Nikolay Stefanov, PhD
Ubisoft Massive

GLOBAL ILLUMINATION IN GAMES

What is global illumination?

Cornell98

- Interaction between light and surfaces
- Adds effects such as soft contact shadows and colour bleeding
- Can be brute-forced using path tracing, but is very slow

Global illumination is simply the process by which light interacts with physical surfaces. Surfaces absorb, reflect or transmit light.

This here is a picture of the famous Cornell box scene, which is a standard environment for testing out global illumination algorithms. The only light source in this image is at the top of the image. Global illumination adds soft shadows and colour between between the cubes and the walls. There is also natural darkening along the edges of the walls, floor and ceiling.

One such algorithm is path tracing. A naive, brute force path trace essentially shoot millions and millions of rays or paths, recording every surface hit along the way. The algorithm stops tracing the path when the ray hits a light.
As you can imagine, this takes ages to produce a noise-free image.
Nonetheless, people have started using some variants of this algorithm for movie production (e.g. “Cloudy With a Chance of Meatballs”)

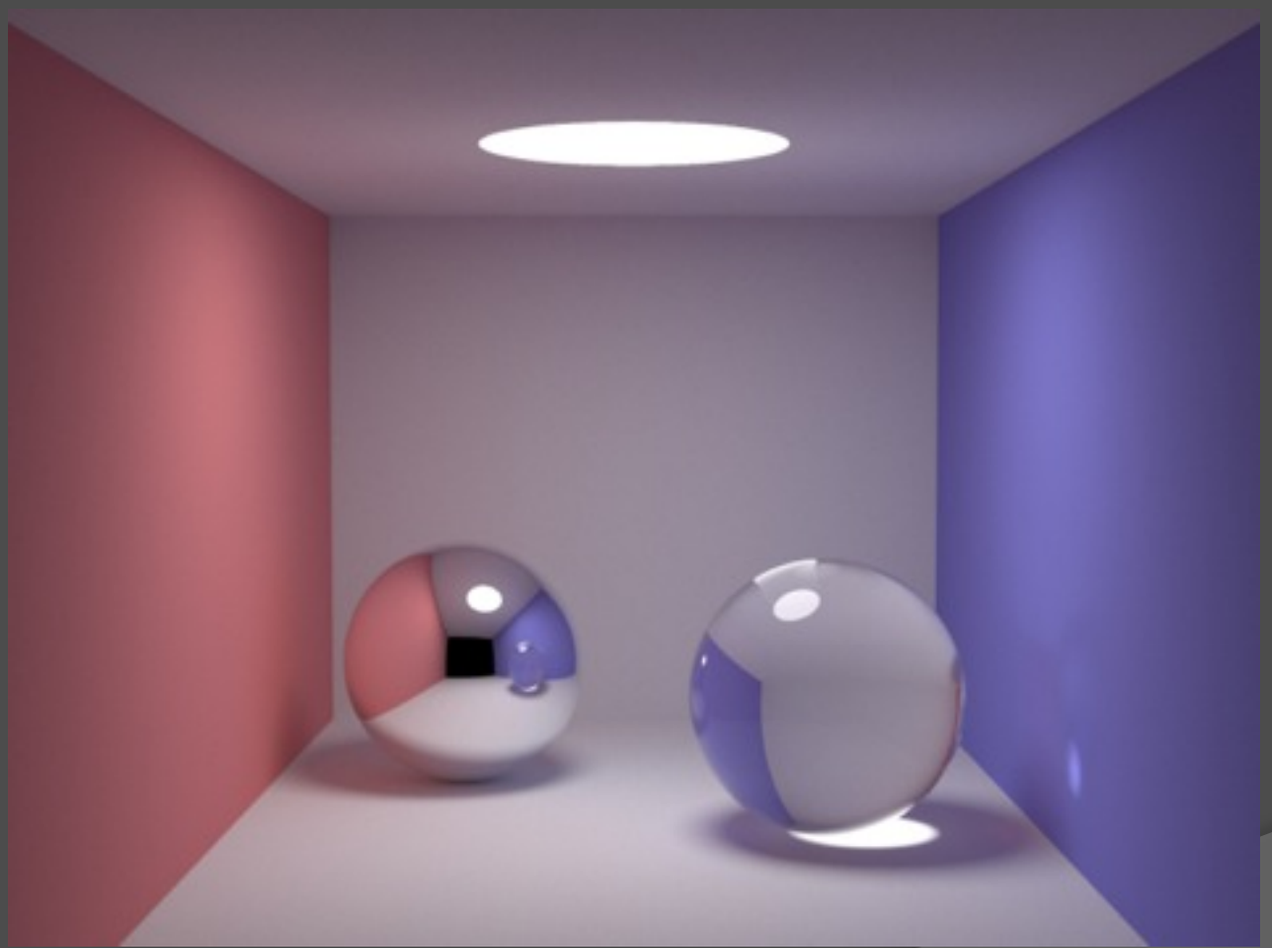
What is global illumination?

$$L_o(\mathbf{x}, \omega, \lambda, t) = L_e(\mathbf{x}, \omega, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega', \omega, \lambda, t) L_i(\mathbf{x}, \omega', \lambda, t)(-\omega' \cdot \mathbf{n}) d\omega'$$

- Scary looking equation!
- But can be done in 99 lines of C++

All of this makes for some scary looking maths, as illustrated here by the rendering equation from Kajiya. However that doesn't mean that the *code* is necessarily complicated.

GI in 99 lines of C++



Beason2010

Here is an image of Kevin Beason's smallpt, which is 99 lines of C++ code. Every pixel in the image uses 25K of samples (paths). This is quite slow and you have to let the program run for a very long time before the results become sufficiently noise-free.

GI in practice

- Don’t need a fully accurate solution
- Movies often use crude approximations
- For example, diffuse inter-reflection only, single light bounce, etc.

In practice, we often don’t need to calculate the whole shebang. Movies and games for the past couple of years have been able to get away with murder.

First off, the light is usually so quickly absorbed that we can just not bother with simulating anything after it bounces once from a material.

Humans are also notoriously bad at perceiving errors in images with global illumination. They will notice it’s not there, but will not be able to spot tiny errors in the solution.

GI in movies

Tabellion2004

This is an example image from the first movie to ever use global illumination – Shrek 2. You can see how the colour bounce adds a lot to the ambient term, and helps us to see the detail in areas which are in shadow.

The smart folks at Dreamworks only simulated one single colour bounce, and they restricted themselves to diffuse inter-reflection only. This helped speed up the rendering times tremendously and still achieved very impressive results.

What about games?

- It's the next "big thing"
- Everyone wants it
- Good lighting / mood sets your game apart
- Have to cheat even more blatantly

Adding global illumination to games is the next big thing in the industry. Remember how a couple of years ago everyone was raving about HDR and bloom, or if you're very old like me – coloured lights and lens flares? Well, it's the same with global illumination – everyone wants to have it.

Game developers this generation have realized that good lighting and mood can help set your game apart from the competition. Especially if you're releasing another "generic military shooter sequel".

Since we also have to maintain 30 frames per second, we have to cheat even more than movies. If you think about it, we have only 33ms to achieve a similar looking thing that games do in minutes on large render farms.

Been there, done that (in a way)

Conker: Live & Reloaded. (c) Rare, MGS

In a way, global illumination has been with us for some time already. This is an image of a game released by the company I was working before coming to Sweden. It's called Conker Live and Reloaded and came out for the original Xbox in 2005.

Conker used a hemisphere lighting model. Esentially, we interpolated between two colours – sky and ground. If the normal pointed upwards, we would choose the sky colour. If it pointed downwards, we would choose the ground colour.

The ground colour was sampled dynamically, by shooting a ray downwards and doing collision detection with the ground. The colour was encoded in the vertices, and we could interpolate it at the point where the ray hit the triangles. On the left you can how the colour is this sort of blue-gray, while on the right it's orange.

This is about the crudest appoximation of colour bleeding that you can get.

Agenda

- Lightmaps
- Spherical harmonics
- Light probes
- Radiance volumes

So let’s discuss a few ways in which developers have been adding GI in their games.
We’ll start with the so called “static solutions” – this is where we precompute everything, and try to minimize the work we have to do at runtime. These include lightmaps, which should be very familiar to most of you, and light probes.
Then we’ll move on to dynamic solutions like radiance volumes.

Lightmaps

Mirror’s Edge. (c) DICE, EA

Let’s start with lightmaps. This is based on some ideas that hark back to the Quake days. Pretty much standard in all current major game engines.

The big players in the field are actually a Swedish company called Illuminate Labs. They provide middleware for lightmapping to pretty much everyone. This image is from a Swedish game called Mirror’s Edge which made heavy use of lightmaps and Illuminate Labs’ Beast. Their art style is chosen so that it further showcases this – very simple, often single colour / white materials. You can see here the colour bleeding between the different surfaces.

Lightmaps

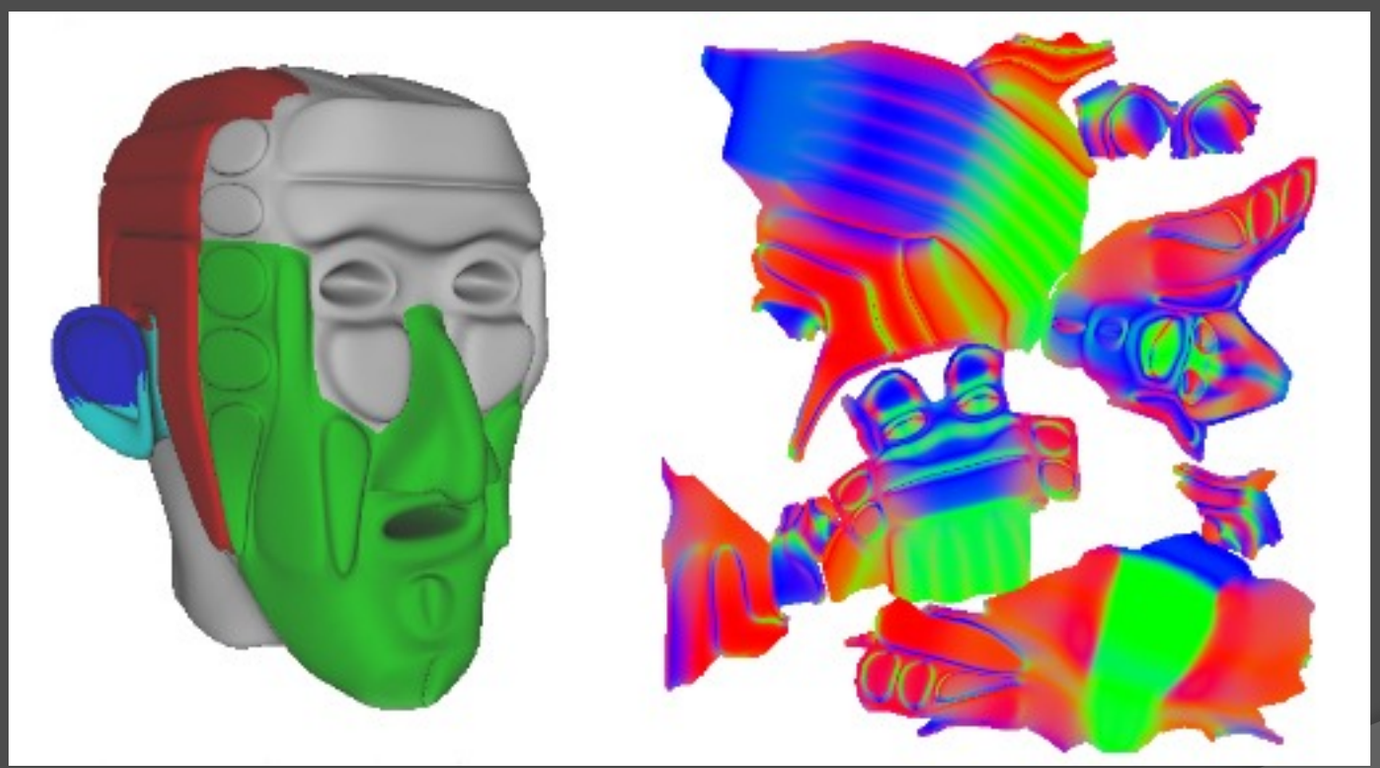
- Precompute lighting over the entire surface of the model
- Store it inside a texture
- Be careful with memory!

What you do is you precompute the incoming lighting over the entire surface of the model. You then store it inside a texture.

Obviously to do this you need a unique mapping, so no two points on the surface of the model point to the same texel in the lightmap.

You need to be careful, since if you want to cover the entire world with lightmaps it will get really memory heavy. Your artists need to balance the size of the textures, so that they manage to stay within the memory limits, but also we don’t want the lightmaps to be really blocky.

UV atlas



MSDN 2010

Here is an image that demonstrates the unique mapping required for lightmaps. The model on the left is decomposed into a bunch of “charts”. On the right you see the lightmap texture and how the charts are laid out in UV space. In this case the image shows the worldspace normal of the mesh. But we can put any data we like in it.

Global illumination and

- It’s precomputed!
- Can use well-known offline rendering algorithms
- Things get interesting when deciding how to store the lighting
- How to combine with the material, especially the normal map?

For example GI. Since it’s all precomputed, we can use well known offline rendering techniques to do the precomputation - for example path tracing or photon mapping. Most of the cost at runtime will be quite small – one or two texture fetches and some ALU operations.

This is all pretty boring stuff. How you store the lighting is where things get interesting. The most important thing is to make sure you have a way to combine the precomputed lighting with the normal map. This will give your offline solution a much higher apparent resolution, since the normal map can add a lot of detail to the lightmaps with very fine-grained variations of the surface normal.

Lightmaps at Massive

We also use lightmaps and global illumination at Massive. Here is a screenshot of one location without lightmaps. It looks pretty awful, since the lighting is completely flat – the sun can’t reach this particular location.

Lightmaps at Massive

Adding a lightmap makes a dramatic difference. You can see the objects pop, there are now shadows, and you can see the darkening in the corners.

Illumination

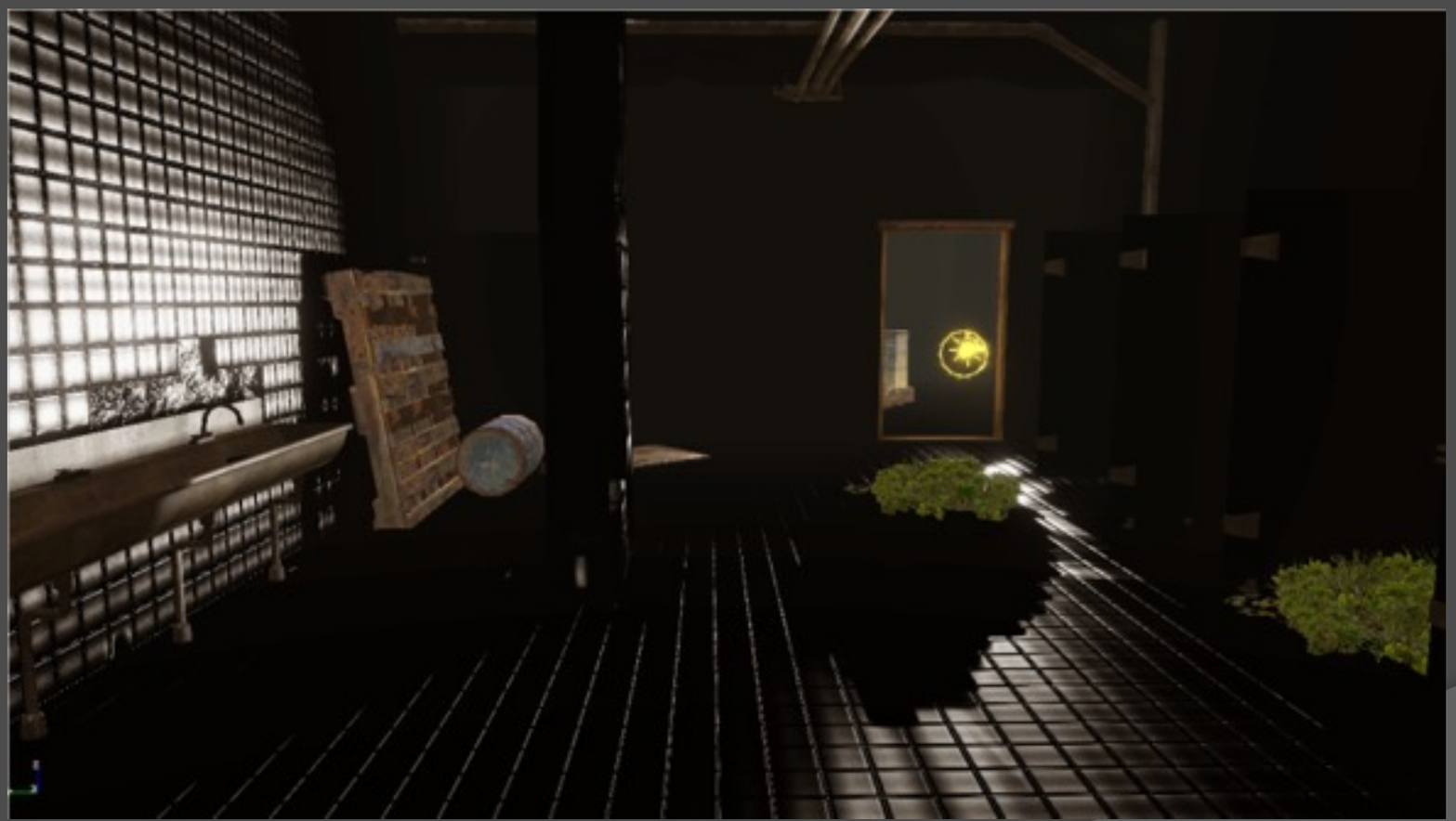
The lightmap can be broken down into a few components
The first one is the illumination. This is just the light that comes from every light source in the scene. We can also add a single light bounce (not shown here).
We combine the lightmaps with the normal maps by storing a dominant light direction in a second texture. Basically this is a vector that tells us which direction the most light is coming from. In the shader, we can compare this vector with the normal, and take away from the light accordingly.

Ambient occlusion

The second component of the lightmap is ambient occlusion. We use the standard way to generate ambient occlusion – from every point in the lightmap, we shoot a bunch of rays and figure out whether they hit something or not. The averaged and normalized result is a value from 0 to 1 and tells such how occluded the point is. To save processing time, we actually shoot a relatively small number of rays, and then blur the results.

In the shader this simply acts as a multiplier for the final shader value. It is not really correct, as it needs to be applied to the ambient term only, but our artists like it the wrong way around.

Specular



Since we know the light direction at every pixel, we can also compute a fake specular term. This helps add to the illusion that the lighting is actually somewhat dynamic.

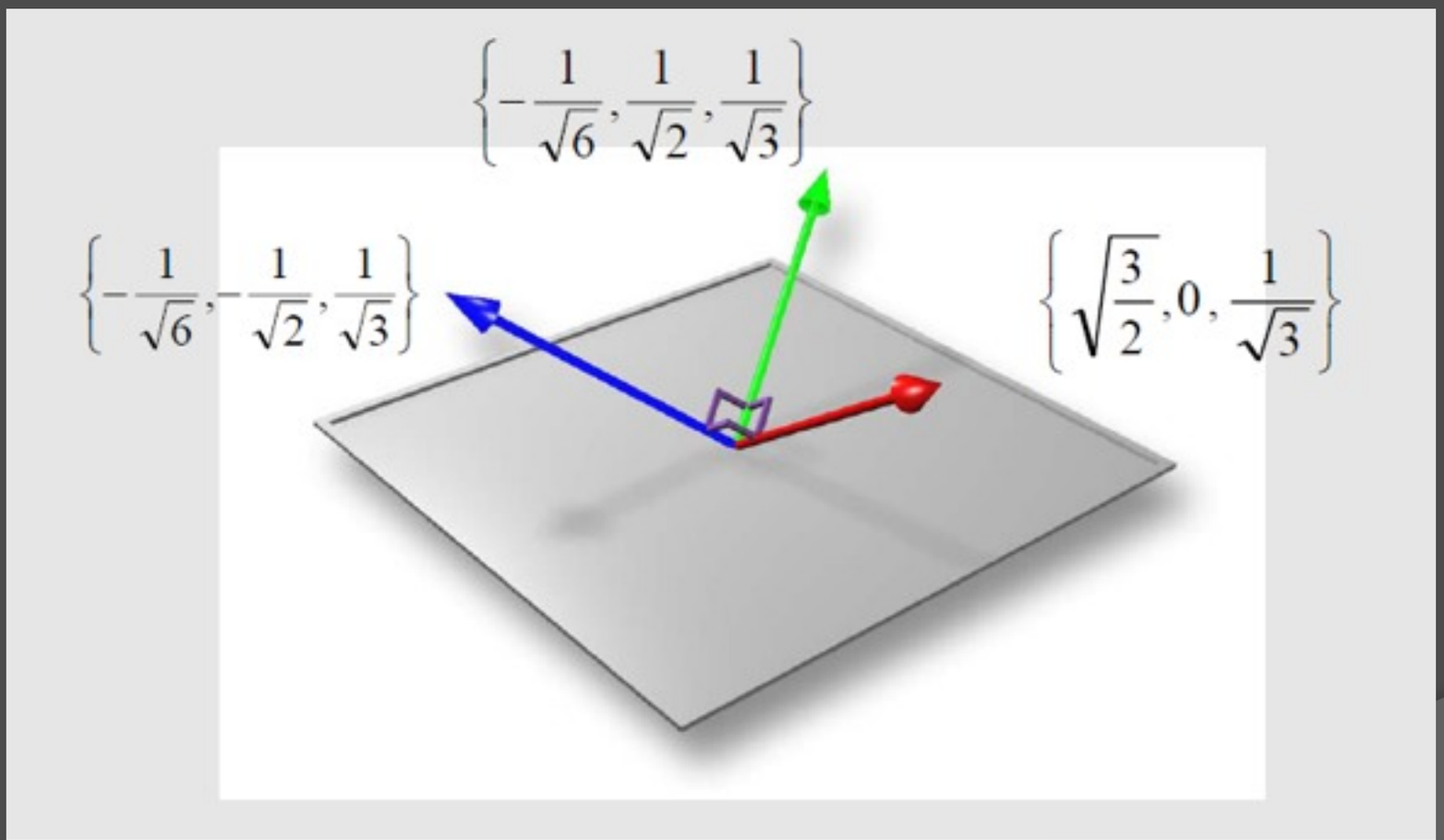
Half-Life 2

Mitchell2006

Of course, everyone has different ways of storing global illumination. I'll give you two of the most important examples here.

One of the first games that managed to incorporate global illumination and lightmaps was Half-Life 2. It was also one of the first ones to successfully combine the normalmaps with GI.

Half-Life 2 basis



Mitchell2006

Half-Life 2 had a clever trick to combine the lightmaps with the normal maps.

The lighting was stored in tangent space, and projected onto three different basis vectors. Essentially it computed how strong (and what colour) the lighting was along each of the three vectors.

When rendering, the normal from the normal map can be used to combine the three different lighting values. The downside is of course that you now need three lightmap where before you only needed one. On the upside though you can now get away with a smaller texture resolution, since a lot of detail is implicitly added by the normal maps.

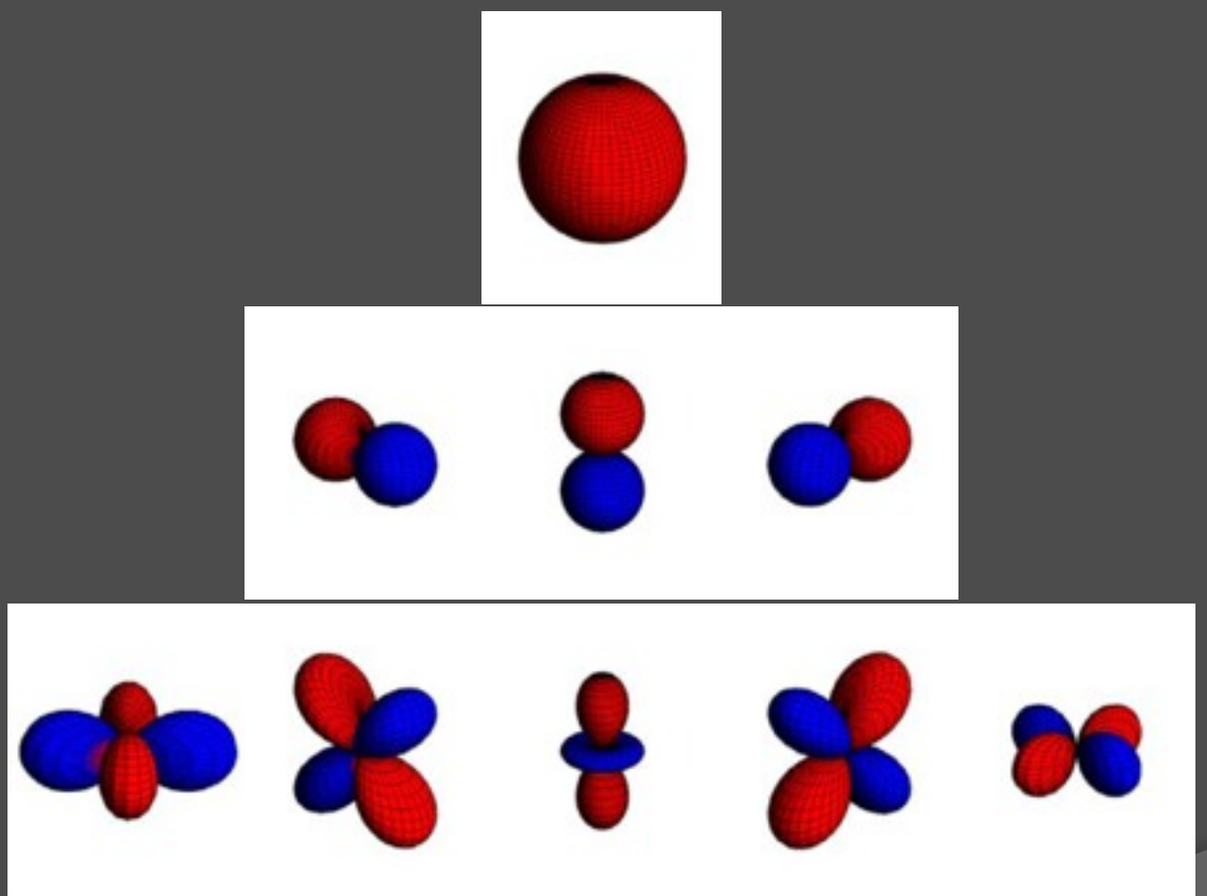
Halo 3

Chen2008

Another game that made extensive use of lightmaps and global illumination is Halo 3. In this case, the lightmaps were represented as spherical harmonics. Spherical harmonics means that the texture memory footprint was even heavier than in Half-Life 2, so they had to use clever compression techniques. You can read more details in their GDC presentations.

This brings us nicely onto our next topic.

Spherical harmonics



Sloan2008

SH are a really great tool for computer graphics. I'll give you a quick introduction here, but you should really go off and do some research on your own.

Spherical harmonics are a set of basis functions, which allows us to represent a signal over the sphere. The functions are organized in bands, and every band has 2 * band + 1 functions in it.

You can think

Spherical harmonics

$$f_l^m = \int f(s) y_l^m(s)\, ds$$

Any signal / function over the sphere can be reconstructed with an expansion using spherical harmonics.

s means the entire sphere
f(s) is the function that is defined on the sphere.
y(l, m) is the spherical harmonic

We assume that we've got a way to analytically evaluate f(s).

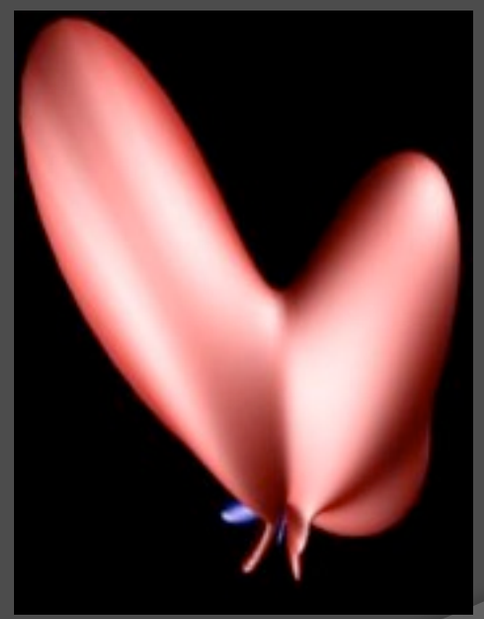
First, we integrate the function against the spherical harmonics and obtain a bunch of coefficients f(l,m). You can do that analytically or using Monte Carlo integration. In Monte Carlo integration, we take lots and lots of random directions. For each direction, we evaluate f in that direction, and also y(l,m). We add the results together, normalize and we end up with f(l,m)

Spherical harmonics

$$\tilde{f}(s) = \sum_{l=0}^{n} \sum_{m=-l}^{l} f_l^m y_l^m(s)$$

When we have the f(l, m), we can reconstruct the original function (evaluate it in any direction) by multiplying together the coefficients and the spherical harmonics function.

Spherical Harmonics



Ramamoorthi2001

What does it mean in practice?

It means we can use any signal over the sphere, in this case, a cubemap. This obviously doesn't have an analytic form, but we can still represent it as spherical harmonics, by integrating as shown previously.

In fact, the resulting representation is 97% accurate if we're interested in only having a diffuse component.

You can also think of it as a way to dramatically compress cubemaps in fact.

Light probes

- Lightmaps are great for static geometry
- What about dynamic (moving) objects?
- Precompute lighting inside a volume of space, rather than a texture

Let’s see how we can use spherical harmonics in another way.
Lightmaps are really great for static geometry, since they allow very high quality lighting.
We have a problem however if we want our dynamic objects (stuff that can be moved around) to have a similar quality of lighting.
A common solution to this is to precompute lighting inside a volume, rather than just on the surface of the geometry. This is referred to as light probes.

Light probes in Killzone 2

Leeuw2009

Killzone 2 precomputes the lighting at a number of locations, represented here as spheres. This is offline done during lightmapping. The probes are stored as spherical harmonics.

Light probes in Killzone 2

Leeuw2009

Every dynamic object, such as this soldier, will find its closest 4 probes and interpolate between them. The result is then “decompressed” back into a sphere mapped texture. The shader then samples this sphere map based on the normal. Think of this as a way way more advanced version of the hemisphere lighting.

More?

- Lightmaps and light probes are static
- Takes a long time to compute
- No time of day
- What if we wanted something completely dynamic?

Lightmaps and light probes are great, but they're a static solution. It takes a long time to pre-compute them – sometimes hours on a complex level. The process of mapping the textures is also quite laborious for artists. Since the lighting they capture is absolutely static, there is no way to have features such as time of day change.

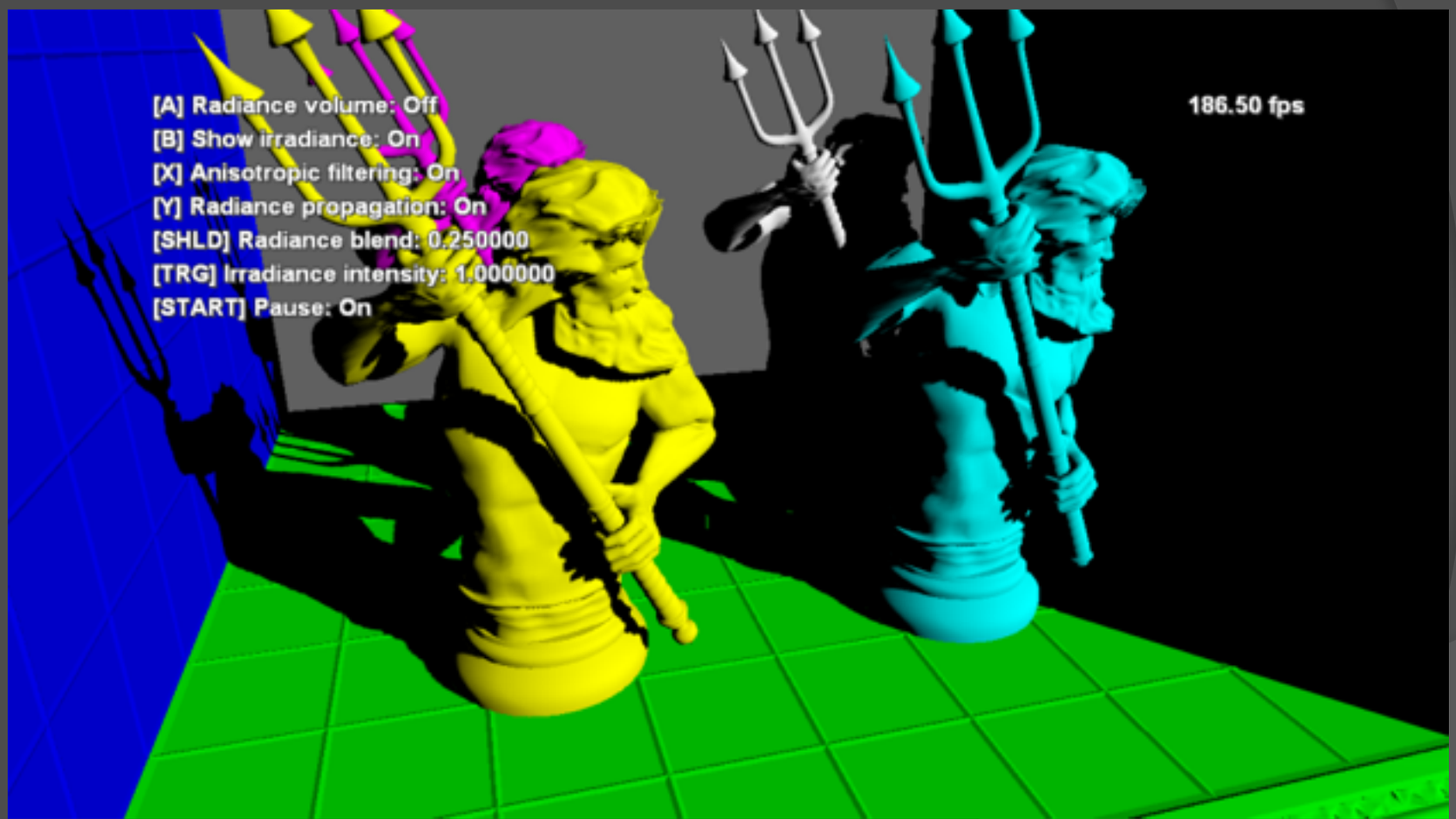
The question is, can we come up with something that is completely dynamic. No precomputation, everything is done at runtime?

Light propagation volumes

Kaplanyan2009

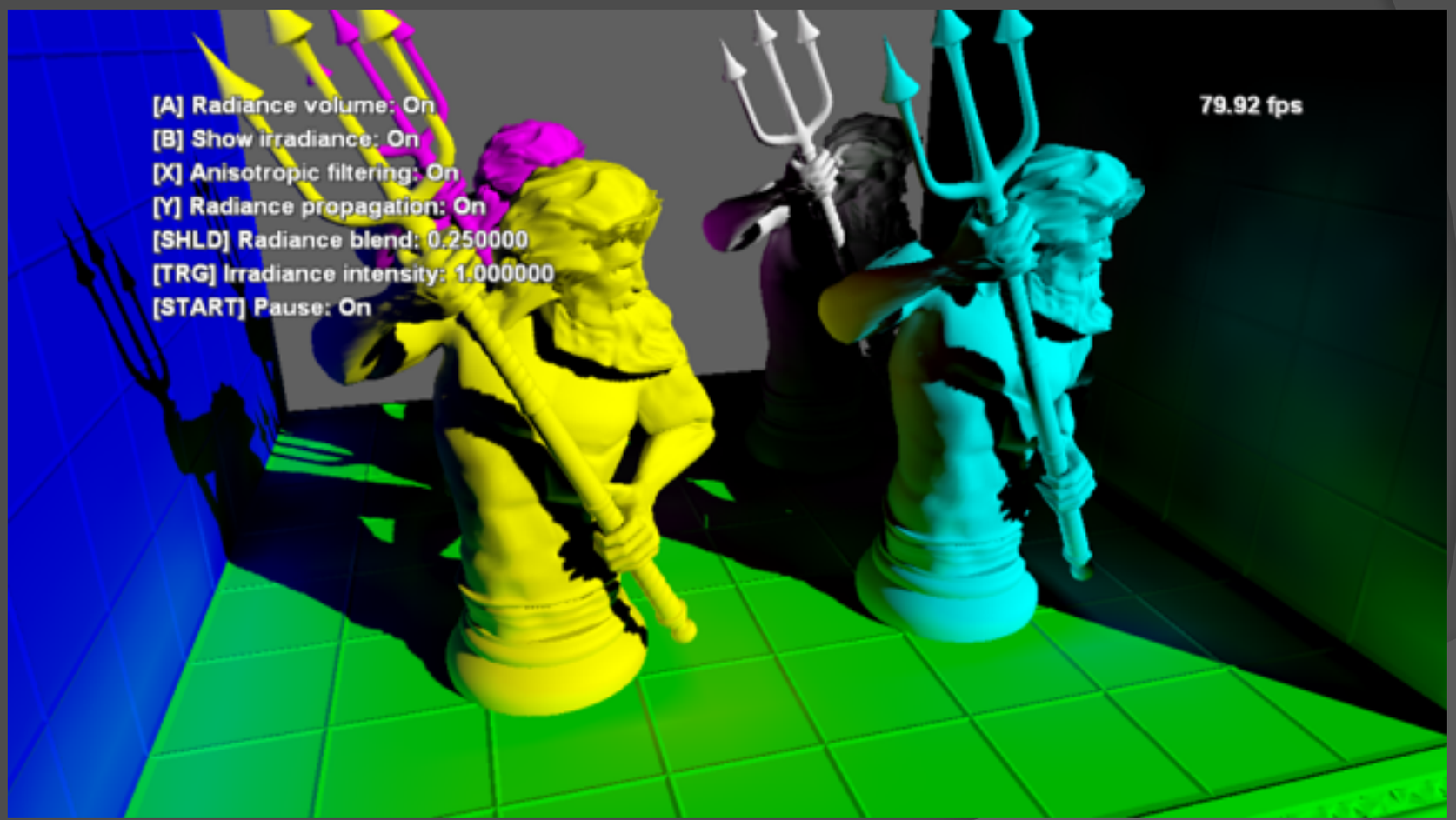
Turns out you can, in a way. Here are some images from the as yet unreleased Crysis 2 that show off a technique called Light propagation volumes. I'll walk you through my own implementation of it.

Direct lighting only



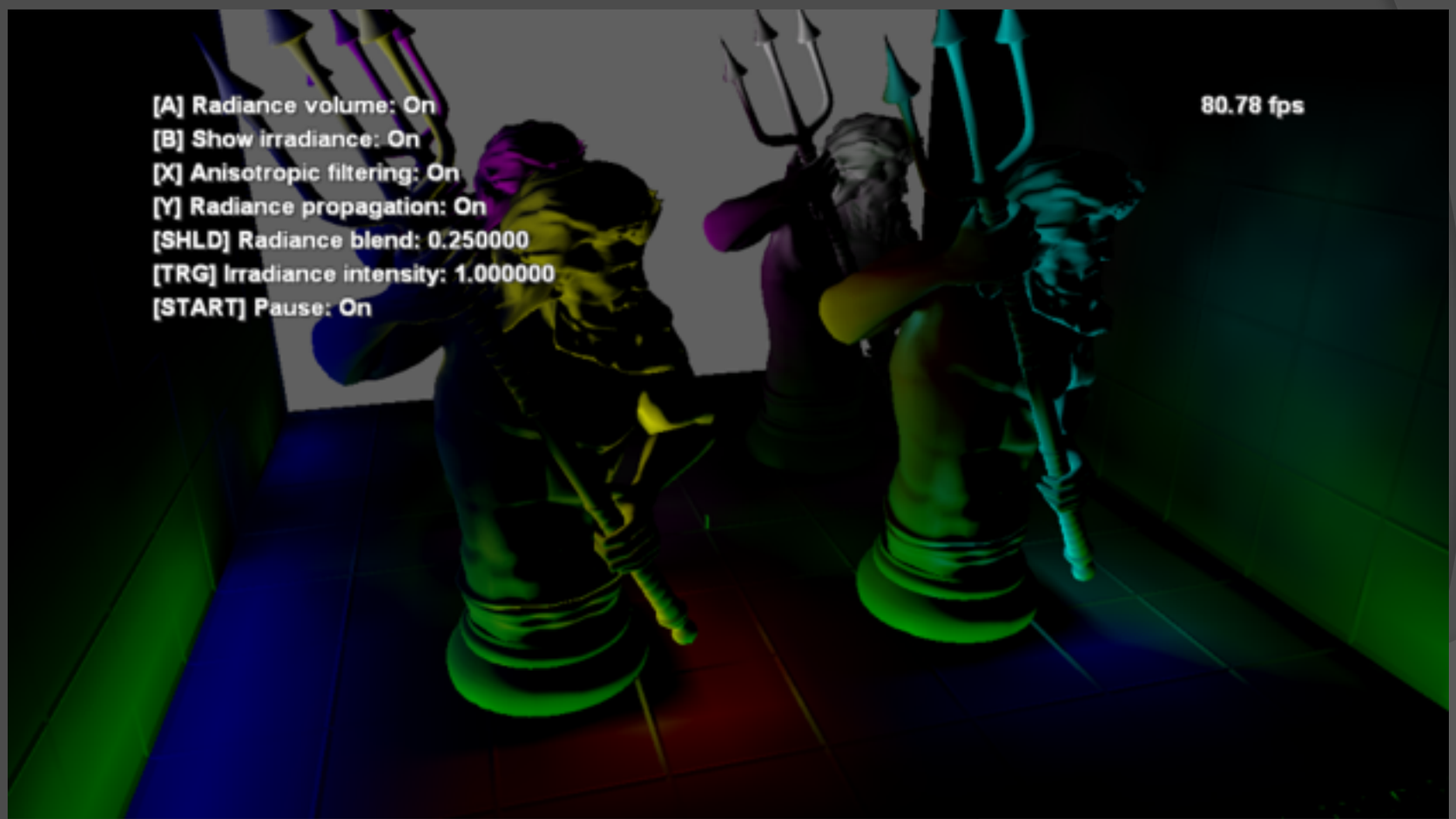
Here is a screenshot from my demo application. It shows direct lighting only, without any ambient term. As a result, the areas in shadow are completely black.

With indirect lighting



Here's what it looks like when you add indirect lighting using the light propagation volume. You can light bouncing off the lit surfaces which illuminates the areas in shadow. You can for example see the reflection on the side wall, and between the different models.

Indirect lighting only



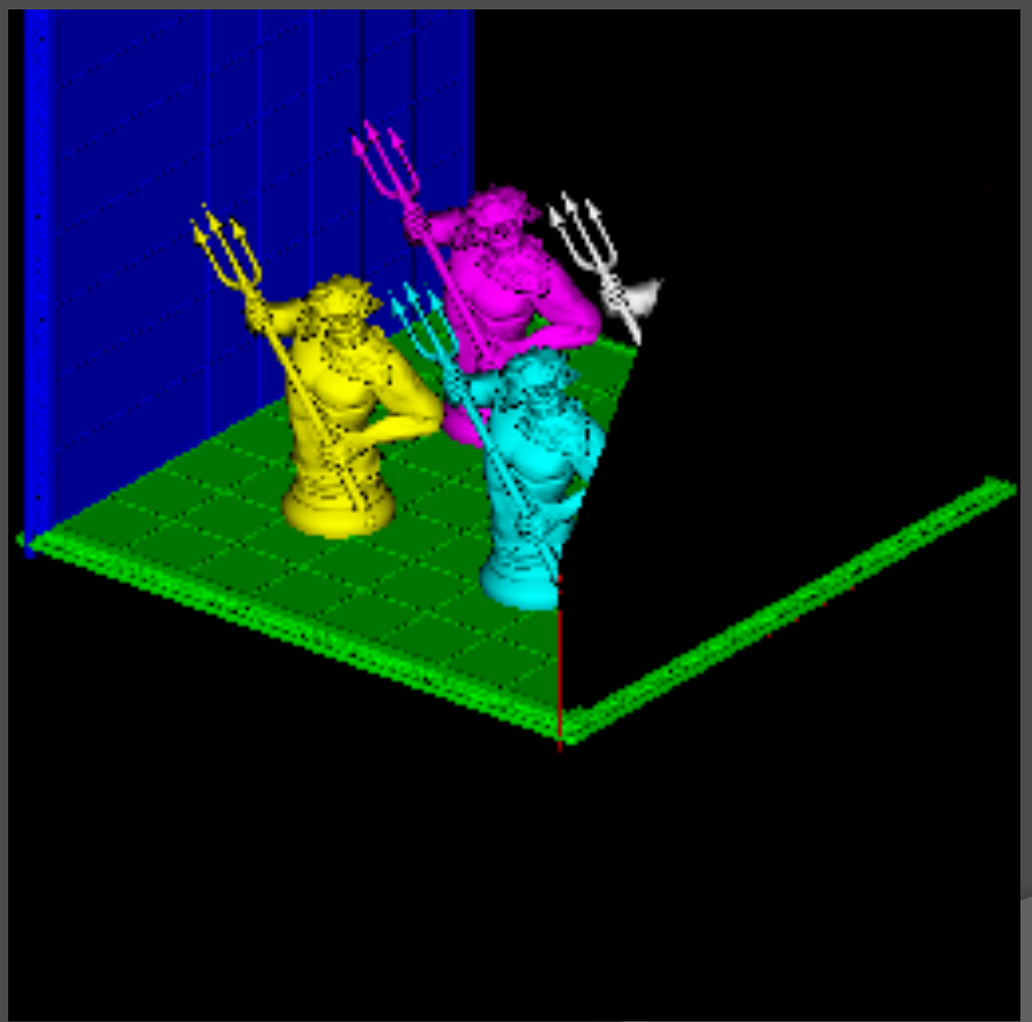
Here is a screen shot showing just the indirect lighting, so you can better see what it adds to the scene.

How does it work?

- Combination of two basic concepts (and spherical harmonics again!)
- The first step is to render a “reflective shadow map”

The algorithm is based on a combination of two relatively simple concepts.
Reflective shadow maps, and irradiance volumes.

Reflective shadow map



Reflective shadow maps are like shadow maps – they represent the scene viewed from the light source. Here's the RSM for the scene I showed you earlier. Unlike shadow maps, they also store the colour of the surface and its normal. Essentially they provide us with information about all the pixels which are directly lit from that particular light source.

How does it work?

- Combination of two basic concepts
- The first step is to render a “reflective shadow map”
- The second step is to downsample the RSM into a “radiance volume”

Now that we have information about which parts of the surface are directly lit, we need to spread that information to other parts of the surfaces. This is referred to as “radiance injection” in the original paper

Irradiance volume

Greger96

What's a radiance volume? Let's start by looking at what an "irradiance volume" is first.

Irradiance volumes are essentially a bunch of light probes on a regular grid. Every probe catches the light that illuminates that particular point in space, or in fancy terms the "irradiance".

Radiance volumes on the other hand encode how much light is emitter from the location of every probe – the opposite of "irradiance" is "radiance".

Radiance injection

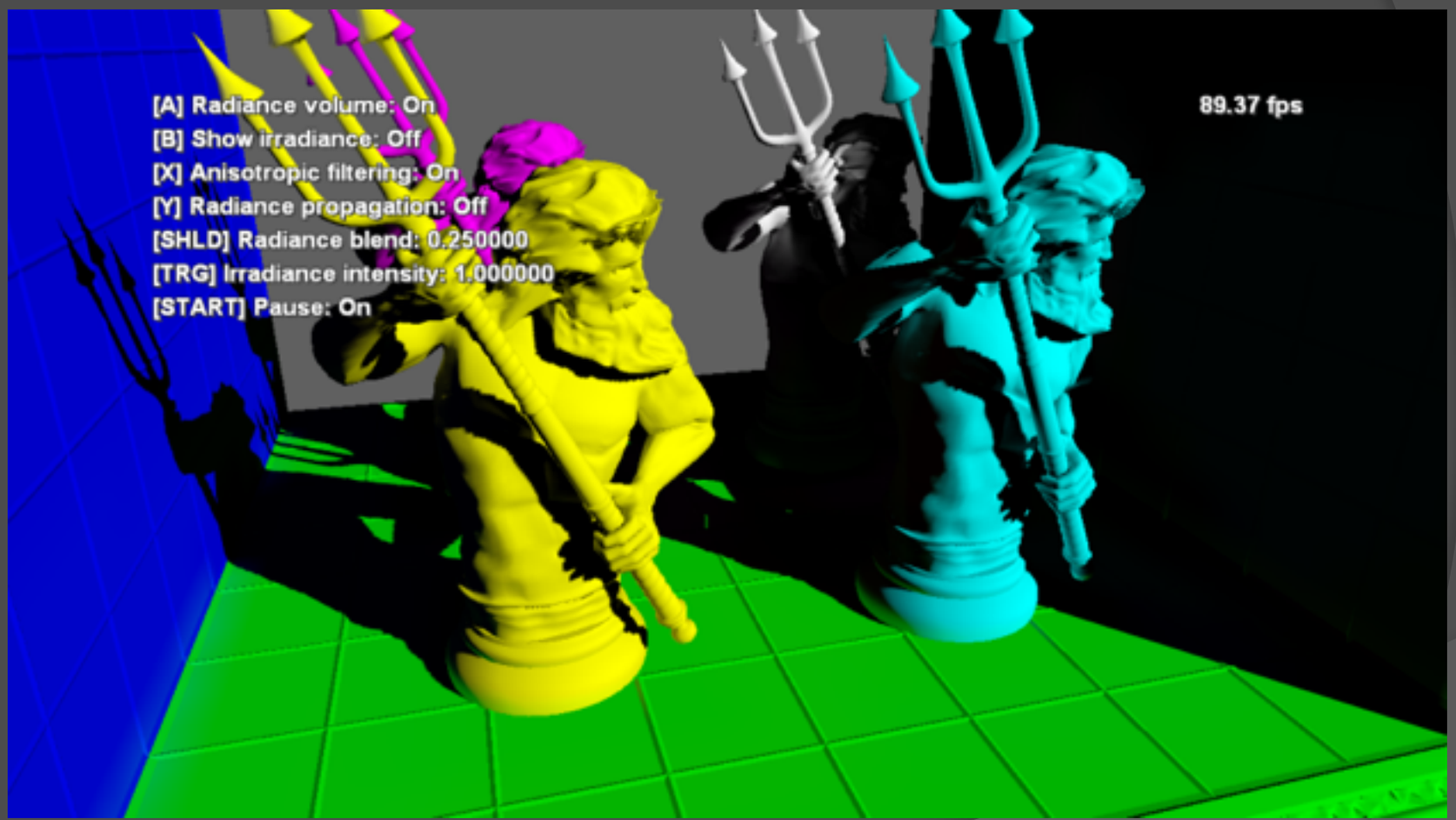
- Use a small volume texture for the radiance volume (32x32x32 is enough)
- Treat every pixel in the RSM as a very small spot light source
- Encode the light it emits into spherical harmonics
- Insert at the appropriate place in the volume texture

How does it work?

- Combination of two basic concepts
- The first step is to render a “reflective shadow map”
- The second step is to downsample the RSM into an “radiance volume”
- When rendering, simply look up into the volume to find out incoming lighting

Then when we render the models, we simply use the position of every pixel in world space to lookup in the radiance volume, and figure out how much light is emitted from that particular location.

Not enough!



However, this is not enough. Since the RSM is very small, most of the volume texture will be completely empty. You can see in this image what we've got so far. There's a bit of colour bleeding, but only between surfaces which are really close together.

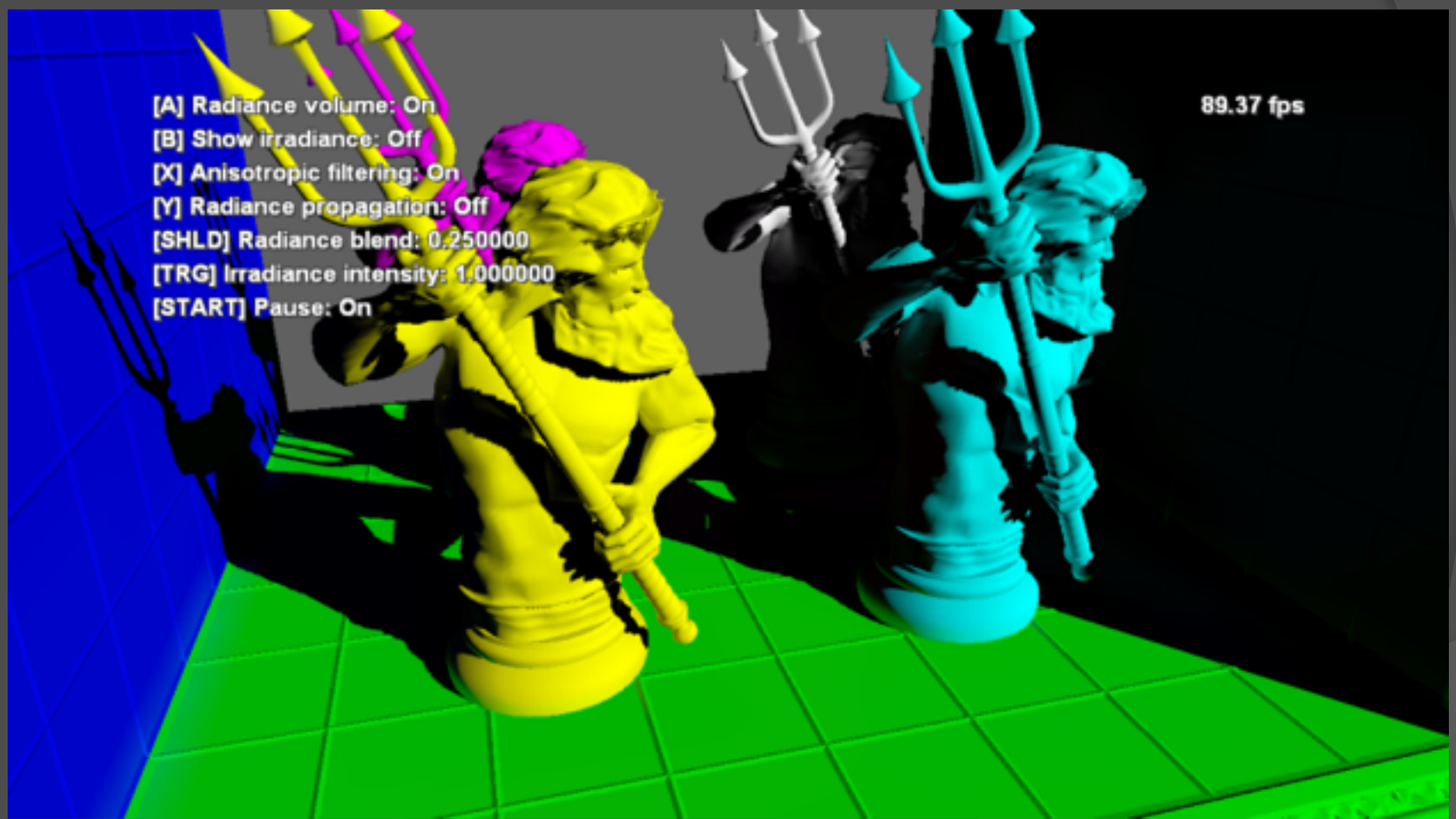
Not enough!

Here is the indirect lighting only. You can see a bit of colour bleeding, but nowhere near as much as we'd like.

Radiance propagation

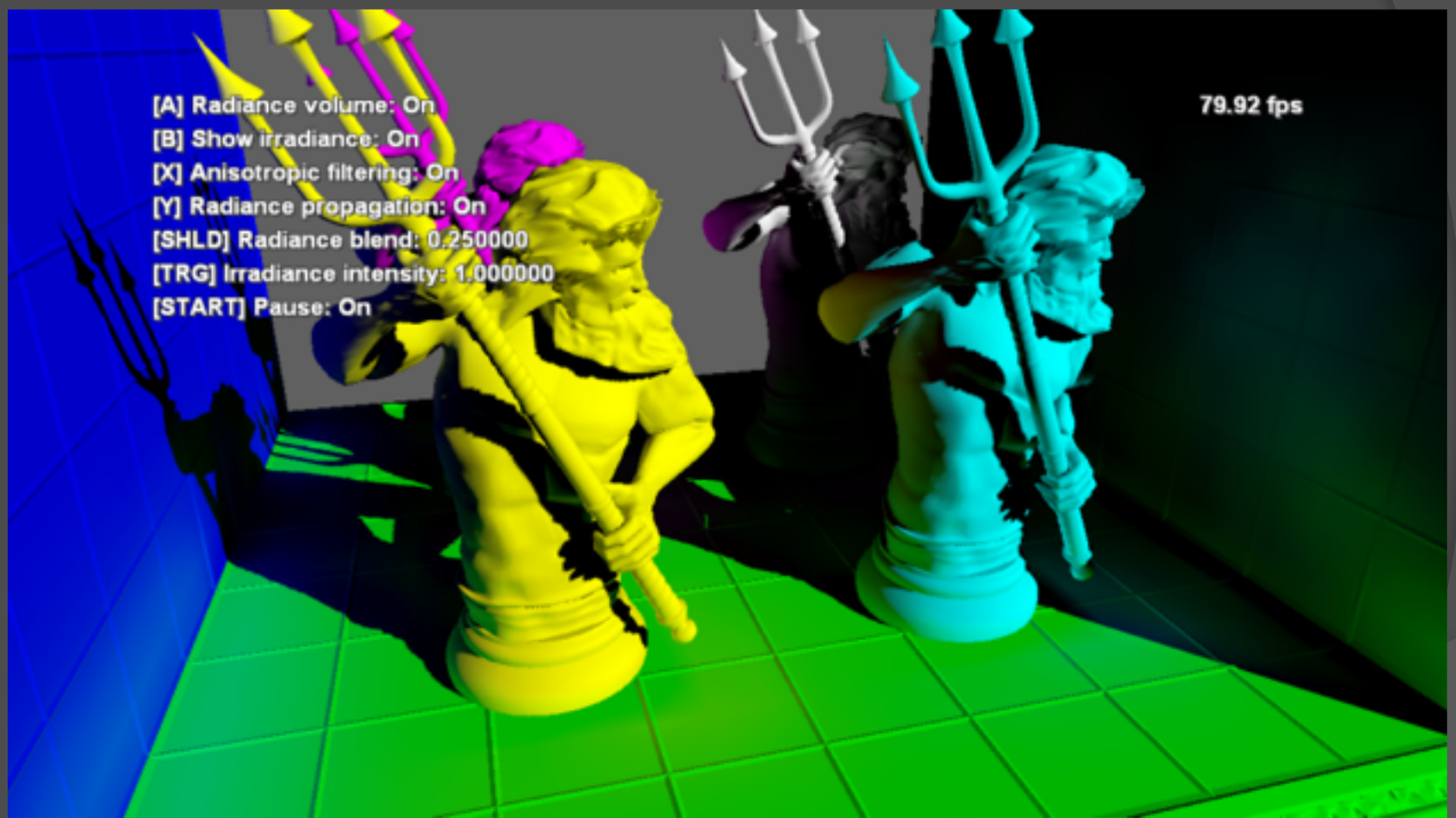
- In order to get better results, need to propagate light between neighbouring cells in the volume
- Every cell figures out how much each of its neighbours emits in its direction
- Think “clever blurring” of the volume texture

Without propagation



Here is the image without propagation again.

With propagation



And here it is with propagation enabled.

Caveats

- Lots of flickering if not careful
- Resolution might be an issue
- No indirect shadows!

This algorithm, although great, has its fair share of caveats.

First off, it is very prone to flickering, since the RSM and the volume texture are very low resolution – there is simply not enough data. In order to combat that, we use a special trick where we downsample the RSM and average neighbouring pixels. This way we ensure that pixels will fall consistently into the same place inside the volume texture.

We also have to use temporal filtering. Essentially we keep a copy of the previous frame volume, and blend it with the current frame.

Resolution is another problem. The volume is very small, so if we stretch it over the entire world we'll get really crappy results. Therefore, we use more volume organized in a cascade that follows the camera.

Finally, the propagation step does not take into account visibility between the different cells. You can use a simple voxel representation of the scene to figure it out, but in general it will be too expensive to bother.

Questions?

- nikolays@massive.se
- http://www.massive.se

References

- [Cornell98] http://www.graphics.cornell.edu/online/box/
- [Beason2010] http://www.kevinbeason.com/smallpt/
- [Tabellion2004] Tabellion et al. "An Approximate Global Illumination System for Computer Generated Films"
- [MSDN2010] http://msdn.microsoft.com/en-us/library/bb206321(VS.85).aspx
- [Mitchell2006] http://www.valvesoftware.com/publications/2006/SIGGRAPH06_Course_ShadingInValvesSourceEngine.pdf
- [Chen2008] http://www.bungie.net/images/Inside/publications/presentations/lighting_material.zip
- [Sloan2008] http://www.ppsloan.org/publications/StupidSH36.pdf
- [Ramamoorthi2001] http://www.cs.berkeley.edu/~ravir/papers/envmap/
- [Leeuw2009] http://sijm.ca/2009/wp-content/uploads/2009/12/michiel-van-der-leeuw.pdf
- [Kaplanyan2009] http://www6.incrysis.com/Light_Propagation_Volumes.pdf
- [Greger96] http://www.gene.greger-weltin.org/professional/publications/thesis.pdf