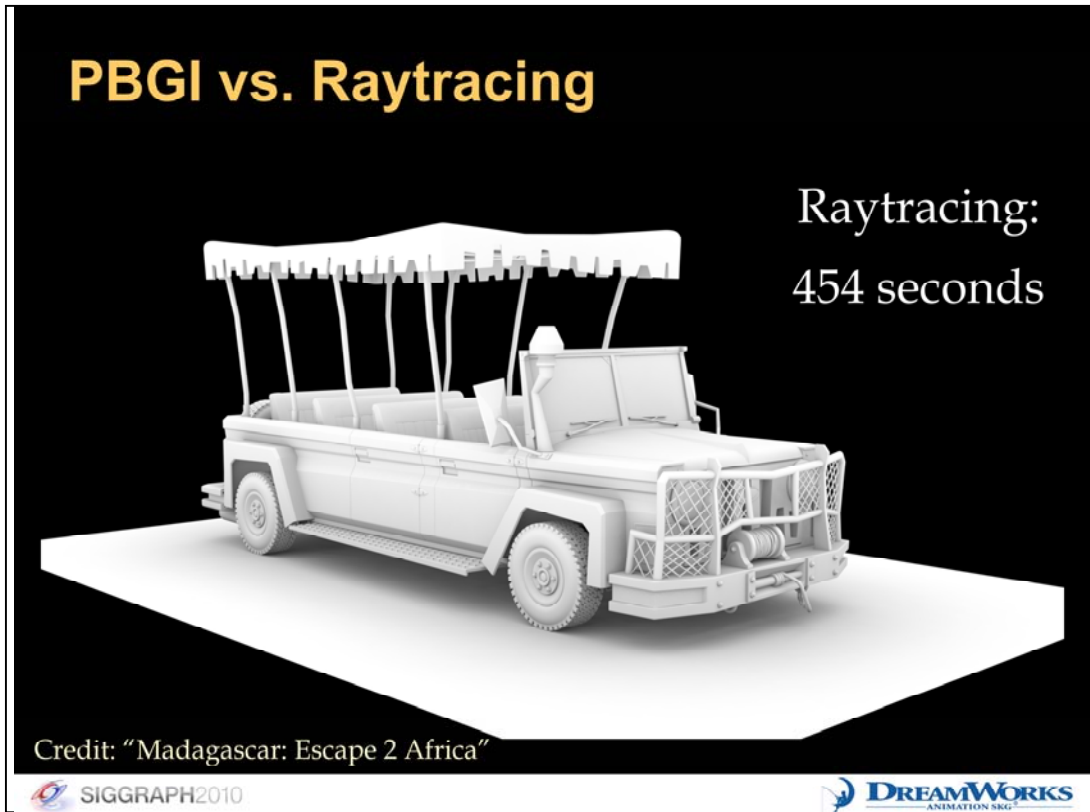
Lighting credit: Onny Carr, Igor Lodeiro, Udai Haraguchi, Mark Edwards

Dragons Fire Lighting Example



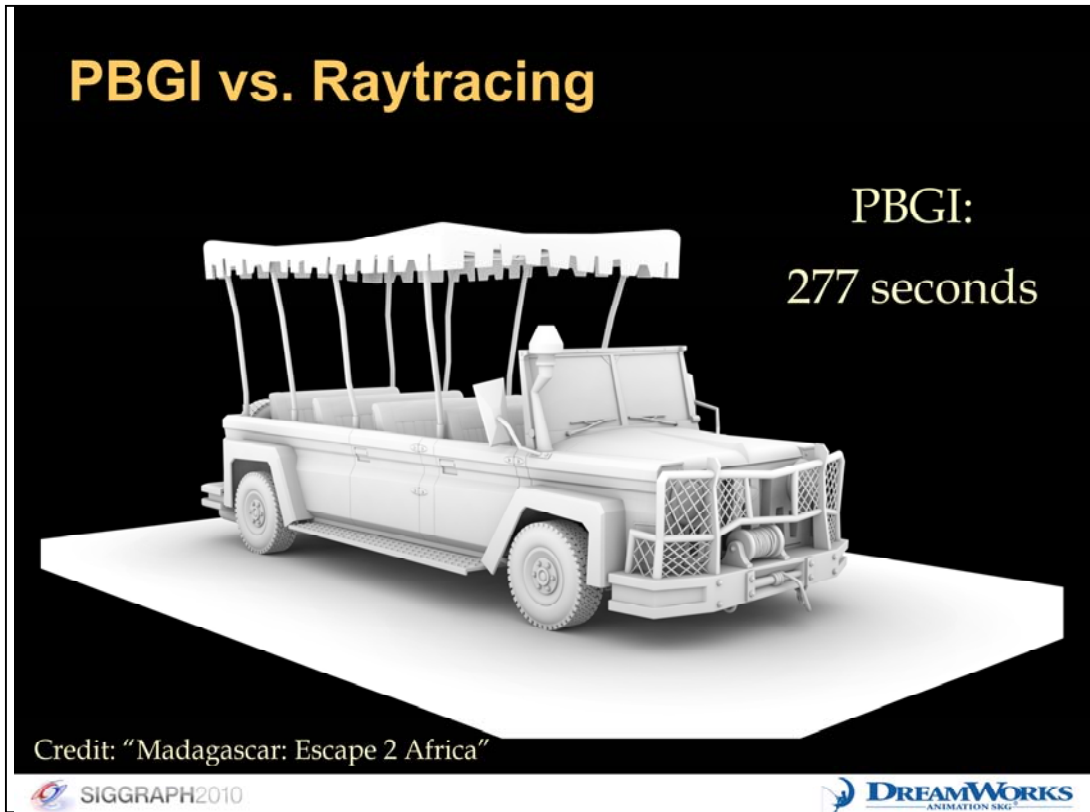
This is another example of fire lighting and occlusion using PBGI. The yellow lighting inside the dragon's mouth, on the crowd, on the rocks and on the boats is mostly coming from the FX fire particles. During the course we will show the sequence in animation.

Lighting credit: Igor Lodeiro, Bert Poole

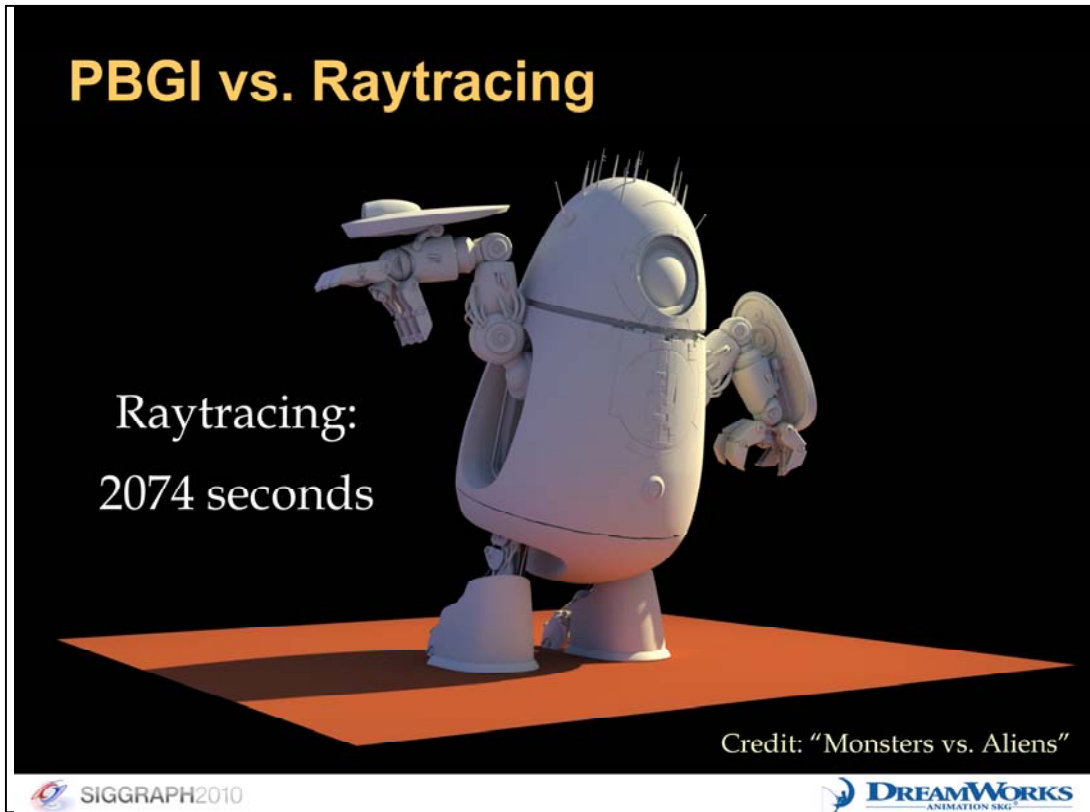


We compare the performance of both of our systems on two test scenes. The scenes have a mix of flat surfaces where ray-tracing with irradiance caching performs really well and higher frequency detail where PBGI is better suited. Both PBGI and ray-tracing quality settings were adjusted to achieve comparable visual quality.

Render times are provided for the frame total render time on a single Intel Xeon core 2.93 Ghz, and includes all the pre-processing steps required by each technique, for an image resolution of 2k x 1k pixels. The total render time is usually much faster when using many cores, but providing results on a single core offer a better reference for comparison.




Notice how with PBGI the front of the car is slightly over-occluded compared to the raytraced image. The over-occlusion is due to the approximate PBGI clustering, but doesn't represent an objectionable artifact. PBGI is about 2x faster than ray-tracing In this case.



Similarly, here is the bounce lighting example showed earlier in the slides.

PBGI vs. Raytracing



PBGI:
436 seconds

Credit: "Monsters vs. Aliens"

SIGGRAPH2010

DREAMWORKS
ANIMATION SKG

PBGI is more than 4x faster than ray-tracing in this example.

PBGI Pros

- Faster render times
 - Up to 4x faster than raytracing in some shots
- Bias vs. noise = image stability
- Unified framework for many effects
- Good at “large ray footprint” effects
 - Softer effects are cheaper
- Geometric detail doesn’t increase cost
- Spatial clustering more effective than tessellation LOD
 - Aggregates primitives together
 - Aggregates shading too
- Only deal with points
 - Vs. primitives + triangle meshes + textures
- Can be made to work coherently out-of-core

 SIGGRAPH2010

 DREAMWORKS
ANIMATION SKG

Finally, we summarize the advantages (Pros) and drawbacks (Cons) of PBGI compared to ray-tracing based GI

PBGI Cons

- Bias vs. noise = intensity shift
- Not good at “small ray footprint” effects
 - Sharp shadows, reflections, refractions
- Lazy building not straightforward
 - Rely on point pre-processing
 - Use upper-bound on maximum needed LOD
 - Not optimal everywhere (back-facing, occlusion-culled geometry)
- Clustering limitations
 - Order 3 SH ringing
 - Circular banding artifacts (Cell / LOD jump)
 - Objects or HDR env maps with hot spots
- So many points!
 - Requires out-of-core implementation

SIGGRAPH2010 DREAMWORKS ANIMATION SKG

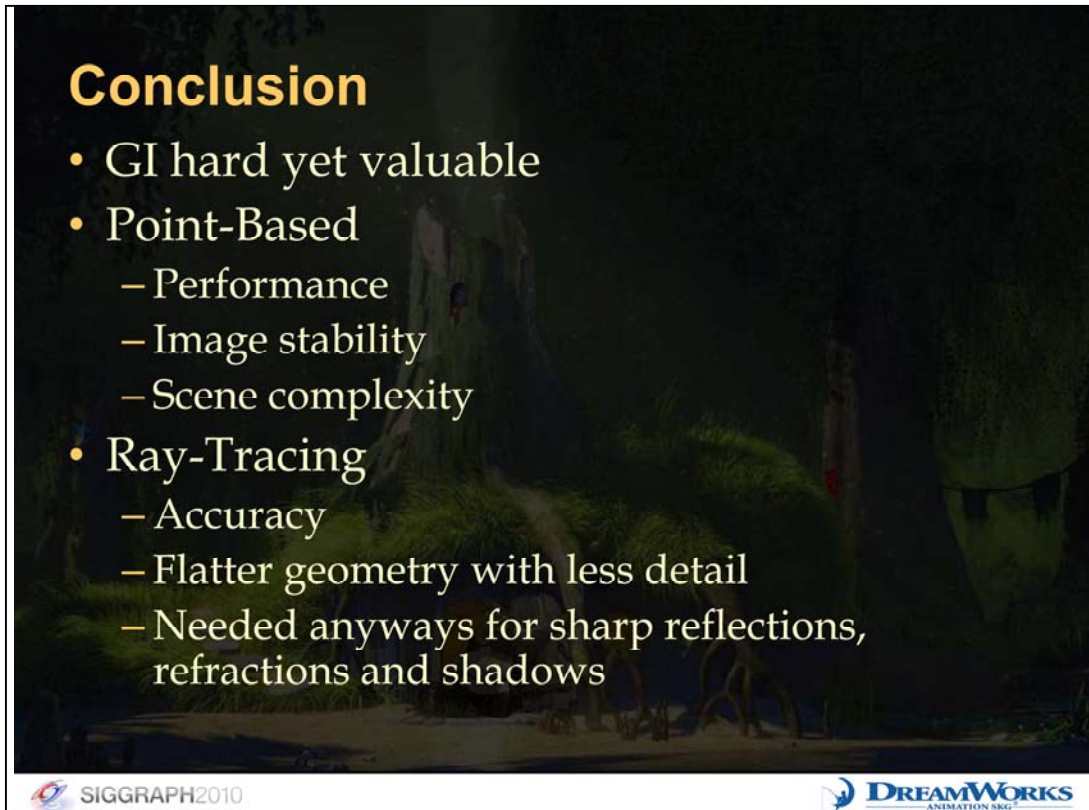
PBGI at lower quality settings generates a biased solution, whereas ray-tracing based GI generates noise. At high quality, the bias will be reduced and may shift lighting intensities unexpectedly. Noise on the other end is more predictable as it will consistently converge to the same noise-free estimate. This makes it less predictable to work interactively with low quality settings with PBGI.

It is not obvious if it is possible to efficiently generate lazily the point or cluster representation for a specific octree-cell and a specific LOD level. So far we rely on pre-processing of all the geometry and therefore need to output points according to the maximum needed LOD. This is not optimal and potentially we need to deal with too many points for surfaces that may be back-facing or occlusion-culled.

Order 3 Spherical Harmonics can only faithfully represent low frequency directional information. Rotating each point before the projection or clustering several points together with different orientations and power might create ringing problems that may show as errors in the reconstructed signal. Additionally, using HDR

environment maps with small hot spots (i.e. a sun light) or assigning high power to small objects in the scene creates spikes of light. Since the cut picking algorithm does not use bi-linear or tri-linear interpolation across octree cells or levels, this can result in "cluster jumps" along the cut that show as circular banding artifacts in the indirect illumination. This happens only occasionally in the most extreme lighting cases.

With PBGI, the fundamental decision is to treat geometry as colored pixel-sized points. In many cases with flat surfaces or with less surface texture detail, the point representation can use much more memory than a coarse polygonal representation, as mentioned in earlier slides. The need for an out-of-core implementation appears with simpler scenes with PBGI than it does with raytracing. This is not a serious limitation though, especially since production scenes tend to have a lot of high frequency surface detail. Furthermore, to PBGI's credit, ray-tracing-based GI does also rely on out-of-core access of the texture data and in some implementations of the geometry and acceleration structures.



Conclusion

- GI hard yet valuable
- Point-Based
 - Performance
 - Image stability
 - Scene complexity
- Ray-Tracing
 - Accuracy
 - Flatter geometry with less detail
 - Needed anyways for sharp reflections, refractions and shadows

SIGGRAPH2010

DREAMWORKS ANIMATION SKG

Besides harsh requirements, GI remains very valuable for animated film production lighting. We've analyzed both ray-tracing and point-based GI approaches and provided insights on the tradeoffs and example usage of both techniques in a production environment.

Overall, ray-tracing is more appropriate for applications seeking higher accuracy or when scenes have large flat surfaces without too much high-frequency texture detail. Points offer a close approximation and are a very appealing alternative for film production. Both approaches produce images that are visually very similar, but point-based GI produces very stable images more efficiently while handling higher scene complexity.

Even if all diffuse and semi-glossy GI effects are rendered using PBGI, ray-tracing is still needed for rendering sharp reflections, refractions or shadows. With this in mind, production rendering engineers might find it a good choice to implement a ray-tracer that is especially optimized at tracing packets of very coherent rays.

References

[Christensen et al. 2003]: Ray Differentials and Multiresolution Geometry Caching for Distribution Ray Tracing in Complex Scenes

[Christensen 2008]: Point-Based Approximate Color Bleeding

[Křivánek et al. 2008]: Practical Global Illumination With Irradiance Caching

[Mirtich 1998]: V-Clip: fast and robust polyhedral collision detection

[Sloan et al. 2005]: Local, deformable precomputed radiance transfer

[Tabellion and Lamorlette 2004]: An approximate global illumination system for computer generated films

[Ward and Heckbert 1992]: Irradiance gradients



[Christensen et al. 2003]: Per H. Christensen, David M. Laur, Julian Fong, Wayne L. Wooten, Dana Batali, "Ray Differentials and Multiresolution Geometry Caching for Distribution Ray Tracing in Complex Scenes", pp. 543-552 in Computer Graphics Forum (Eurographics 2003 Conference Proceedings), Blackwell Publishers, September 2003

[Christensen 2008]: Per H. Christensen, "Point-Based Approximate Color Bleeding", Pixar Technical Memo #08-01, July 2008

[Křivánek et al. 2008]: Jaroslav Křivánek, Pascal Gautron, Greg Ward, Henrik Wann Jensen, Eric Tabellion, Per H. Christensen, "Practical Global Illumination With Irradiance Caching", ACM SIGGRAPH 2008 Class, Los Angeles, August 2008

[Mirtich 1998]: Brian Mirtich, "V-Clip: fast and robust polyhedral collision detection", ACM Transactions on Graphics (TOG), v.17 n.3, p.177-208, July 1998

[Sloan et al. 2005]: Peter-Pike Sloan, Ben Luna, John Snyder, "Local, deformable precomputed radiance transfer", ACM Transactions on Graphics (TOG), v.24 n.3, July 2005

[Tabellion and Lamorlette 2004]: Eric Tabellion, Arnauld Lamorlette, "An approximate global illumination system for computer generated films", ACM Transactions on Graphics (TOG), v.23 n.3, August 2004

[Ward and Heckbert 1992]: Greg Ward, Paul Heckbert, "Irradiance gradients", Third Eurographics Workshop on Rendering, 1992, pages 85--98.



I would like to acknowledge the people listed on this slide, who have contributed to valuable ideas or helped with the development of our GI systems. A large part of the credit also goes to the lighters and CG supervisors who have used and pushed the limits of our tools in production and with them produced the beautiful images we can find in these slides and in DreamWorks' movies.

Monsters vs. Aliens, Kung Fu Panda, Shrek, Shrek The Third, Madagascar, Madagascar: Escape To Africa®, Shrek 2, How To Train Your Dragon, Shrek Forever After TM & © 2010 DreamWorks Animation LLC. All Rights Reserved.