Rock-Solid Shading

Image Stability Without Sacrificing Detail

Dan Baker and Stephen Hill

SIGGRAPH2012

Advances in Real-Time Rendering in 3D Graphics and Games

Part 1

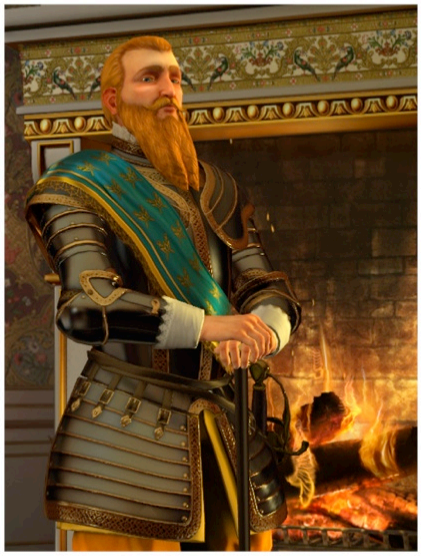
Why Naïve Techniques are Naïve

(Dan Baker)

Goals: What makes a shader good?

SIGGRAPH2012

- Make stuff look good
  - Stable
  - Clean, no aliasing
  - Expressiveness in material types
  - Simple, intuitive models
- Main tools for this: BRDFs
  - Blinn-Phong, Banks, Ashikhmin-Shirley
- Most materials, stylized or not, can be expressed by a simple BRDF



There is no easy way to objectify what looks good, but I find it useful to lay our goals down as to what we consider “good” means, to make it more objective.

Blinn-Phong, properly factored and filtered, turns out to be quite expressive.

Basics: Blinn-Phong

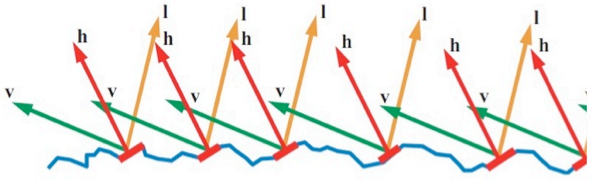
- Diffuse part isn't bad
  - Concentrate on specular part
- Review: $L_s = P * pow(dot(N, H) , P)$
  - N = Normal
  - H = Half Angle
  - P = Power
- This is a normalized variant Blinn-Phong
  - If you aren't normalizing, shame on you!

The normalization factor on Blinn-Phong is vital for a number of reasons. It prevents the total illumination from the function from getting darker, and though this could be compensated by an artist, what it means is that an artist shouldn't have to modify the Ks value when they are changing the shininess of an object. It also automatically makes a scene 'HDR', because high powers will return very powerful highlights.

This normalization factor will also prove helpful later on when we switch to a different BRDF, since this one will be normalized.

Microfacet theory

- Blinn-Phong is no hack
- Microfacets
  - Made of tiny mirrors
  - Light bounces back when mirrors line up
  - A distribution function of mirrors
  - The closer the half angle is to the normal, the greater percent mirrors line up with viewer, the brighter the reflection



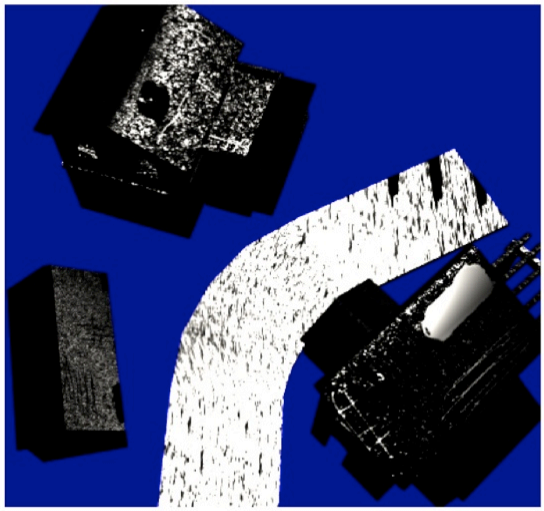
There is more to reflectance than this, specifically when we talk about materials which have some level of light scattering on the surface. But in terms of many hard materials, the microfacet model is a good one.

We know how to implement Blinn Phong, right?

```
float BlinnPhongSpec(float2 TexCoord, float3 LightDir)
{
   float3 N = NormalMap.Sample(SS_DEFAULT, TexCoord);
   float  SpecPow = PowerMap.Sample(SS_DEFAULT, TexCoord);

   float3 Half = normalize(N + LightDir);

   return SpecPow * pow(dot(Half, N), SpecPow);
}
```

Problem #1: Objects are too shiny

- Steps
  - Create a simple image with a rough bump map
  - Render image at 4096x4096
  - Downsample to 512x512
  - Render same image at 512
  - Compare results
- BRDF has changed
  - Warning: Object may appear shinier than it is



These screenshots are from a quick and dirty demo. The right way to do this is to use a texture parameterization and render to it using Texture Space Lighting. This will create very stable, nice looking results.

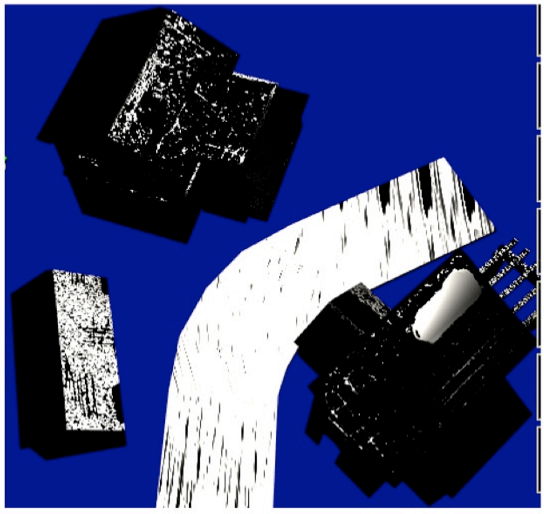
For my slides on this, check out:

http://www.slideshare.net/mobius.cn/advanced-lighting-techniques-dan-baker-meltdown-2005

It's a bit old now, but still good info.

Problem #1: Objects are too shiny

- Steps
  - Create a simple image with a rough bump map
  - Render image at 4096x4096
  - Downsample to 512x512
  - Render same image at 512
  - Compare results
- BRDF has changed
  - Warning: Object may appear shinier than it is



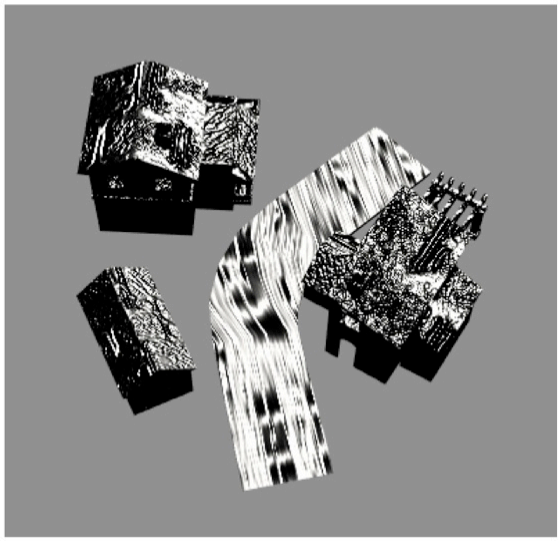
Real World Example: Granite

- Polished Granite Counter Tops
  - Typical for kitchens, bathrooms
  - No laminate material, just a sealant
  - Specularity since surface is smooth
- Leathered Granite
  - A rough stone look
  - Shockingly accurate term, since leather has a dull specular highlight
  - Same material, just rougher!

As of now, my kitchen remodel still Isn't done.

Problem #2: Aliasing

- Sampling issues can cause normal to suddenly catch a highlight
  - Sparkles
  - Causes artifacts in HDR bloom pass
  - Highlight may be missed altogether
- Prevents using high powers
  - Will often see spec done with an env map to get sharp highlights



The inability to use high powers was a huge problem for us for Sid Meier's Civilization 5. I think people typically use reflectance maps to mitigate this, but this didn't work well for our scale. In general, it makes Blinn-Phong far less useful because the range of settings is very narrow.

Why don't we see problems in offline rendering?

- Sampling rate is often locked – e.g. REYES
  - Even wrong, samples are same frame to frame
  - Removes much of temporal aliasing from the equation
- Everything is over-sampled
  - A hundred samples per pixel isn't uncommon
  - Sampled at infinity most problems disappear
- Solved? No, but brute force has mitigated it

As I am not an expert in offline rendering, I don't know the answers to these questions to my own satisfaction. I believe that brute force has helped them, but an elegant solution would be better.

Conclusion

- Didn't meet goals
  - Unstable: resolution greatly affects large scale effects
  - Aliases: temporally and spatially
  - Unexpressive: unable to use a wide gamut of power
- How do we implement Blinn-Phong correctly?
  - Could do a texture based lighting approach – a la REYES
  - Could we find a similar BRDF that actually behaves itself?

Part 2

Mechanics of LEAN Mapping

(Dan Baker)

LEAN Mapping

- Linear Efficient Anti-aliased Normal Mapping, I3D 2010
  - Shipped in Sid Meier's Civilization V for water
  - Deploying for all asset production going forward
- Advantages:
  - Temporally stable
  - Resolution stable
  - Can use high powers (e.g. 10,000+)
  - Blinn-Phong content can be easily converted
  - Automatically anisotropic

A copy of our paper can be found at:

http://www.csee.umbc.edu/~olano/papers/

Or in the I3D proceedings:

Marc Olano and Dan Baker, "LEAN Mapping", I3D 2010: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games.

This talk is a little vague due to time constraints, but the I3D paper describes the specifics.

LEAN Mapping Review

Given a normal from a bump map:

N = (N.x, N.y, N.z)

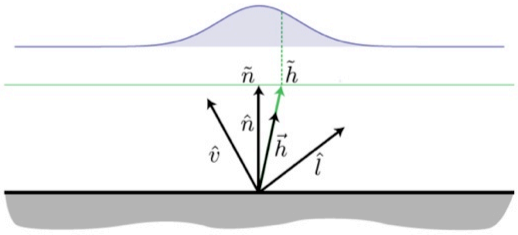
Then we create another map M:

M = (B.x^2, B.x*B.y, B.y^2)

Where:

B = (N.x/N.z, N.y/N.z)

Not redundant data! We need the linear filtered version of these terms!



$$\frac{1}{\sqrt{|\Sigma|}} e^{-\frac{1}{2}\left((\bar{h}_n - \bar{b}_n)^T \Sigma^{-1} (\bar{h}_n - \bar{b}_n)\right)}$$

$$\Sigma = \begin{bmatrix} M.x - B.x * B.x & M.z - B.x * B.y \\ M.z - B.x * B.y & M.y - B.y * B.y \end{bmatrix}$$

Storage

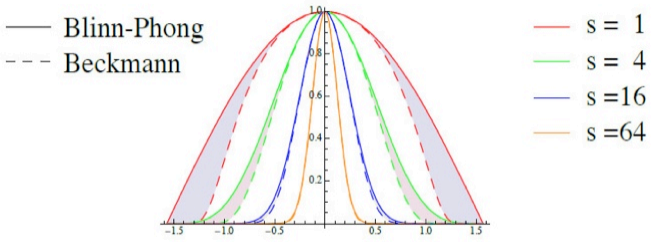
- 5 Channels
  - X, Y offset from center bump, can be 8 bit
  - X^2, Y^2, X*Y, needs to be 16 bit if you want nice high specular powers (and you do!)
- Compression might be possible, but since linear filtering is used, we can't sacrifice this
- Middle ground solutions to be covered next

Initializing Content from Blinn Phong

- Base specular power **s** added as 1/s along the X^2 and Y^2 terms
- So M map (X^2, Y^2, XY) becomes (X^2 + 1/s, Y^2 + 1/s, XY)
- Storing the inverse power means we need 16 bit precision for powers > 256
- **Observation:** even if we are using Blinn-Phong, storing power as 1/s will cause MIP filter to operate correctly

How well does it approximate Blinn-Phong?

- For low powers (e.g. < 16), LEAN mapping responds differently than Blinn-Phong
- May need to retune some content



We had some issues with low power materials being different – in some of our shaders we actually LERP to Blinn-Phong for very low powers since the aliasing/reflectance issues aren't as great for low powers.

CLEAN mapping

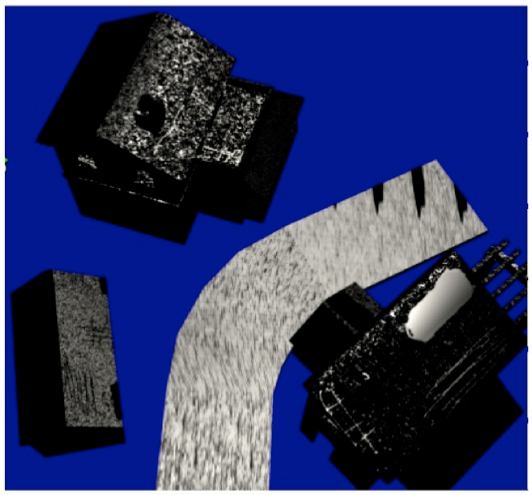
- 5 Values can be expensive: alternative, lose anisotropy

$$\sum = \begin{bmatrix} M - B.x * B.x & M - B.x * B.y \\ M - B.x * B.y & M - B.y * B.y \end{bmatrix}$$

- Store 3 values, X, Y, (X^2 + Y^2)/2
  - Only (X^2 + Y^2)/2 needs be stored at high precision

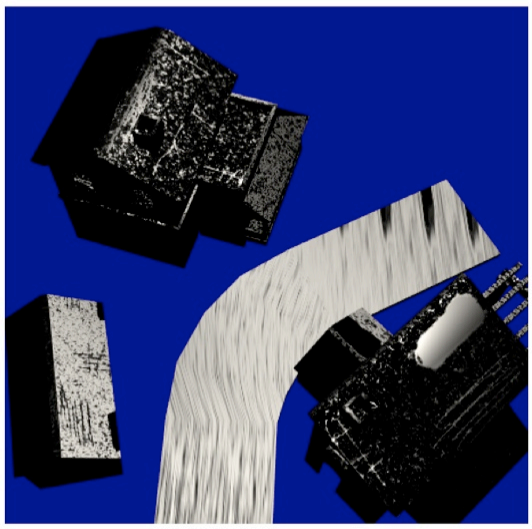
Problem #1: Better

- Highlights are more stable under minimization filter



Problem #1: Better

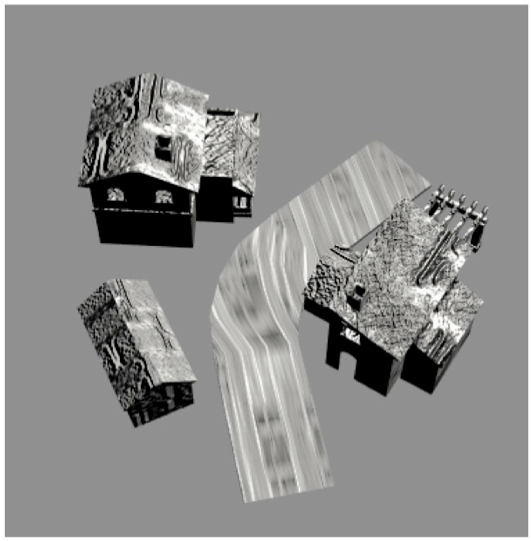
- Highlights are more stable under minimization filter



The fact that there are still some illumination differences is likely due to the fact we shift the distribution on the tangent plane rather than reorient the surface.

Problem #2: Solved

- No aliasing
- In fact, seems to alias less than the real world



Conclusion

- Meets goals
  - Stable: resolution rendered at does not affect large scale effects
  - Anti-aliased: linear hardware filters work appropriately
  - Expressive: can use large powers, and anisotropy supported
- Still some issues:
  - Divergence with Blinn-Phong at low powers
  - Storage space requirements are higher

We have experimented with different compression schemes, and have found that splitting the high and low precision into different channels of a BC5 texture actually works reasonably well. Artifacts from incorrect linear filtering weren't terrible.

Part 3

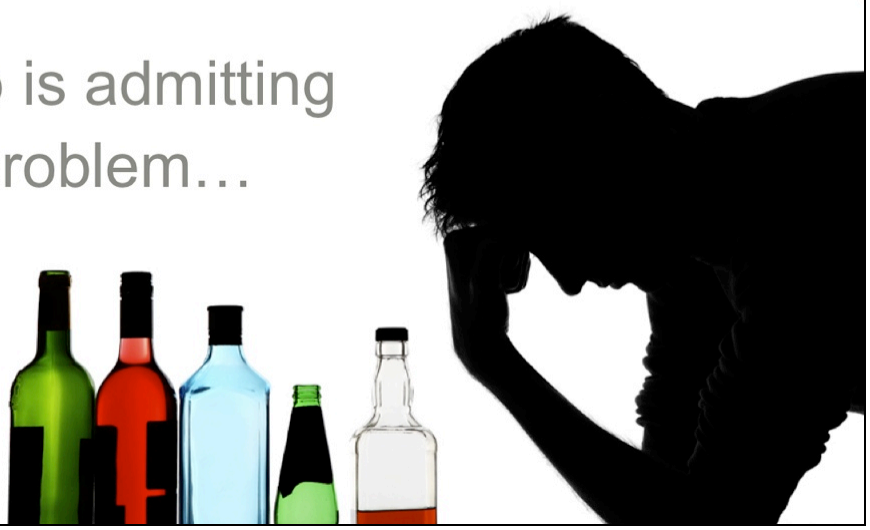
Shader Aliasing Anonymous

(Stephen Hill)

Shader Aliasing Anonymous

SIGGRAPH2012

The first step is admitting you have a problem…



A year ago I had a wakeup call: I'd been throwing my life away, focusing on the wrong things. What's the point in supporting many dynamic lights, for instance, if the shading is *all wrong*?

This was a 'self-intervention'; the art team, on the other hand, had just come to accept temporal aliasing and lack of appearance preservation.

The first rule is to admit that you have a problem. The second rule is: don't go pointing this out to artists before you have an action plan! I started showing them the many ways in which things were broken and they hated me for it. Pride in their work turned to disgust. You can't 'unsee' this stuff!

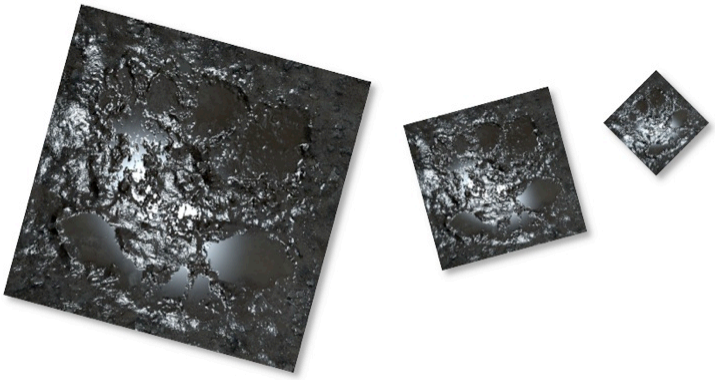
So, I went looking for solutions…

Options: Texture-Space Shading

SIGGRAPH2012

- Key idea: **MIP the lighting!**
- See: Advanced Lighting Tech [Baker05]

- Gold standard
- It'll cost you!
- Virtual texture cache?



The first of these is texture-space shading (TSS). It's a simple idea: light at the base mip texture resolution and generate mipmaps (on the fly).

This is problematic in two respects:

1) You need a unique parameterisation (no UV overlap)
2) Shading is likely to be prohibitively expensive (and variable), since small on-screen objects are still shaded at a much higher rate.

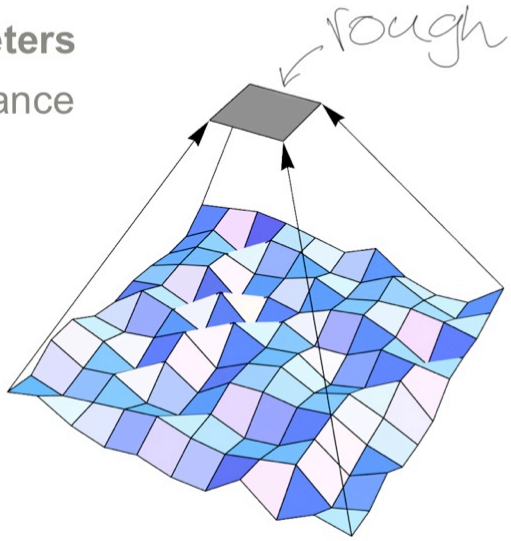
The second issue can be overcome by combining TSS with another technique.

The reason I'm mentioning TSS is that it's a useful ground-truth method (like super sampling, as Dan showed) to compare other methods against.

Options: Fitting

SIGGRAPH2012

- Key idea: **Find best-fit parameters**
- E.g. normal, roughness, reflectance
- See:
  - BFGS fitting [Baker05]
  - SpecVar Maps [Conran05]
  - Frequency Domain Normal Map Filtering [Han07]
- Slow, fragile, discontinuous



Fitting is one technique that could be combined with TSS.

The idea is to pre-calculate adjusted surface/material properties that best approximate the average lighting response of the base textures (albedo, specular and normal maps), for each MIP level.

The fitting process can be fiddly to get right. For instance, you need to ensure coherent results between texels, so that hardware texture filtering can be used.

The bigger problem is that the process can be slow, particularly if you're trying to solve for several parameters. In practice, the average normal that you get from regular mipmap generation works well, so fitting can be focused on other properties, such as roughness/ glossiness, which has a big impact on appearance when using energy-conserving specular. Still, anything that takes many seconds/minutes for a large texture will inhibit artist iteration.

Options: Direct Variance Estimation

- Key idea: **Estimate variance -> new roughness**
- See “Mipmapping Normal Maps” [Toksvig04]

$$\sigma^2 = \frac{1 - |N_a|}{|N_a|}$$

Fortunately, there’s a much more direct approach (avoiding fitting entirely) if you’re just concerned with adjusting the specular power.

In 2004, Michael Toksvig came up with a supremely elegant method that does this based on the length of the average filtered normal (Na), that we might get when sampling the normal map in a shader.

The length provides an estimate of the variance of the original normals…

Options: Direct Variance Estimation

- Key idea: **Estimate variance -> new roughness**

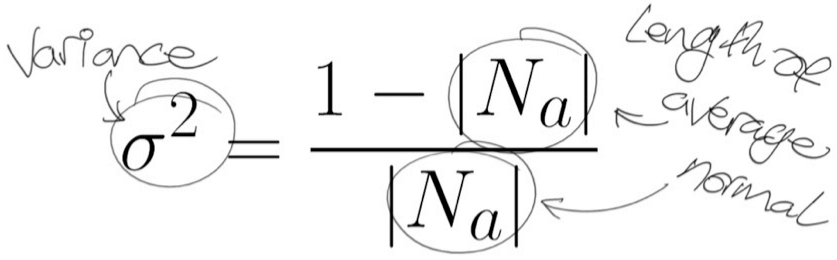
$$\sigma^2 = \frac{1 - |N_a|}{|N_a|}$$

Length of average normal

Here we're plugging the length into this simple equation…

Options: Direct Variance Estimation

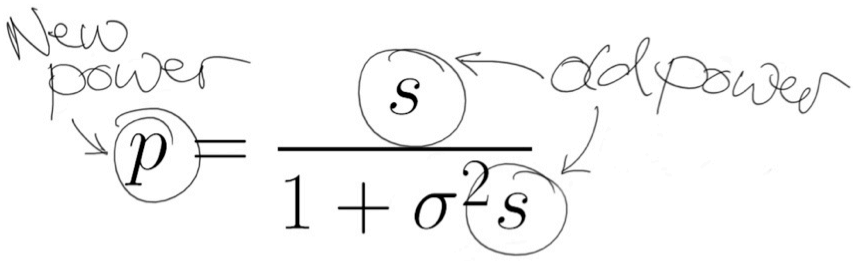
- Key idea: **Estimate variance -> new roughness**



…and out pops the variance estimate.

Options: Direct Variance Estimation

- Key idea: **Estimate variance -> new roughness**



Using this second equation, we can then adjust our old specular power, s, based on the variance.

This gives us a new specular power, p, that we shade with instead.

Options: Direct Variance Estimation

SIGGRAPH2012

- Key idea: **Estimate variance -> new roughness**

On the left: the original specular result. Highlights shimmer a lot under object, camera or light motion.

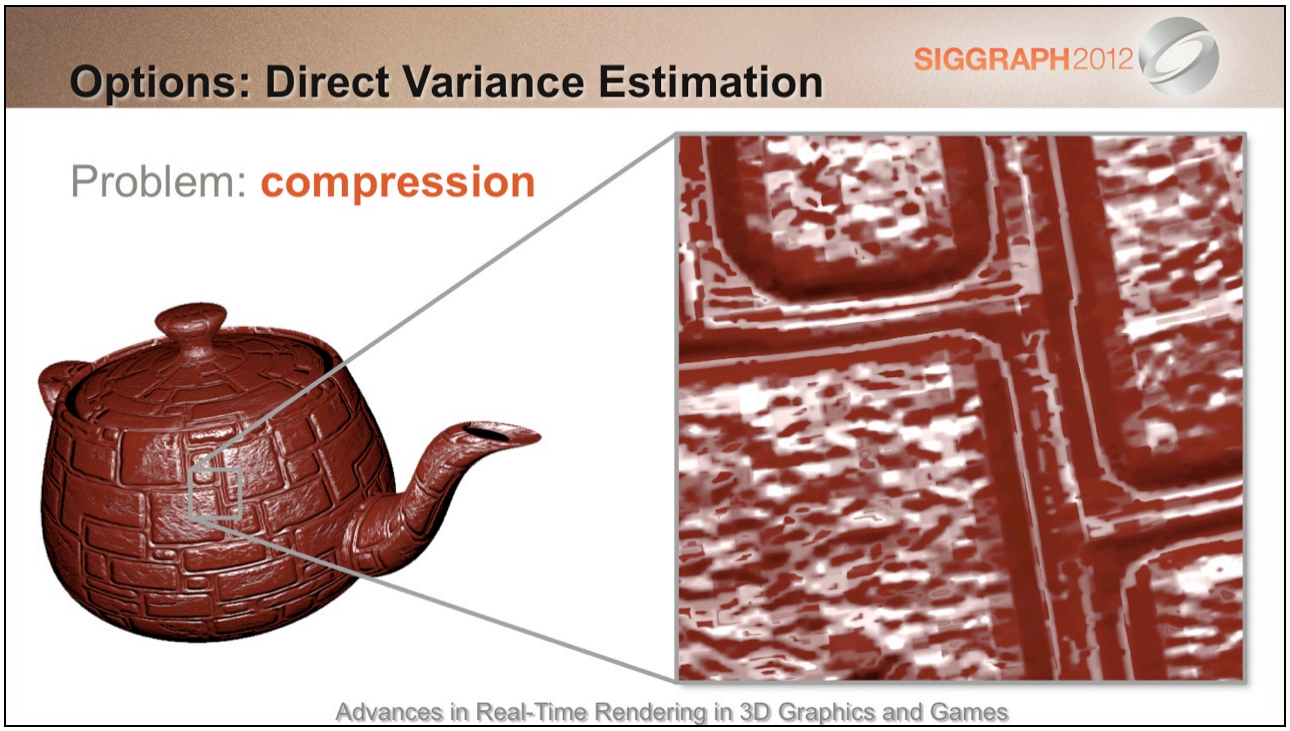
In the middle: the adjusted, anti-aliased result. There's a lot less shimmering and the teapot looks similar at all scales.

On the right: a visualisation of gloss adjustment (Toksvig scale factor). Flat areas are light, rough areas are dark. This makes intuitive sense.

Here's a demo with a similar setup:

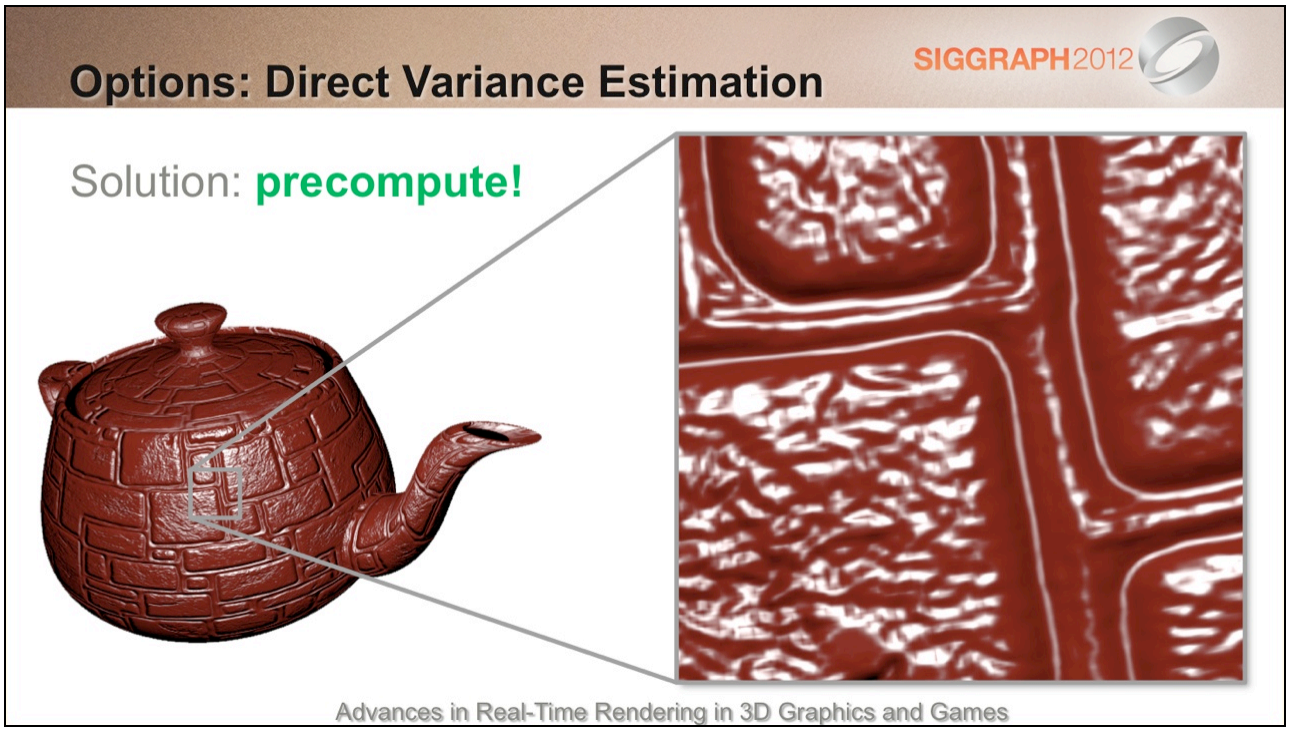
http://selfshadow.com/sandbox/gloss.html

In *theory*, these calculations can be performed in the shader itself…



However, there's a major practical issue: DXT compressed normals don't place nice with variance estimation.

With two-component normal map formats, we can't even estimate variance (normal length is assumed to be 1).



Fortunately, the problems go away if we precompute on texture import, prior to compression.

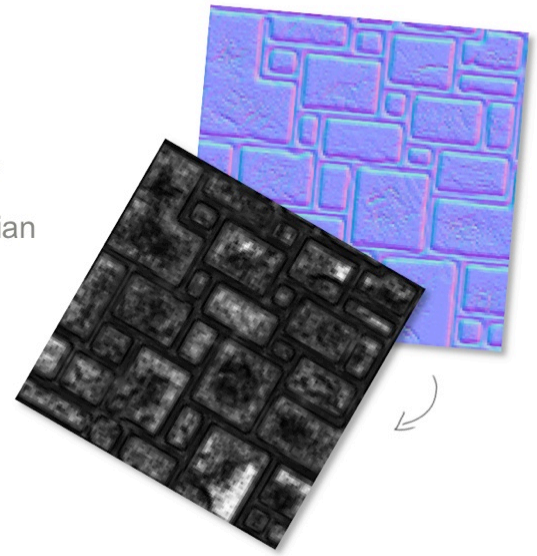
How do we do this?

Baking Variance

SIGGRAPH2012

For each MIP:

- Apply small filter for bilinear
  - E.g. Tent [Lazarov11] or Gaussian
- Calculate variance
- Store result:
  - Directly, or adjust gloss map
  - Allow edits?



Simple: for each MIP level of the normal map, compute the variance at each texel and store the result in a texture.

Note: it's a good idea to apply a small (3x3) filter here to simulate hardware texture filtering that will happen at runtime. This further reduces aliasing and leads to smoother mip transitions (particularly from the base mip level).

Instead of storing variance directly, another option is to adjust the (artist-painted) gloss map. This has the advantage that there's no extra shading cost at run time – everything just works automatically. However, it does tie the two textures together, which could be an issue if you regularly mix and match to save time/memory.

It's perhaps tempting to allow artists edit the results, but this has the danger of reintroducing aliasing and/or breaking the appearance in the distance. So far we haven't found any need for this.

Games are already shipping with this or very similar implementations. See [Lazarov11], [McAuley12].

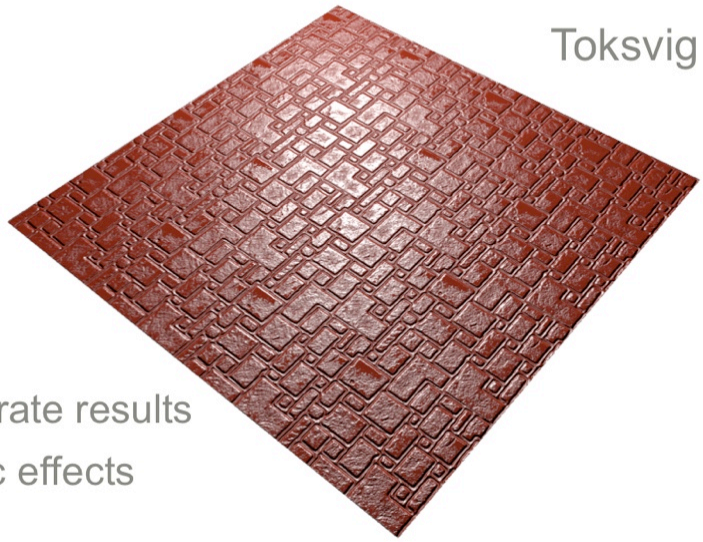
Adjusted Gloss

Versatile… specular AA ‘everywhere’!

- **Dynamic reflections**:
  - 1. Generate MIPs
  - 2. MIP-biased lookup
  - Or DX11: variable Gaussian? Image-Space Gathering?
- **Voxel Cone Tracing** [Crassin11]
- **Reflection billboards** [Mittring11]
- **Reflection occlusion?**

Once you have variance-adjusted glossiness, it pays off everywhere it’s used!

Options: Prefiltering (LEAN)



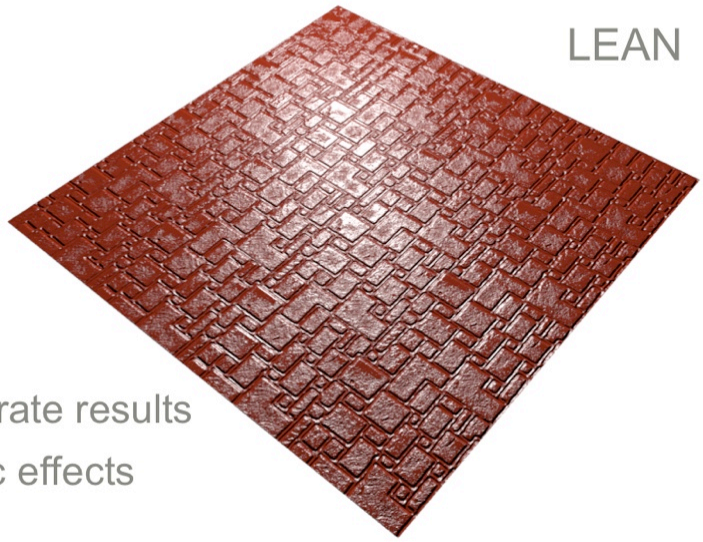
Good:

- More accurate results
- Anisotropic effects

However, although this method can work well, we can do better in some cases…

Options: Prefiltering (LEAN)

SIGGRAPH2012



Good:

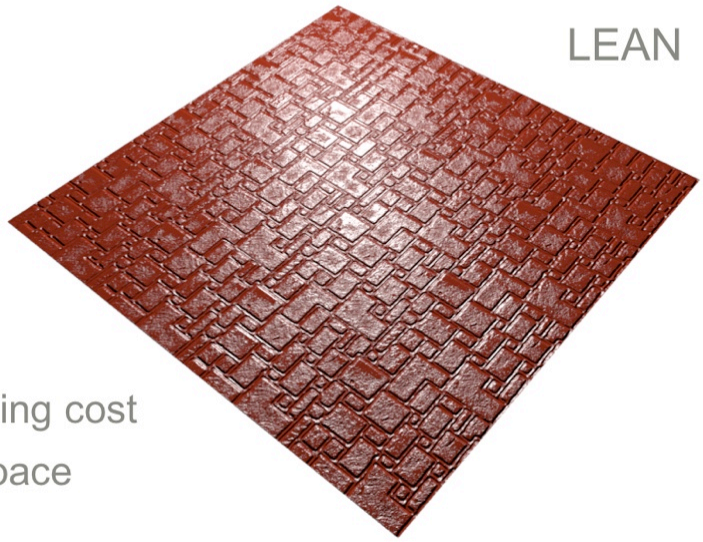
- More accurate results
- Anisotropic effects

Advances in Real-Time Rendering in 3D Graphics and Games

LEAN mapping results in a tighter highlight here, which is closer to the ground truth.

Options: Prefiltering (LEAN)

SIGGRAPH2012



Bad:

- Memory
- Extra shading cost
- Tangent space

Advances in Real-Time Rendering in 3D Graphics and Games

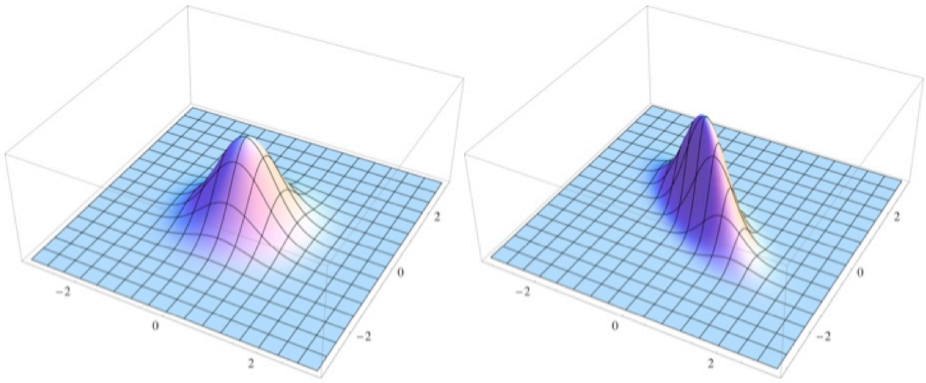
However, there are some downsides…

Memory in particular is a big issue. 16 bit storage is needed if you want to go to higher powers, as Dan already mentioned.

This is a major storage multiplier over DXT1 or DXT5, plus we need two textures!

LEAN Times

- Bivariate normal distribution $\Sigma = \begin{bmatrix} \Sigma_x & \Sigma_z \\ \Sigma_z & \Sigma_y \end{bmatrix} = \begin{bmatrix} \sigma_x^2 & \rho\,\sigma_x\,\sigma_y \\ \rho\,\sigma_x\,\sigma_y & \sigma_y^2 \end{bmatrix}$
- Visual explanation:



Before I cover some thoughts on overcoming these drawbacks, I'd like to briefly give some additional background on LEAN mapping that helped me understand things more intuitively.

On the left is a regular normal (Gaussian) distribution. It has a similar falloff to Blinn-Phong, particularly for higher specular powers.

LEAN mapping an extension of this – in fact it's a *bivariate* normal distribution. The covariance matrix that's reconstructed in the shader (as Dan mentioned earlier) describes this distribution. There are two variances (tangent and bitangent) and correlation, ρ. These change the shape and angle of the lobe.

It's easy to see how LEAN mapping can better approximate the distribution of normals. In particular, if there's more variance in a particular direction, the lobe will stretch out and produce an anisotropic highlight.

LEAN Times

Memory

- Bake, as with Toksvig…
  - Bilinear simulation still important!
- Store covariance matrix: [Σ_x, Σ_y, Σ_z]?

$$\Sigma = \begin{bmatrix} \Sigma_x & \Sigma_z \\ \Sigma_z & \Sigma_y \end{bmatrix}$$

- Can have precision issues

On the memory front, we could theoretically bake the covariance matrix offline and store that, similar to baking Toksvig.

This isn't strictly linearly filterable, but in practice it's not too bad.

However, there can still be precision problems, particularly with Σ_z.

LEAN Times

Memory

- Bake, as with Toksvig…
  - Bilinear simulation still important!
- Store 2 gloss values

$$\Sigma = \begin{bmatrix} \Sigma_x & \Sigma_z \\ \Sigma_z & \Sigma_y \end{bmatrix} \approx \begin{bmatrix} \Sigma_x & 0 \\ 0 & \Sigma_y \end{bmatrix}, \text{store } \frac{\log_2(1/\Sigma_{\{x,y\}})}{13}$$

- Could use BC5 or DXT5
- Optionally store correlation: $\rho = \Sigma_z/\text{sqrt}(\Sigma_x\Sigma_y)$

Another option is to store two gloss values in log space. This is a popular encoding for gloss, as it's pretty linear in terms of highlight size. This makes it intuitive to paint, and it also behaves well with texture filtering and compression.

The correlation could be stored too, but you may find that this isn't needed with your game content. For instance, in the case of the earlier brick texture, most anisotropy in the normals is strongly axis-aligned (along brick edges), so the correlation is low.

LEAN Times

Shading Cost

- Cheaper with two gloss values

```
// unpack normal
float4 t1 = tex2D(lean1, texcoord);
float3 N = float3(2*t1.xy - 1, t1.z);

// unpack gloss
float2 g = tex2D(gloss, texcoord);
float2 p = exp2(g*6.5);

// compute specular
float2 h = h.xy/h.z - N.xy/N.z;
h *= p;
float e = dot(h, h);
float spec = exp(-0.5*e)*p.x*p.y;
```

```
// unpack normal
float4 t1 = tex2D(lean1, texcoord);
float3 N = float3(2*t1.xy - 1, t1.z);

// unpack B and M
float4 t2 = tex2D(lean2, texcoord);
float2 B = (2*t2.xy - 1)*sc;
float3 M = float3(t2.zw, 2*t1.w – 1)*sc*sc;

// convert M to E
float3 E = M - float3(B*B, B.x*B.y);
float Det = E.x*E.y - E.z*E.z;

// compute specular
float2 h = h.xy/h.z - B;
float e = (h.x*h.x*E.y + h.y*h.y*E.x - 2*h.x*h.y*E.z);
float spec = (Det <= 0) ? 0 : exp(-0.5*e/Det)/sqrt(Det);
```


By storing two log-space gloss values, we can cut down on the shading complexity too.

LEAN Times

SIGGRAPH2012

Deferred

- Needs a surface frame…
- Store a quaternion?
  - Precision issues
  - Encode/decode overhead
  - Go tile-based? Forward+?

Quat.x	Quat.y	Quat.z	Quat.w/?
N.x	N.y	Gloss.x	Gloss.y

Standard quat
R8G8B8A8

The final challenge is with deferred shading. The LEAN distribution requires a tangent frame, so we need to store/reconstruct this somehow.

One compact option is to store a quaternion.

This image shows (0, 0, 1) rotated into world space using this quaternion, followed by lighting. With R8G8B8A8, this produces banding beyond what you would get from storing the world-space normal directly with the same precision, particularly if you compare against “Best Fit Normals” [Kaplanyan10].

LEAN Times

SIGGRAPH2012

Deferred

- Needs a surface frame…
- Store a quaternion?
  - Precision issues
  - Encode/decode overhead
  - Go tile-based? Forward+?

Quat.x	Quat.y	Quat.z	Quat.w/?
N.x	N.y	Gloss.x	Gloss.y

Reconstruct largest component + R10G10B10A2

Advances in Real-Time Rendering in 3D Graphics and Games

To improve precision over R8G8B8A8, we can get rid of the largest component, store the index and reconstruct it later. Precision can then be improved by rescaling the remaining components and using R10G10B10A2. See [Frykholm09] for more details.

This leads to more encode and decode overhead, although for tile-based deferred rendering, the decode cost is amortized.

There is still the per-light overhead of transforming the half vector into tangent space using the quaternion (6 instructions).

It's a classic tradeoff of storage cost vs. performance. I'm not recommending this approach for current consoles, where MRT space and shader cycles are typically at a premium, but it's a potential option for DX11.

Some space can be clawed back to make room for two gloss values by storing x and y of the tangent space normal (reconstructing z later).

AA Ahoy!

- Detail Normal Maps
- Geometry
- Diffuse
- Environment Maps

But we're not done. Specular aliasing from normal maps is just one of several issues!

Detail Normal Maps

- 1. Store [x, y, 0, variance]
- 2. Use a decent normal blending method
  - See: http://blog.selfshadow.com/publications/blending-in-detail/
- 3. Combine base gloss with detail variance

$$\frac{1}{s'} = \frac{1}{s} + \sigma^2$$

- 4. Fade out to constant variance

Let's say that we've gone ahead and implemented specular AA based on normal map roughness.

What happens later if we want to slap a *detail* normal map on top?

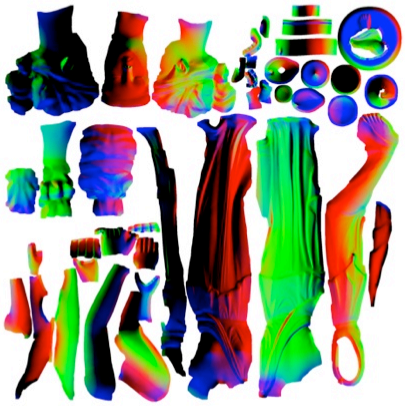
One (statistically sound) solution is to sum the variance. This works so long as the two normal maps are not strongly anti-correlated – i.e. they don't cancel each other out. (See the LEAN Mapping paper for more details).

To do this, take the reciprocal of the existing specular power, s, add the variance from the detail map, then finally invert again to get the new power, s'.

This is a simple rearrangement of the second equation for Toksvig mapping covered earlier. I find this version a little easier to remember and it could be convenient if you're summing additional variance from other sources. Other sources you say? Well, let's see…

Geometry AA

- Another source of variance!
- Idea 1: prefilter geometric normals
  - 1. Dilate
  - 2. MIP
  - 3. Toksvig
- Needs atlas!



Geometry is another source of normal variance and therefore aliasing!

We could prefilter our object-space normals and use Toksvig to get the variance.

However, just like TSS, we would need a unique parameterisation (an ‘atlas’), which isn’t convenient!

Still, this gives us something to compare against.

Geometry AA

- Idea 2: Pixel Quad Message Passing [Penner11A]
- Access neighbours ⇒ average ⇒ variance
- Code for average:

```
float2 dir = 0.5 - frac(vpos*0.5 - 0.25)*2;

float3 n0 = N;
float3 n1 = ddx_fine(n0)*dir.x;
float3 n2 = ddy_fine(n0)*dir.y;
float3 n3 = ddy_fine(n1)*dir.y;

float3 nn = n0 + n1 + n2 + n3;
```

Instead, we can adapt Eric Penner's Pixel-Quad Message Passing technique (presented at Advances last year) to access the other normals in the pixel quad at run time.

Note: high-quality derivatives are available with DX10/11 or PS3.

We can then average the normals and calculate variance.

Note: this code has been optimised a little, which explains the lack of a *0.25. Also, in case it's not obvious, these are interpolated vertex normals that have been renormalised.

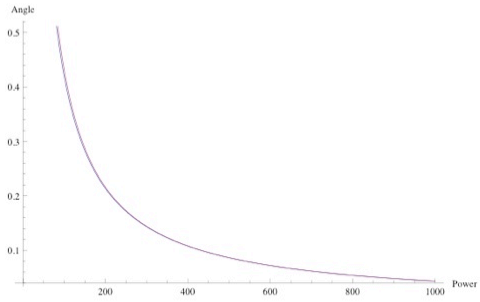
Behold, similar results to before!

(Please forgive the mesh flipping and different orientation; the last image comes from a RenderMonkey project, whereas this one comes from a separate DX11 test app.)

Geometry AA

SIGGRAPH2012

- Idea 3, from Kaplanyan & Valient:
- Combine normal cone (curvature) and specular lobe cone
  - 1. Convert spec power to cone angle
  - 2. Add curvature angle:




```
float3 dN = fwidth(N);
float3 new_normal = normalize(N + dN);
float curvature = acos(dot(new_normal, N))/(pi*0.5);
```

  - 3. Convert back to new power
- Being used in production now!

Anton Kaplanyan was kind enough to share an alternative method developed in collaboration with Michal Valient at Guerrilla Games.

This process essentially works by adding spread/cone angles instead of variance.

First we convert the original specular power to a cone angle. (I won't go into the details, but this is based on a curve that relates specular power to the solid angle of the highlight above a threshold.)

Next, we calculate a delta normal based on the deviation (fwidth) of the adjacent normals.

We then add this angle and convert back to a specular power.

Geometry AA

- Optimised code:

```
float3 dN = fwidth(N);
float3 new_normal = normalize(N + dN);
float curvature = sqrt(1 - dot(new_normal, N));

float angle = 4.11893/sqrt(power) + curvature;
power = 16.9656/(angle*angle);
```

- Similar results
- Future work

Using a cheap approximation to acos (there's no direct GPU support for this, so it expands to several instructions), we arrive at a pretty compact result.

The results are similar to the variance version…

Geometry AA

- Optimised code:

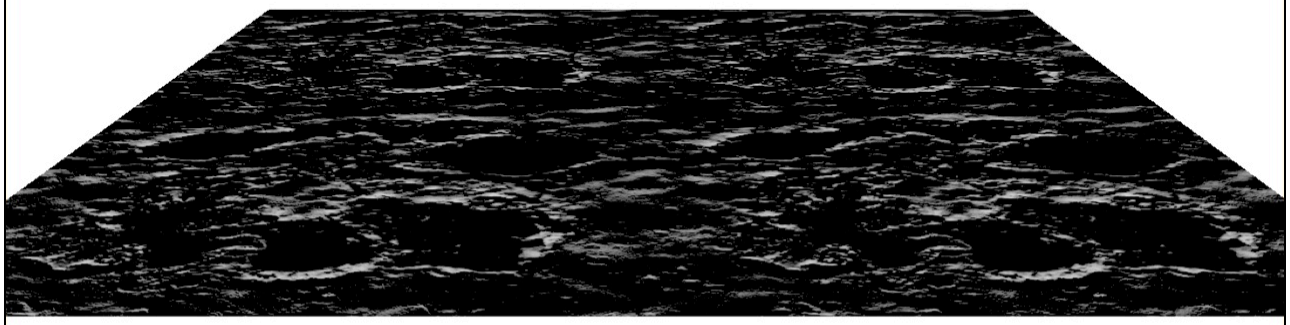
```
float3 dN = fwidth(N);
float3 new_normal = normalize(N + dN);
float curvature = sqrt(1 - dot(new_normal, N));

float angle = 4.11893/sqrt(power) + curvature;
power = 16.9656/(angle*angle);
```

- Similar results
- Future work

…here's the variance version again.

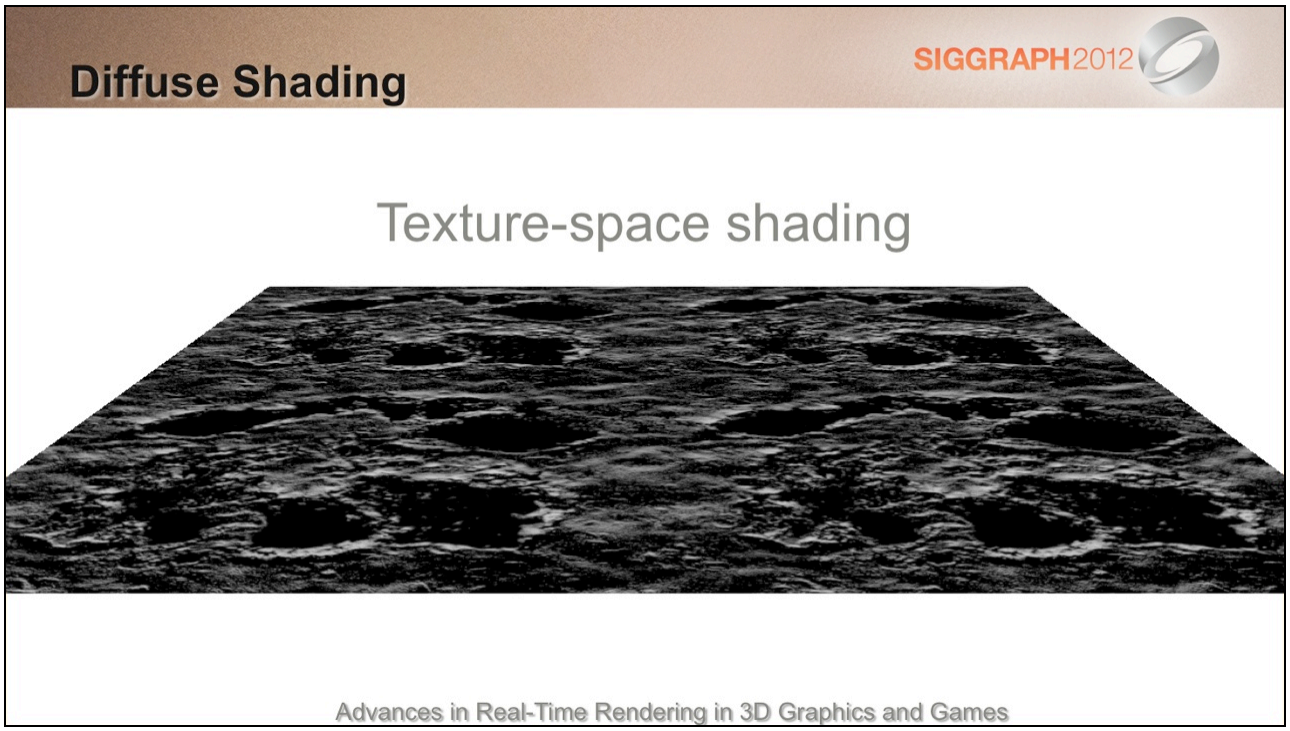
Diffuse Shading

Average normal



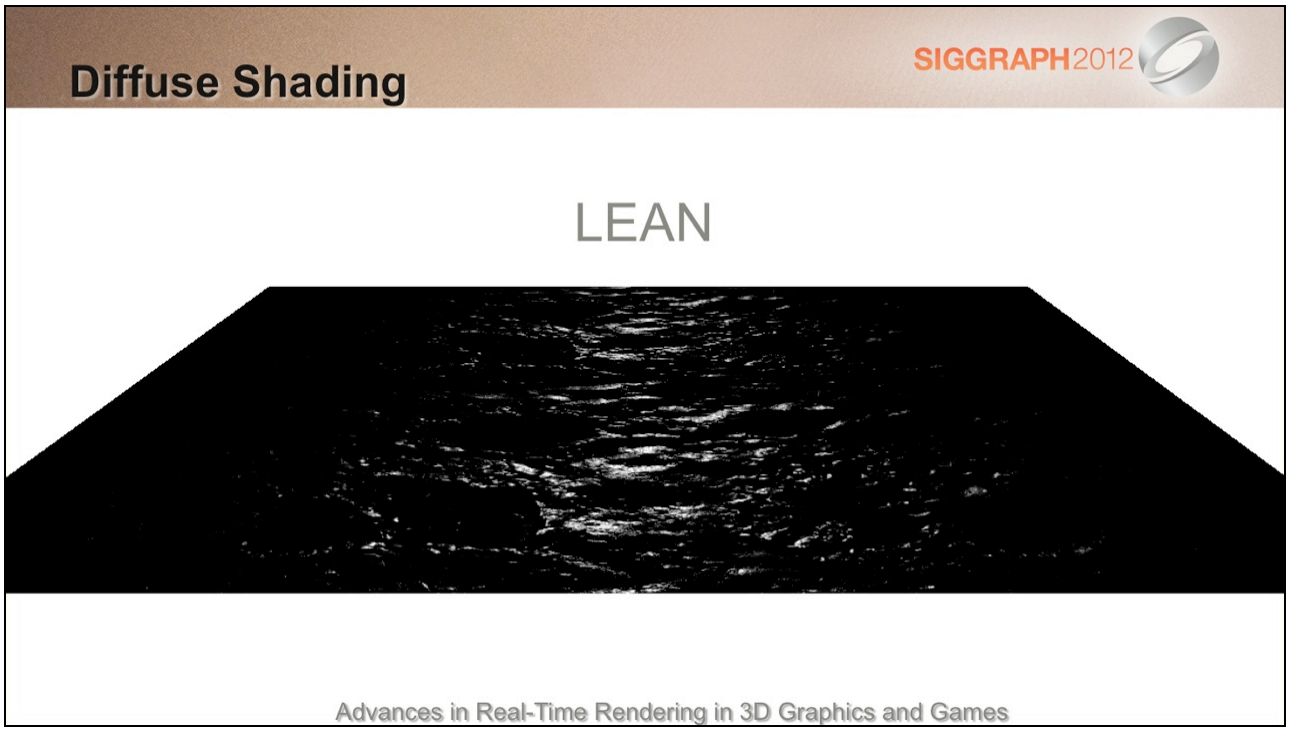
Diffuse shading isn't immune to aliasing either and we can get the same problems as specular under normal map minification.

Here's a plane lit with a light at a grazing angle using the filtered (averaged and renormalised) bump normal.



If we compare against TSS we can see that we're losing a lot of luminance.

(Bump-level self-shadowing is another issue, which I'll be skipping over here.)



This can affect specular too, since N.L is part of the lighting equation outside of the BRDF.

Here's the result with TSS. It's a lot brighter and more detailed

Diffuse Shading: What's Wrong?

- What we should be doing:

$$\int_{\Omega} (\underline{\mathrm{n}_i \cdot \mathbf{l}})(\underline{\mathrm{n}_i \cdot \mathbf{h}})^p ds$$

What we should be doing is a full integral of the specular BRDF and N.L for all of the original normals in the footprint of the current pixel.

(You can think of this as a discrete weighted sum instead, but I've use an integral to convey a continuous signal, including interpolated normals).

Note: I've simplified things here quite a bit, by stripping the BRDF down, ignoring energy conservation, for instance. Ideally, some other terms should be considered too.

Diffuse Shading: What's Wrong?

- What we should be doing:

$$\int_{\Omega} (\underline{\mathrm{n}_i \cdot \mathbf{l}})(\underline{\mathrm{n}_i \cdot \mathbf{h}})^p ds$$

- What we're doing now:

$$(\underline{\mathrm{n}_a \cdot \mathbf{l}}) \int_{\Omega} (\underline{\mathrm{n}_i \cdot \mathbf{h}})^p ds$$

What we're doing now (with Toksvig or LEAN) is approximating the integral on the right (just the specular BRDF) but we're still using the average normal for N.L.

Diffuse Shading: What's Wrong?

- What we should be doing:

$$\int_{\Omega} (\underline{\mathrm{n}_i \cdot \mathbf{l}})(\underline{\mathrm{n}_i \cdot \mathbf{h}})^p ds$$

- What we're doing now:

$$(\underline{\mathrm{n}_a \cdot \mathbf{l}}) \int_{\Omega} (\underline{\mathrm{n}_i \cdot \mathbf{h}})^p ds$$

- Compromise:

$$\boxed{\int_{\Omega} (\underline{\mathrm{n}_i \cdot \mathbf{l}}) ds} \int_{\Omega} (\underline{\mathrm{n}_i \cdot \mathbf{h}})^p ds$$

Approximating the full integral is a bit of a challenge, but we can get closer to the ground truth by factoring the full integral into two separate integrals.

So now we just need to find a reasonable approximation to the left-hand side.

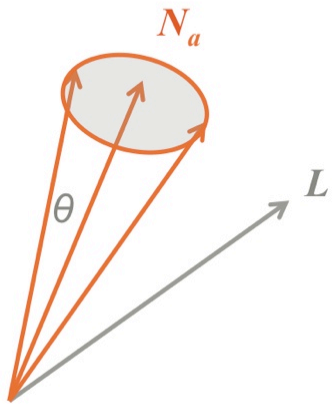
The result can then be used both for diffuse AA and for better specular AA.

Diffuse Shading: Solution

- Normal variance ⇒ cone
- Lighting integral for cone of normals around average normal (Na)
- Cone angle: cos(θ) = 2*length(N_a) - 1

$$\int_{v=0}^{1}\int_{u=0}^{1} \max(\mathrm{UniformSampleCone}(u, v, \cos\theta) \cdot L, 0)$$

See: Physically Based Rendering 2nd Ed.



Back to cones!

We can trivially convert normal variance to a cone angle: cos(angle) = 2*|Na| - 1

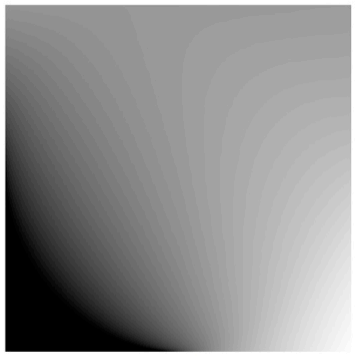
Then, an we can approximate the earlier integral as an integration of N.L over this cone of normals.

Diffuse Shading: Solution

Could precompute integral…

- Just like Pre-Integrated Skin Shading [Penner11B]

```
float len = length(Na);
float3 N = Na/len;
tex2D(LUT,
  float2(dot(N, L)*0.5 + 0.5, len));
```



This integral can be precomputed via numerical integration and stored in a LUT.

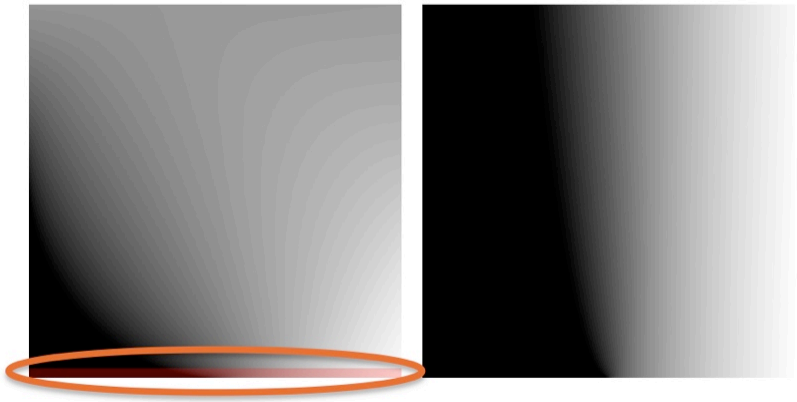
This is just like Eric Penner's Pre-Integrated Skin Shading technique (again, presented last year at Advances).

Diffuse Shading: Solution

SIGGRAPH2012

Shrinking the LUT

- Important region: first ~25 degrees



In practice, we can focus in on a smaller cone of normals, reducing the size of the texture and therefore the potential for texture cache thrashing.

We could do the same for the x axis too.

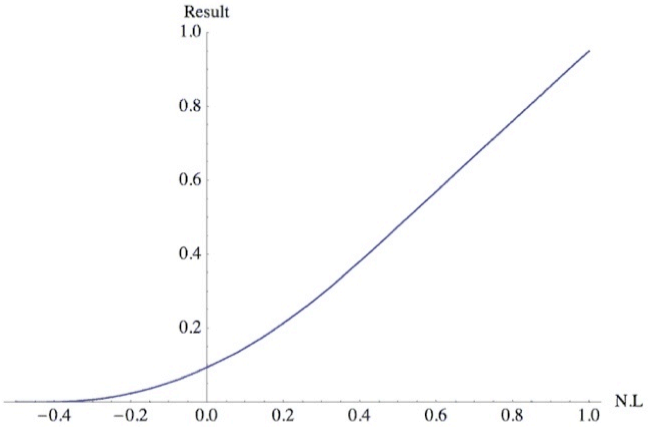
The first 25 degrees seems to make the biggest difference in the cases I've seen so far. Beyond that you start to get unwanted 'wrap' that looks wrong with unshadowed lighting.

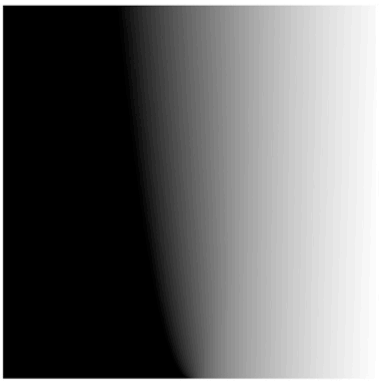
Diffuse Shading: No LUT

SIGGRAPH2012

Avoiding a LUT

- Curve fit: sliding tail, x^2





Advances in Real-Time Rendering in 3D Graphics and Games

It's possible to go a step further and replace the LUT entirely by approximating the curve.

It turns out that the 'tail' is close to an x^2 curve that joins with a straight line (regular N.L).

The video shows how things change with variance.

Diffuse Shading: No LUT

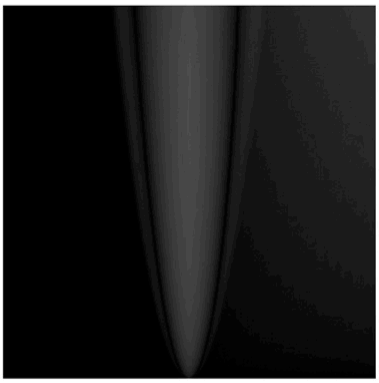
SIGGRAPH2012

Avoiding a LUT

- Boil it down...

```
float DiffuseAA(float3 N, float3 L)
{
  float a  = dot(N, L);
  float w  = max(length(N), 0.95);
  float x  = sqrt(1.0 - w);
  float x0 = 0.373837*a;
  float x1 = 0.66874*x;
  float n  = x0 + x1;
  return w*((abs(x0) <= x1) ? n*n/x : saturate(a));
}
```

Difference x10



With the help of Mathematica, I've been able to boil this down to the following function.

The result is pretty close to the 'focused' LUT.

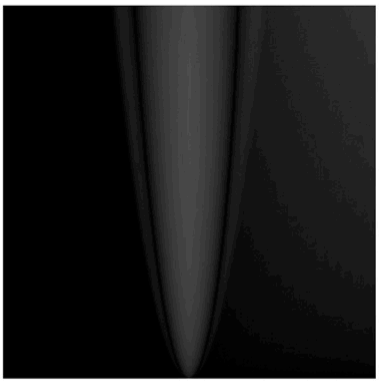
Diffuse Shading: No LUT

SIGGRAPH2012

Avoiding a LUT

- ~18 instructions (fxc) ☹
  - Arguably *too* good a fit!
  - Can probably approximate
- Toksvig-like issues:
  - Compressed normals
  - Store prefiltered length/variance

Difference x10



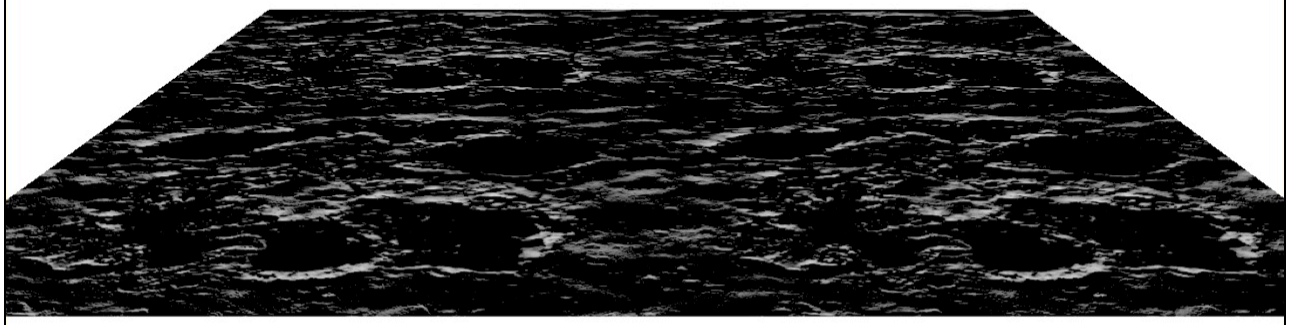
Unfortunately this costs too many shader instructions at the moment (it's per light!)
Some of this cost could be amortised, but it's likely that a cheaper approximation is possible.

We also face the same problem as with Toksvig when it comes to compressed normals,
so in practice the length of the normal or the variance should be stored somewhere.

Diffuse Shading: Results

SIGGRAPH2012

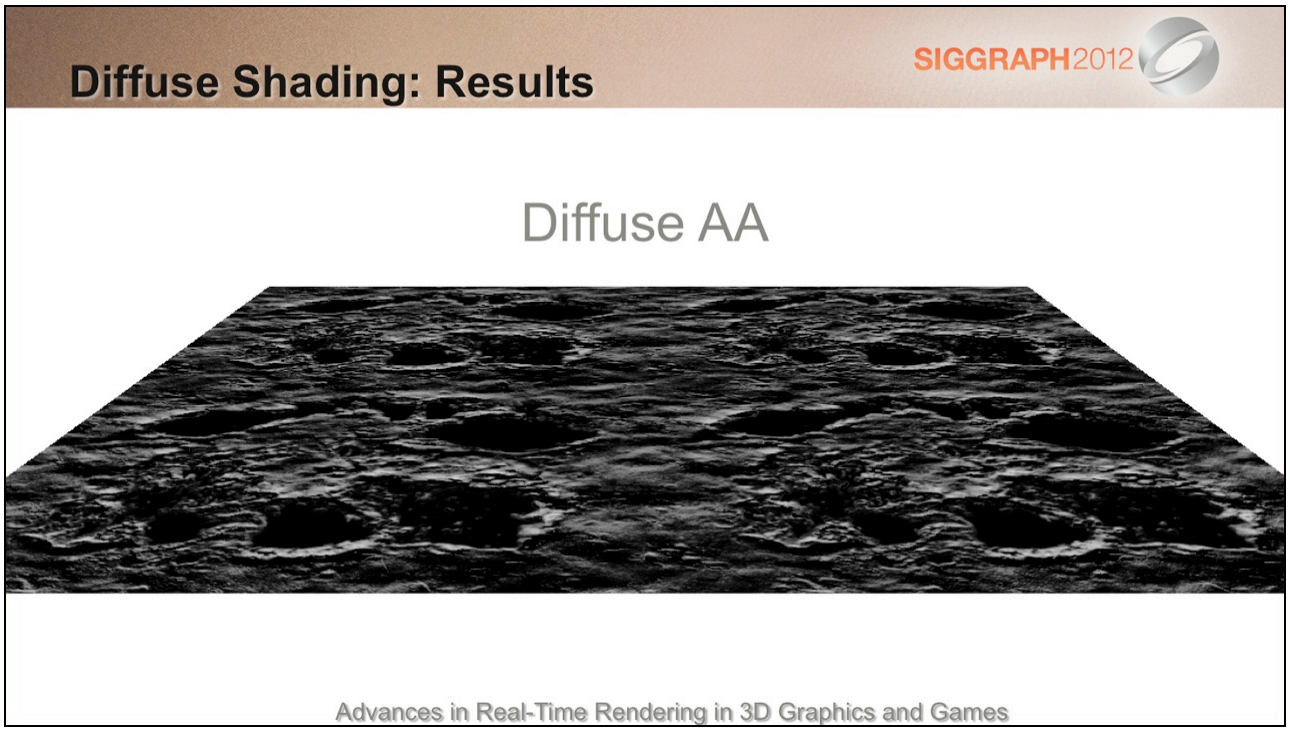
Average N



Advances in Real-Time Rendering in 3D Graphics and Games

Back to our problem case…

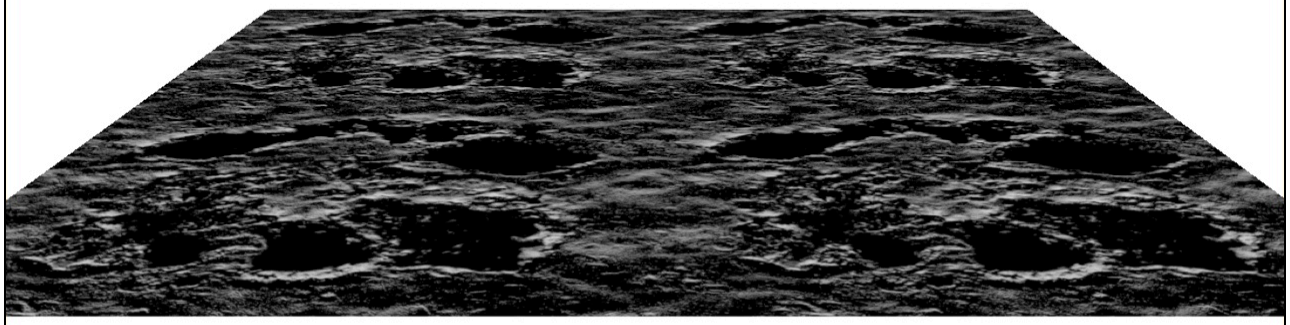
Diffuse Shading: Results



Here are the results with the integral. This looks promising…

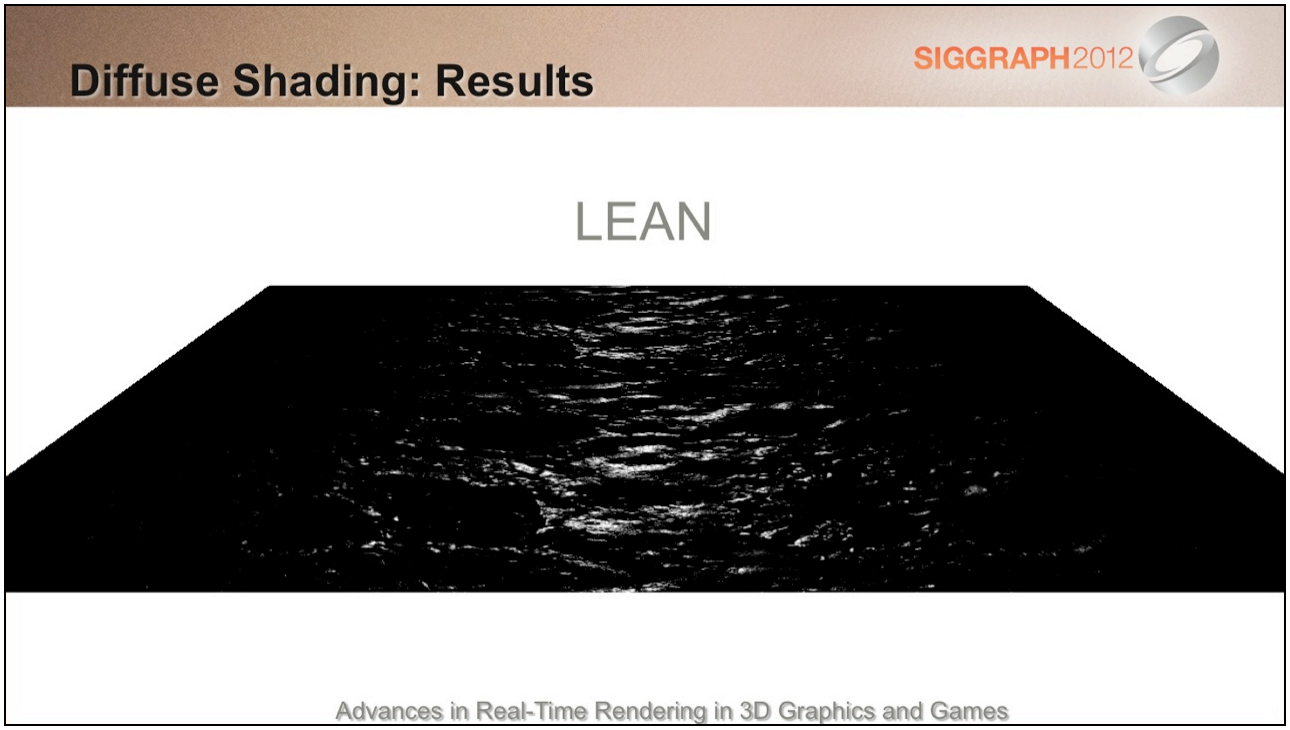
Diffuse Shading: Results

Texture-space shading



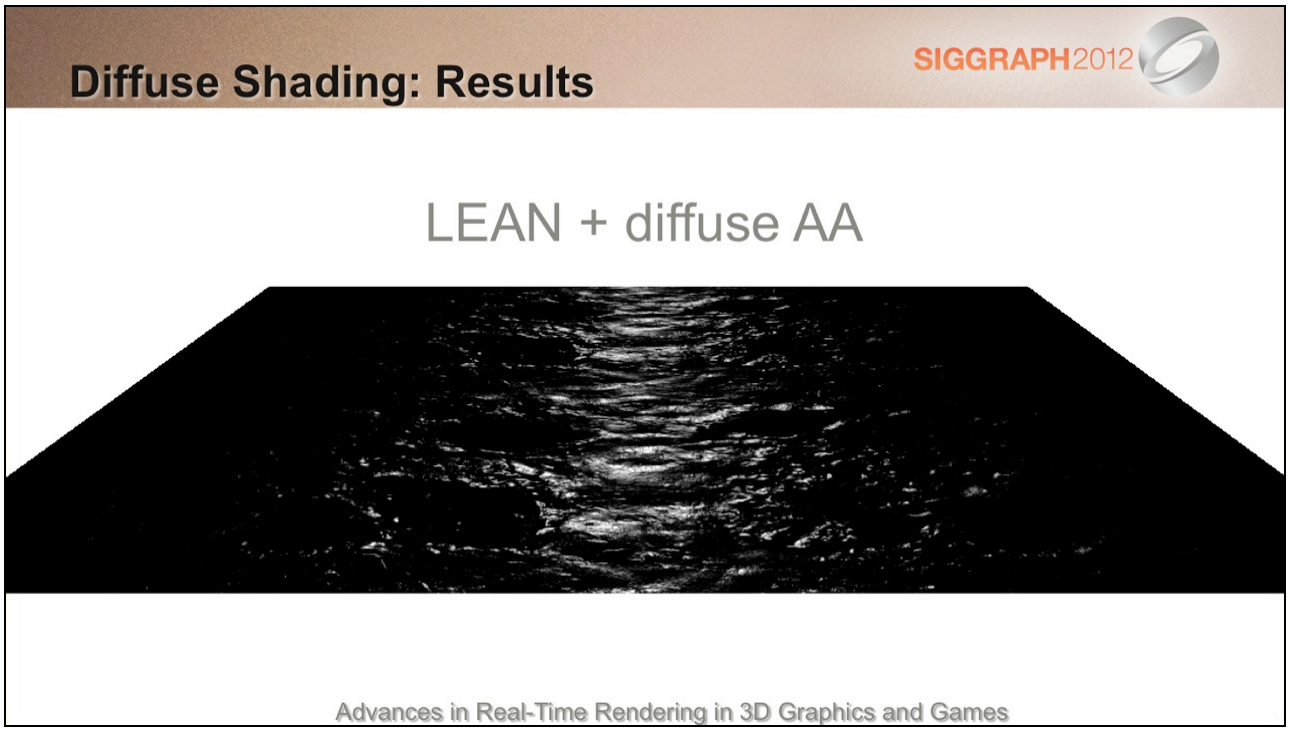
In fact it's a close approximation of TSS!

Diffuse Shading: Results



What about specular?

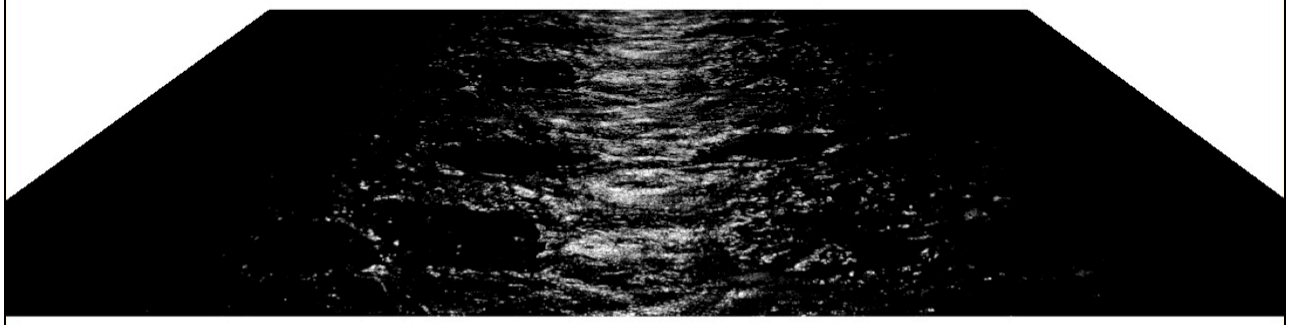
Diffuse Shading: Results



Ta da! Much brighter (and also more stable)

Diffuse Shading: Results

Texture-space shading



It's not identical to TSS, but there are several reasons for this:

1) Integral approximations. We've factored the full integral into two separate integrals
2) These simpler integrals have in turn been approximated (LEAN, diffuse AA)
3) Additionally, I'm actually undersampling the signal here with TSS (2048^2 not enough!). When I focused the resolution on a smaller section of the plane, the results were a little darker and smoother (i.e. a little closer to the LEAN + diffuse AA result)

Diffuse Shading: Results

Here's a link to a simple WebGL demo that shows the diffuse AA component.

As with specular AA, there's far less temporal shimmering and appearance is better preserved at lower mip levels with "Diffuse AA" enabled (zoom out, or adjust the mip bias).

Environmental Lighting

Standard approach:

- 1. Prefilter with Phong lobe
- 2. `tex2Dlod(Env, float4(R, bias));`

Room for improvement:

- Microfacet model
- Specular lobe depends on L and V!

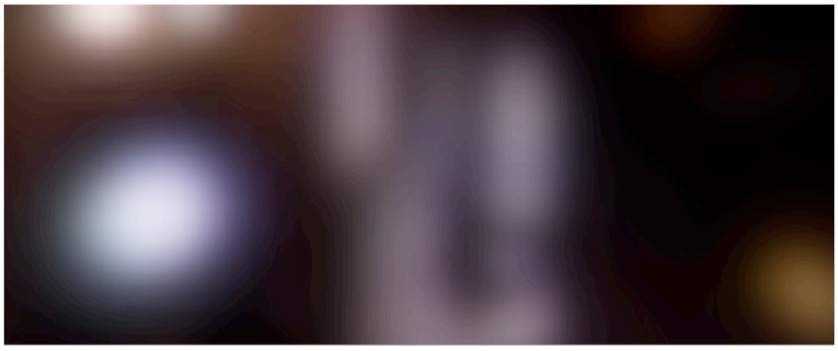
Environment mapping can be another major source of aliasing.

Toksvig gloss adjustment in conjunction with Phong prefiltering is a good option here.

However, it would be nice to go a step beyond this and use a microfacet BRDF.

Environmental Lighting

- Phong:

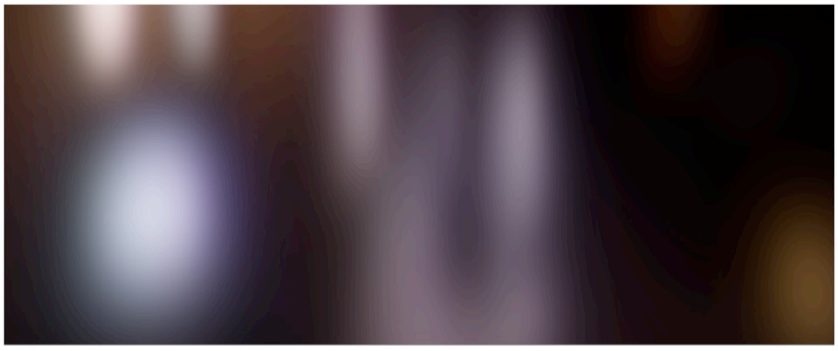


- Filtered Importance Sampling [Colbert08]

Here's Phong prefiltering on a plane. Highlights remain the same shape independent of view angle.

Environmental Lighting

- Blinn-Phong:

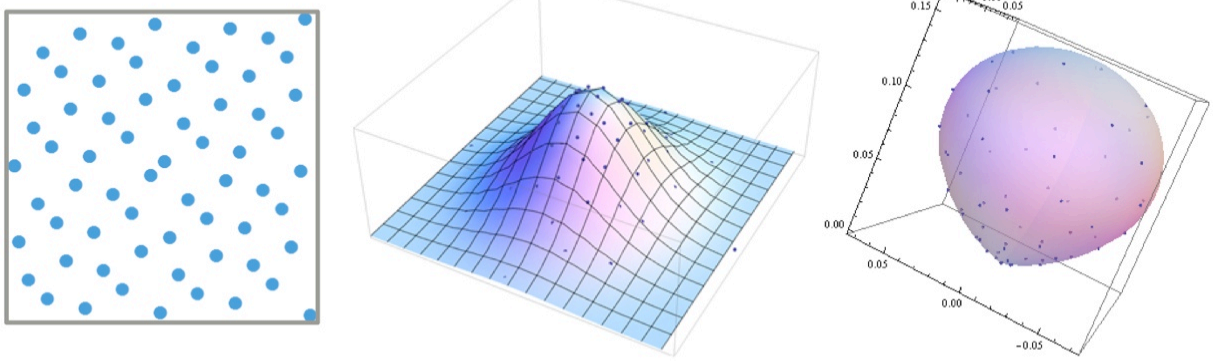


- Filtered Importance Sampling [Colbert et al. 07]

Here's the same result with Blinn-Phong. The highlights spread out at shallower angles, which is more realistic.

Filtered Importance Sampling (FIS) is one route to achieving this. It involves taking multiple weighted (and mip-biased) samples of the environment map.

Environmental Lighting



Random numbers

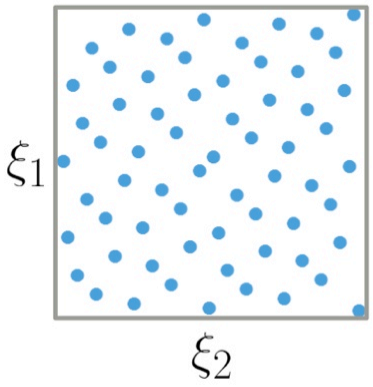
Warp to distribution

Generate sample directions

The sample directions importance sample the specular distribution.

There are several steps to this process.

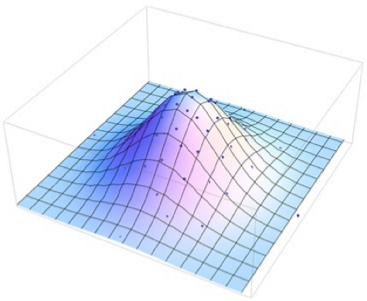
Environmental Lighting: Beckmann



$$h_x = \sigma\sqrt{-2\log\xi_2}\cos 2\pi\xi_1$$
$$h_y = \sigma\sqrt{-2\log\xi_2}\sin 2\pi\xi_1$$
$$H = \text{normalize}([h_x, h_y, 1])$$

We start off with a uniform distribution of random numbers.
(A low-discrepancy 2D point set like Hammersley is one option here.)

Environmental Lighting: Beckmann



sqrt(1/s)

Box-Muller transform

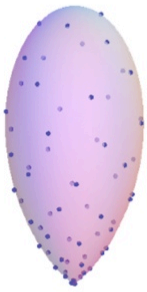
$$h_x = \sigma\sqrt{-2\log\xi_2}\cos 2\pi\xi_1$$
$$h_y = \sigma\sqrt{-2\log\xi_2}\sin 2\pi\xi_1$$
$$H = \text{normalize}([h_x, h_y, 1])$$

Next we can use the Box-Muller transform to generate a normal distribution. In this case for (isotropic) Beckmann – a close match to Blinn-Phong, as Dan showed.

In practice, this can be done offline and then scaled by the standard deviation (sqrt[1/power]) on the fly.

This warps the random points to fit the distribution.

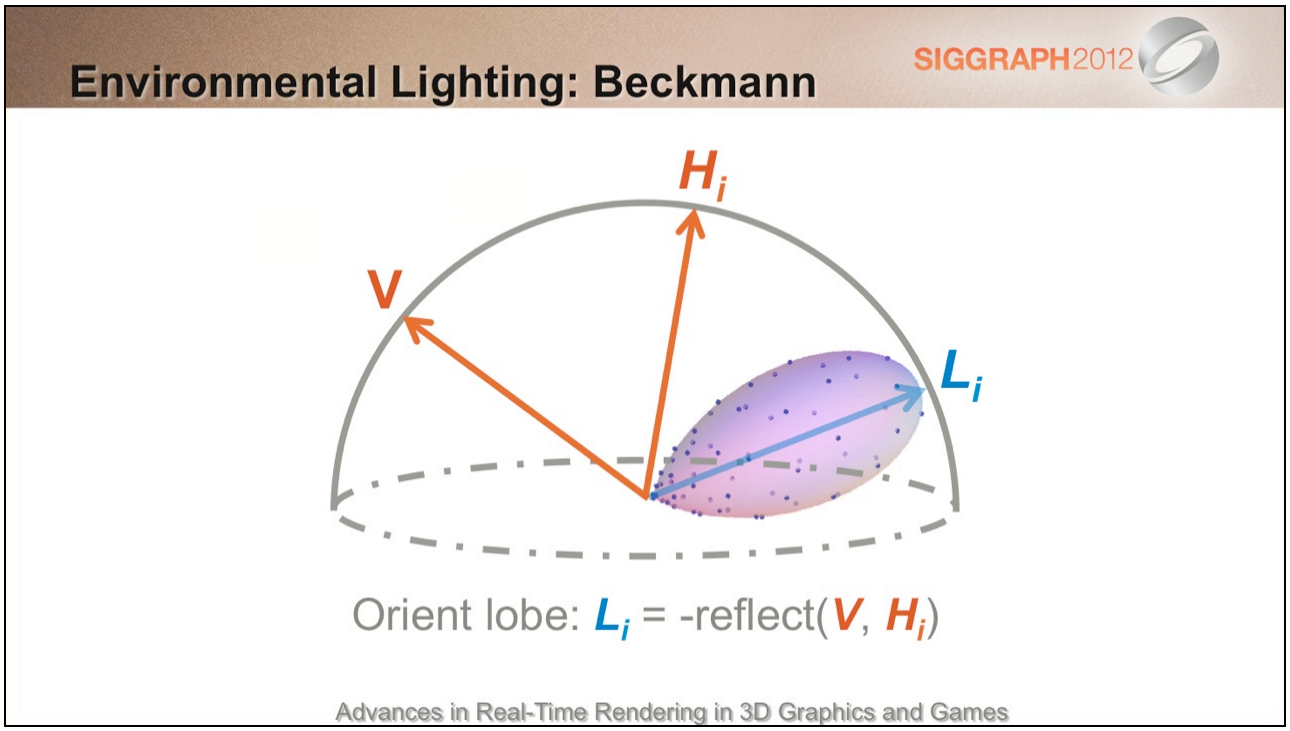
Environmental Lighting: Beckmann



Specular lobe
(H scaled by distribution
for visualisation)

$$h_x = \sigma\sqrt{-2\log\xi_2}\cos 2\pi\xi_1$$
$$h_y = \sigma\sqrt{-2\log\xi_2}\sin 2\pi\xi_1$$
$$H = \text{normalize}([h_x, h_y, 1])$$

Next, ‘unprojecting’ onto the upper hemisphere gives us a half vector.



Finally, we can generate a sampling direction, Li, by reflecting the view vector about this half-angle vector, Hi.

(Well, there is a final, final step, which is to transform this direction from tangent space to world space, for lookup.)

This is sort of the opposite of what we normally do when lighting with point sources, where we have a known light and view vector, from which we calculate the half vector.

Environmental Lighting: LEAN

LEAN Importance Sampling

- Start with regular Box-Muller:

$$z_x = \sqrt{-2\log(\xi_2)}\cos(2\pi\xi_1)$$
$$z_y = \sqrt{-2\log(\xi_2)}\sin(2\pi\xi_1)$$

This process can easily extended to the bivariate normal distribution used by LEAN mapping.

As before, we use Box-Muller to generate a normal distribution.

Environmental Lighting: LEAN

SIGGRAPH2012

LEAN Importance Sampling

- Start with regular Box-Muller:

$$z_x = \sqrt{-2\log(\xi_2)}\cos(2\pi\xi_1)$$
$$z_y = \sqrt{-2\log(\xi_2)}\sin(2\pi\xi_1)$$

- Use LEAN distribution:

$$\Sigma = \begin{bmatrix} \Sigma_x & \Sigma_z \\ \Sigma_z & \Sigma_y \end{bmatrix} = \begin{bmatrix} \sigma_x^2 & \rho\,\sigma_x\,\sigma_y \\ \rho\,\sigma_x\,\sigma_y & \sigma_y^2 \end{bmatrix}$$

Next, we use the covariance matrix…

Environmental Lighting: LEAN

LEAN Importance Sampling

- Start with regular Box-Muller:

$$z_x = \sqrt{-2\log(\xi_2)}\cos(2\pi\xi_1)$$
$$z_y = \sqrt{-2\log(\xi_2)}\sin(2\pi\xi_1)$$

- Use LEAN distribution:

$$\Sigma = \begin{bmatrix} \Sigma_x & \Sigma_z \\ \Sigma_z & \Sigma_y \end{bmatrix} = \begin{bmatrix} \sigma_x^2 & \rho\,\sigma_x\,\sigma_y \\ \rho\,\sigma_x\,\sigma_y & \sigma_y^2 \end{bmatrix}$$

- Warp the points:

$$h_x = \sigma_x z_x + \mu_x$$
$$h_y = \sigma_y\left(\rho z_x + \sqrt{1-\rho^2}\,z_y\right) + \mu_y$$

…to warp the points to the distribution.

Note: mu_x and mu_y come from the projected average normal.

Environmental Lighting: LEAN

SIGGRAPH2012

LEAN Importance Sampling

- Start with regular Box-Muller:

$$z_x = \sqrt{-2\log(\xi_2)}\cos(2\pi\xi_1)$$
$$z_y = \sqrt{-2\log(\xi_2)}\sin(2\pi\xi_1)$$

- Use LEAN distribution:

$$\Sigma = \begin{bmatrix} \Sigma_x & \Sigma_z \\ \Sigma_z & \Sigma_y \end{bmatrix} = \begin{bmatrix} \sigma_x^2 & \rho\,\sigma_x\,\sigma_y \\ \rho\,\sigma_x\,\sigma_y & \sigma_y^2 \end{bmatrix}$$

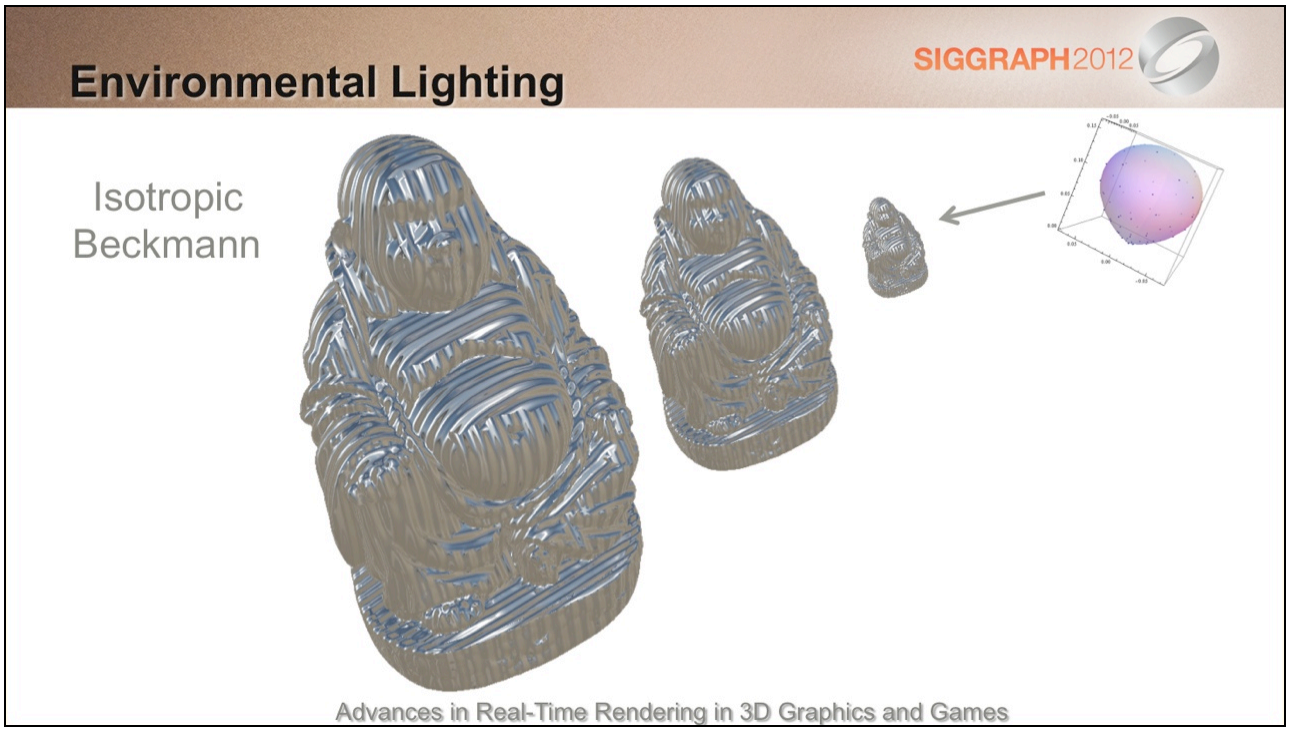
- Warp the points:

$$h_x = \sigma_x z_x + \mu_x$$
$$h_y = \sigma_y\left(\rho z_x + \sqrt{1-\rho^2}\,z_y\right) + \mu_y$$

- 'Unproject' as before:

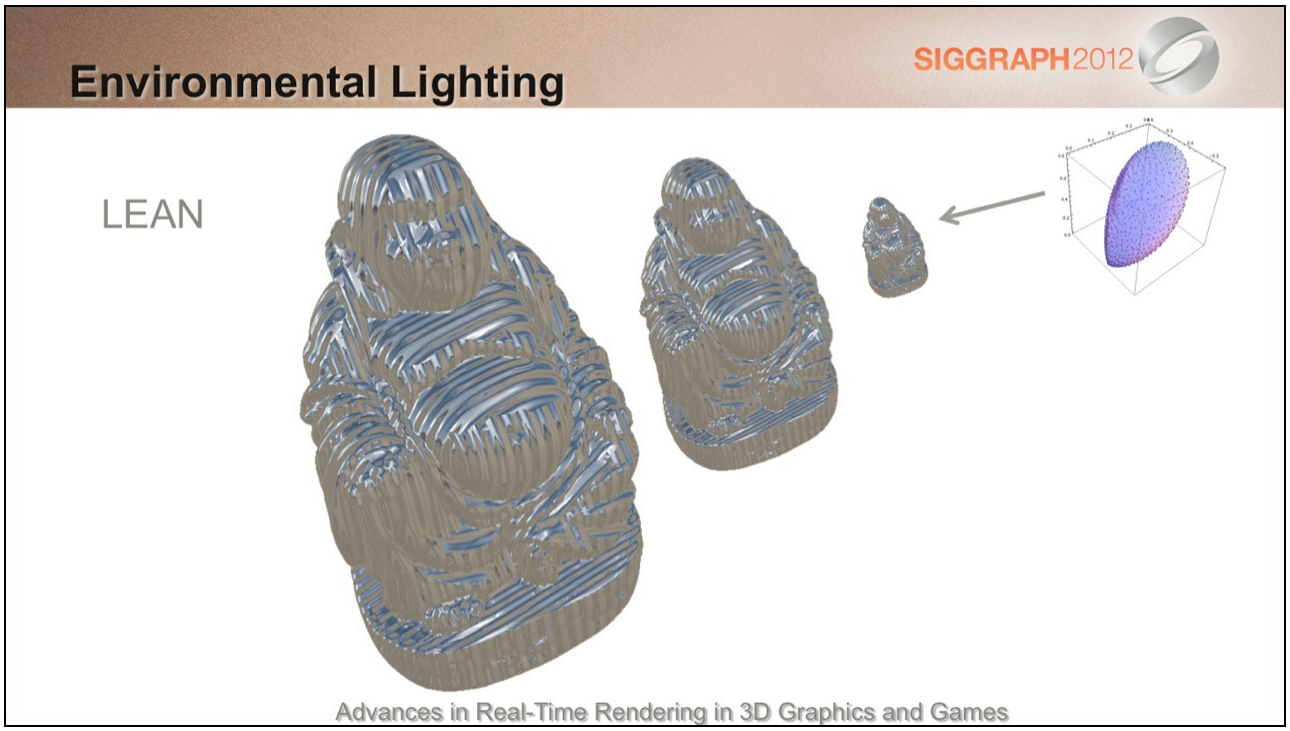
$$H = \text{normalize}([h_x, h_y, 1])$$

Then we unproject to get the half vector and reflect as before.



Either distribution produces the stretched highlights shown earlier.

What's interesting is what happens with a strongly grooved (anisotropic) normal map.
For Beckmann (or Blinn-Phong), the result in the distance is overly diffuse.



On the other hand, with the LEAN distribution, the distant appearance is similar to the high-res version.

Environmental Lighting

- Next step: Approximate with anisotropic tap(s)
- Covariance ⇒ Ellipse

Advances in Real-Time Rendering in 3D Graphics and Games

Unfortunately, quite a few samples are needed with FIS. More samples are needed for highly anisotropic cases (64+), whereas you can get away with less for semi-glossy, isotropic situations.

It's possible that a different parameterisation and anisotropic texture fetches could result in a cheap approximation.

Environmental Lighting

SIGGRAPH2012

Expensive, but useful framework:

- BRDF prototyping (See also: BRDF Explorer - Disney)
- Faster Phong prefiltering (use Multiple Importance Sampling?)
- Area lighting…

Advances in Real-Time Rendering in 3D Graphics and Games

In spite of this, it's a useful framework to have around for prototyping purposes.

The approach could also be used to accelerate cubemap prefiltering.

Part 4

Civilization V and Beyond

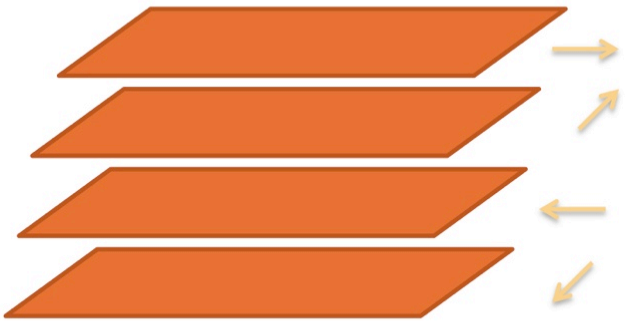
(Dan Baker)

Civilization V

- Used LEAN mapping for water
- Scale issues: Water at a distance behaves much like crumpled aluminum

Civilization V

- 4 scrolling planes to simulate waves
- Interference patterns surprisingly convincing
- Can add LEAN maps together with appropriate cross terms



Our Current Model

- CLEAN/LEAN depending on needs, for analytic lights
- Pre-convolved (Phong) cubemap array* for reflection
  * MIPs suck for low powers
- Power computed from LEAN data
- Simplifies art model: Just need a few parameters to be expressive

Cubemap array index = the average of covariance matrix diagonal (top left, bottom right) inverted, times 4.

Future Work

- Continuous LOD (detail maps, normals, geometry)
- Efficient, accurate cubemap filtering (LEAN)
- Investigate energy conservation of LEAN under minification
- Factor other terms (shadowing, masking, etc.)

References

[Baker05] Advanced Lighting Techniques, Meltdown 2005.

[Baker11] Spectacular Specular: LEAN and CLEAN Specular Highlights, GDC 2011.

[Bruneton12] A Survey of Non-linear Pre-filtering Methods for Efficient and Accurate Surface Shading, TVCG 2012.

[Colbert08] Real-Time Shading with Filtered Importance Sampling, EGSR 2008.

[Conran05] SpecVar Maps: Baking Bump Maps into Specular Response, SIGGRAPH 2005.

[Crassin11] Interactive Indirect Illumination Using Voxel Cone Tracing, Pacific Graphics 2011.

[Frykholm09] The BitSquid low level animation system, 2009.

[Kaplanyan10] CryENGINE 3: Reaching the Speed of Light, SIGGRAPH 2010.

[Lazarov11] Physically-based lighting in Call of Duty: Black Ops, SIGGRAPH 2011.

[McAuley12] Calibrating Lighting and Materials in Far Cry 3, SIGGRAPH 2012.

[Mittring11] The Technology Behind the DirectX 11 Unreal Engine "Samaritan" Demo, GDC 2011.

[Olano10] LEAN Mapping, I3D 2010.

[Penner11A] Shader Amortization using Pixel Quad Message Passing, GPU Pro 2, 201

[Penner11B] Pre-Integrated Skin Shading, GPU Pro 2, 2011.

[Han07] Frequency Domain Normal Map Filtering, SIGGRAPH 2007.

[Toksvig04] Mipmapping Normal Map

Acknowledgements

- Josh Barczak
- Marc Olano
- Anton Kaplanyan
- Michal Valient
- Eric Penner
- Stephen McAuley
- Vicki Ferguson
- Isabelle Gagnon
- Christian Diaz

We would like to thank various people for their help.

Questions?