The Tenth Planet

If it existed

The Blinn-Phong Normalization Zoo

Posted on 2011-08-25 by christian

It is good to see how physically based shading is finally gaining momentum in real time graphics and games. This is something I have been advocating for a long time. Developers are spreading the word. I was especially surprised to learn about Call of Duty: Black Ops joining the club [1]. Even a slick 60Hz-shooter with no cycles to spare can afford to do PBS today!

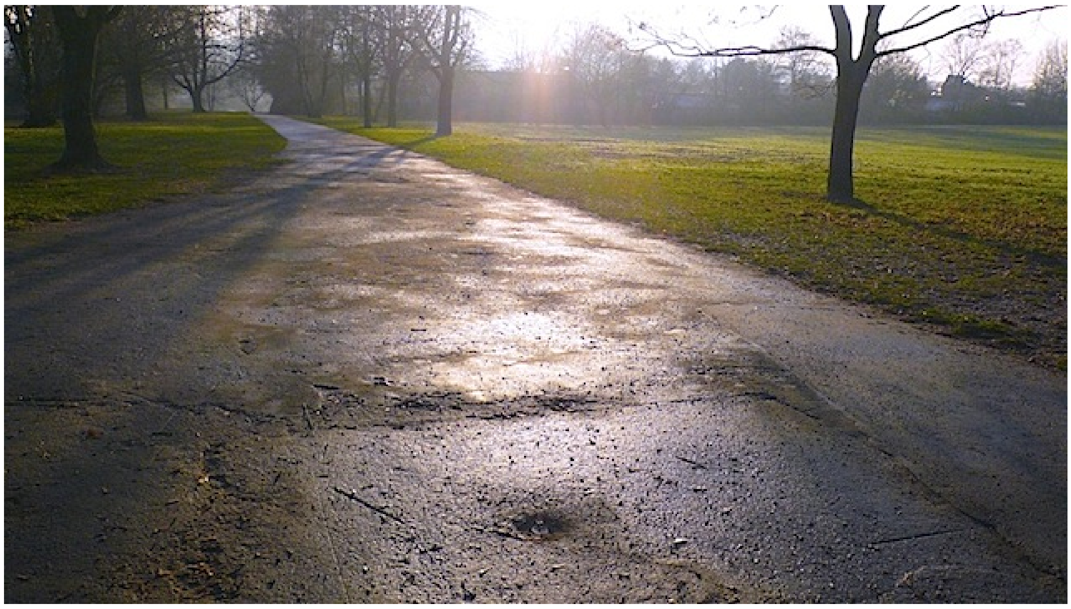
This leads me to the topic of this post, the normalization of the Blinn-Phong specular highlight. Why am I writing about it? It came to my mind recently with the current batch of publications from people adopting physically based shading models. This got me checking the maths again and I compiled a list with normalization factors for different shading models, given here in this post. I would also like to elaborate a little on the model that I wrote about in ShaderX^7 [2]. Be aware this post is a large brain dump.

Blinn-Phong Normalization: A paradigm shift for texturing

Let me first devote some lines to emphasize how this normalization business is not only a matter for engine developers or shader authors, but also for texture artist education. Normalization means, in simple terms, that the shading model scales the intensity of the specular highlight in proportion to its angular size, such that the total reflected energy remains constant with varying surface smoothness, aka. glossiness. This may sound like a minor technical detail, but it really is a paradigm shift.

STANDARD BLINN-PHONG	NORMALIZED BLINN-PHONG
Specular map has all the details	Gloss map has all the details
Glossiness constant	Specular level constant (almost)

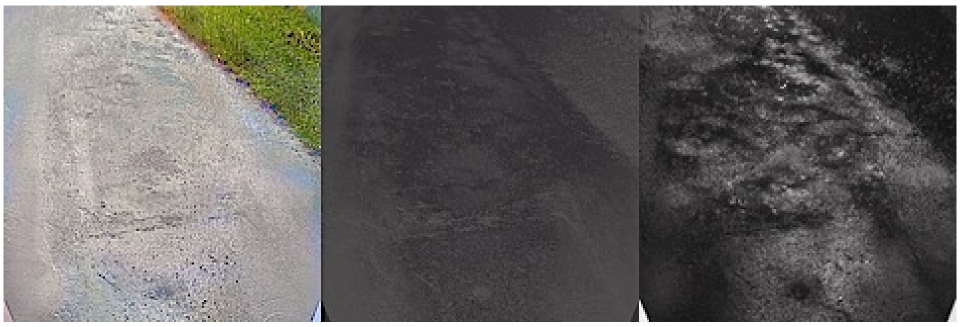
With standard Blinn-Phong shading, artists were used to paint surface variation into a texture connected to specular/reflective intensity, while the glossiness was only an afterthought, often not textured at all. With a normalized shading model, all the detail goes into the glossiness texture, while it is the specular intensity that becomes boring. So, is everything different now? In short, yes. And to drive the point home, I'll give a visual example.



— Wet and dry parts on an asphalt surface, in backlight

Consider the scene in the image above. The scene shows an an asphalt surface in backlight with dramatically different reflectances for wet and dry parts. Isn't this paradox? The refractive index of water is actually lower than that of the asphalt minerals, so everything else being equal, the water-covered surface should have less reflection. But things are not equal, because the water-covered parts have a much smoother surface. A smoother surface focuses the reflected sunlight into a narrower range of outgoing directions, making it more intense at the same time. This is the reason for the glaring parts of the photo.

With this said, I dabbled a little bit with GIMP to try and factor the above image into diffuse, specular and glossiness textures, much in the same way as I had advised artists to do when I was technical director at Replay Studios. The result is shown below.



— Factored into diffuse, specular and glossiness textures

The diffuse texture is a straightforward idealization of the subsurface color, devoid of any lighting and reflections. Since this cannot be observed directly, it must be inferred, but it should be much more saturated than the real-world appearance. I may be useful to have photos taken with a polarizing filter as a reference.

The specular texture defines the reflectivity for the case when light, viewer and normal are aligned. This is the minimum value, and the shading model will increase the reflectivity for any other configuration. There is not much artistic freedom here, since this value is connected to refractive index of the material. It is usually in the range of 2% to 5% for everything which is not a metal or a crystal. In the example above, there is almost no point in texturing it.

Instead, all the surface variation went into the glossiness map. This map is taken to be a logarithmic encoding of the Blinn-Phong specular exponent, which is a measure of surface smoothness. Black represents a single-digit exponent, indicating a very rough surface. White represents an exponent in the thousands, indicating a very smooth surface. Since the shading model is normalized, both the size and the intensity of the specular highlight is controlled with this value alone.

TEXTURE	PHYSICAL INTERPRETATION
Diffuse	Color due to average subsurface absorption
Specular	Fresnel reflectance at normal incidence
Gloss	Blinn-Phong exponent, a measure of surface smoothness

There is more to physically-based shading then just a normalized Blinn-Phong specular term. To get a dramatic and convincing contre-jour effect, it is important to model the behavior at grazing angles. This was one of the visual goals for the Velvet Assassin engine. See it realized, for example, in these scenes, and also pay attention to the wet puddles on the floor, which are mostly an effect of varying glossiness. Another play with varying glossiness can be seen the wet streaks on the fuel tank here. (I'm not affiliated with the commenter, btw, I just found these videos to be of acceptable quality.) As it turned out during the development, the Fresnel factor alone does not have enough 'omph'. It needed the combination of a Fresnel factor and the micro-facet geometric visibility factor to get the right effect at grazing angles. More on this below.

In the earlier engine for SpellForce 2, there was only one material with a per-pixel varying glossiness like that, the ice/crystal material. It is possible to get a hint of how it looked like here. Since this game was based on shader model 2, I needed to sacrifice shadow map taps to get the Blinn-Phong normalization in (without Fresnel). For all other materials, the glossiness was fixed and the normalization factor was a precomputed constant. What the engine did have was diffuse and specular hemisphere lighting. This allowed the artists to leave the diffuse texture black for parts of shiny metal, which is another important aspect of PBS. The soldier's armor in this scene is a good example of reflecting sky- and ground colors. After some time, even these relatively crude, "physically inspired" shading models proved to be really popular with the team.

Normalization vs Energy Conservation

When I speak about normalization I understand a different thing than energy conservation. In energy conservation, one looks at the total hemispherical reflectance for a given BRDF, and guarantees that it is strictly bounded by unity for all incoming light directions. Normalization, on the other hand, is just as strong as we define it. We can demand, for instance, that some parameter (the specular exponent in this case) has no effect the total hemispherical reflectance for a single, conveniently chosen incoming light direction. Then we have normalized it. In mathematical terms,

$$\text{for a given } \omega_i, \quad \int_\Omega f_r(\omega_i, \omega_o) \cos\theta_o \, d\omega_o = \text{const.}$$

It is possible (and may be even useful) to normalize BRDFs that are not energy conserving [8], like the original Phong and Blinn-Phong (the ones without the additional cosine term). It is also entirely possible to normalize something else than hemispherical reflectance, for instance we can normalize the micro-facet distribution function.

The Normalization Zoo

I verified the following list of normalization factors with the help of MuPAD, which is a great symbolic algebra tool. Unfortunately this is no longer available as a separate product, which is a real pity. I rate it second maybe only to Mathematica.

MODEL	RDF/INTEGRAND	NORMALIZATION	LITERATURE
Phong	$(\mathbf{R}\cdot\mathbf{L})^n$ $\cos^n\theta\sin\theta$	$\frac{n+1}{2\pi}$	
Phong (modified)	$(\mathbf{R}\cdot\mathbf{L})^n(\mathbf{N}\cdot\mathbf{L})$ $\cos^n\theta\cos\theta\sin\theta$	$\frac{n+2}{2\pi}$	[3,5,8]
Blinn-Phong	$(\mathbf{N}\cdot\mathbf{H})^n$ $\cos^n\frac{\theta}{2}\sin\theta$	$\frac{n+2}{8\pi} < \frac{n+2}{4\pi\left(2-\exp_2(-n/2)\right)} < \frac{n+4}{8\pi}$	[6]
Blinn-Phong (modified)	$(\mathbf{N}\cdot\mathbf{H})^n(\mathbf{N}\cdot\mathbf{L})$ $\cos^n\frac{\theta}{2}\cos\theta\sin\theta$	$\frac{n+6}{8\pi} < \frac{(n+2)(n+4)}{8\pi\left(\exp_2(-n/2)+n\right)} < \frac{n+8}{8\pi}$	[3,9]

MODEL	NDF/INTEGRAND	NORMALIZATION	LITERATURE
Blinn-Phong (NDF)	$(\mathbf{N}\cdot\mathbf{H})^n$ $\cos^n\alpha\sin\alpha$	$\frac{n+1}{2\pi}$	[2,4,5]
Blinn-Phong (heightfield)	$(\mathbf{N}\cdot\mathbf{H})^n$ $\cos^n\alpha\cos\alpha\sin\alpha$	$\frac{n+2}{2\pi}$	[10,11]

The upper part of the table shows the normalized *reflection density function* (RDF). This is the probability density that a photon from the incoming direction is reflected to the outgoing direction, and is the BRDF times $\cos\theta$. Here, θ is the angle between $\mathbf{N}$ and $\mathbf{L}$, which is, for the assumed view position, also the angle between $\mathbf{V}$ and $\mathbf{L}$, resp. $\mathbf{R}$ and $\mathbf{L}$. The literature column shows where I have seen these normalizations mentioned previously.

The lower part of the table shows the normalized *normal distribution function* (NDF) for a micro-facet model. This is the probability density that the normal of a micro-facet is oriented towards $\mathbf{H}$. It is the same expression in spherical coordinates than that for of the Phong RDF, just over a different variable, α, the angle between $\mathbf{N}$ and $\mathbf{H}$. The heightfield distribution does it slightly different, it normalizes the projected area of the micro-facets to the area of the ground plane (adding yet another cosine term).

Two Pi Or Not Two Pi

This is a good opportunity to elaborate on this π-business, because it can be confusing time and again. The RDFs and NDFs necessarily contain π, as the result of an integration over parts of a sphere. In the shader code however, π does usually not appear, because it cancels out when multiplying with irradiance, which is itself an integration over parts of a sphere.

Consider a point light source modeled as a small and very far away sphere, at distance r. Let this sphere have radius r_0 and a homogeneous emissive surface of radiance L. From the point of view of the receiving surface, the solid angle subtended by this sphere is a spherical cap, and so the irradiance integrates to $E = \pi L (\frac{r_0}{r})^2$. So there you have π in the formula for the irradiance of a point light, neatly canceling with the π from the RDF/NDF. This exercise can be done with all other types of illumination, it should always remove the factor π. (See the comment section for an in-depth explanation.)

Minimalist Cook-Torrance (ShaderX^7 style)

I wrote earlier in this post how it was important to capture the effects at grazing angles. The original Cook-Torrance model [7] has expensive factors for the distribution of normals (D), the Fresnel reflectance (F) and geometric occlusion of micro-facets (G). The normalized Blinn-Phong distribution can be used to greatly simplify D. A clever simplification for G was given by Kelemen and Szirmay-Kalos [4], by combining G with the foreshortening terms in the denominator; it should then be called the geometric *visibility* factor (V). The Schlick approximation is good to simplify F, but still with all the lerp'ing and one-minus'ing that is going on, it generates many shader instructions.

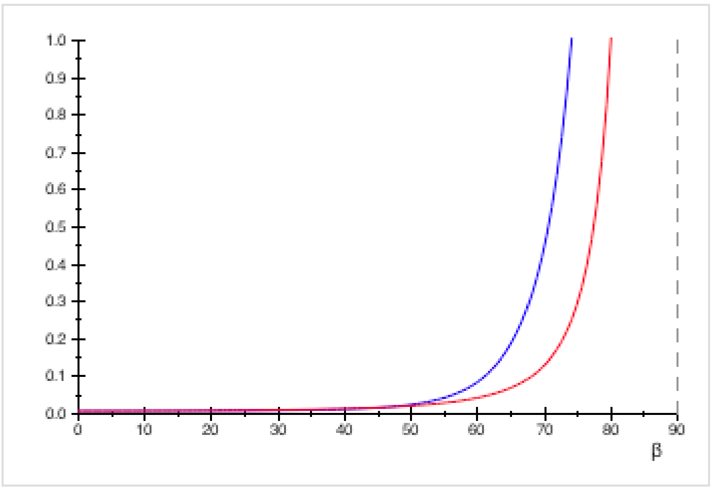
For Velvet Assassin, I devised a further approximation by combining the effect of F and V in a single expression with a significantly reduced instruction count, which I would call 'minimalist Cook-Torrance'. I looked at graphs and found, that in order to capture the essence, it suffices to divide by a higher power of $\mathbf{L} \cdot \mathbf{H}$, and get rid of F altogether. This is where the odd power of 3 comes from. In BRDF form, this is

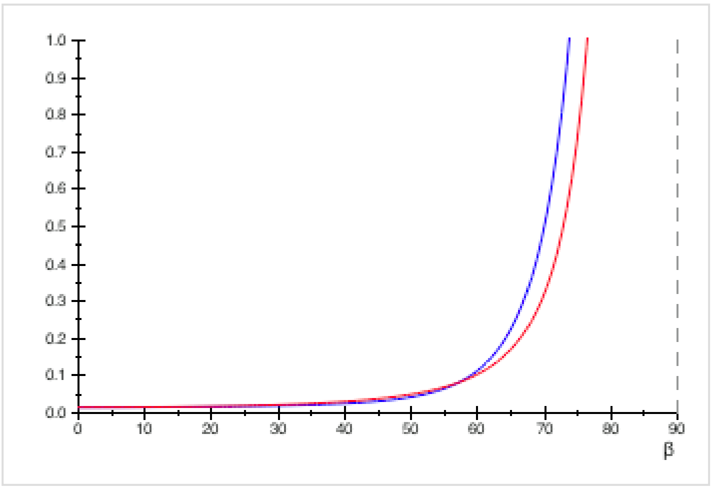
$$f_r = \frac{k_d}{\pi} + k_s \frac{(n+1)(\mathbf{N} \cdot \mathbf{H})^n}{8\pi(\mathbf{L} \cdot \mathbf{H})^3},$$

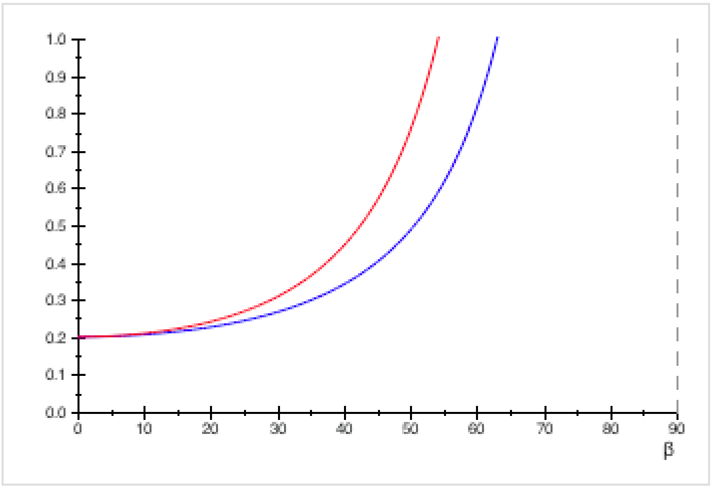
where k_d is the diffuse reflectance, k_s is the specular reflectance, and n is the specular exponent. This is the formula that was published in the ShaderX^7 article [2] and the one that shipped with Velvet Assassin (see here for the code). The formula can be decomposed into a contribution from D (the normalized Blinn-Phong NDF, see table above) and a contribution from a combined F and V:

$$f_r = \frac{k_d}{\pi} + k_s \mathrm{DFV}, \qquad \mathrm{D} = \frac{(n+1)}{2\pi}(\mathbf{N} \cdot \mathbf{H})^n, \qquad \mathrm{FV} = \frac{1}{4}(\mathbf{L} \cdot \mathbf{H})^{-3}.$$

It is possibly the crudest approximation to Cook-Torrance there is, but an effective one no less. There will be no Fresnel color shift and no explicit interpolation to white (saturation is supposed to take care of that), so the model is linear in k_s, which is a computational advantage of its own. The model will underestimate the reflectance at grazing angles for low-index materials like water, and overestimate the reflectance for high-index materials like metals. For the typical range of di-electrics however, the estimation is just right.







The graphs above show the minimalist model with the $(\mathbf{L}\cdot\mathbf{H})^3$ denominator (red graph) against the combined effect of Schlick and the $(\mathbf{L}\cdot\mathbf{H})^2$ denominator of Kelemen and Szirmay-Kalos (blue graph), for specular reflectances of 0.02 (top), 0.05 (middle) and 0.8 (bottom), respectively. The graphs do not align perfectly, but the 'omph' is there. So, what about the instruction count?

MINIMALIST	F-SCHLICK WITH KELEMEN/SZIRMAY-KALOS
Multiply denominator by $\mathbf{L}\cdot\mathbf{H}$ for a third time. Done.	Calculate $1-\mathbf{L}\cdot\mathbf{H}$

	Raise that to the 4th or 5th power.
	Calculate $1 - k_s$.
	Lerp.

If you already have $(\mathbf{L} \cdot \mathbf{H})^2$, the simplified model just needs one additional multiplication. The combination of F-Schlick and Kelemen/Szirmay-Kalos needs 5 or 6 more instructions in a dependency chain from that point. It doesn't help if a Fresnel factor from the environment map already exists, since that is based on $\mathbf{N} \cdot \mathbf{V}$. In terms of bang for the buck, if there are many lights in a loop (and I had a forward renderer with up to 8 lights per pass), then I'd recommend the minimalist version.

[1] Dimitar Lazarov, "Physically-based lighting in Call of Duty: Black Ops" SIGGRAPH 2011 http://advances.realtimerendering.com/s2011/index.html
[2] Christian Schüler, "An Efficient and Physically Plausible Real-Time Shading Model." ShaderX 7, Chapter 2.5, pp. 175–187
[3] Fabian Giesen, "Phong Normalization Factor derivation" http://www.farbrausch.de/~fg/stuff/phong.pdf
[4] Kelemen and Szirmay-Kalos, "A Microfacet Based Coupled Specular-Matte BRDF Model with Importance Sampling", Eurographics 2001 http://www.fsz.bme.hu/~szirmay/scook.pdf
[5] Lafortune and Willems, "Using the modified Phong reflectance model for physically based rendering", Technical Report http://graphics.cs.kuleuven.be/publications/Phong/
[6] Yoshiharu Gotanda, "Practical Implementation of Physically-Based Shading Models at tri-Ace", SIGGRAPH 2010 http://renderwonk.com/publications/s2010-shading-course/
[7] Robert Cook and Kenneth Torrance, "A reflectance model for computer graphics" http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.83.7263
[8] Robert Lewis, "Making Shaders More Physically Plausible", Computer Graphics Forum, vol. 13, no. 2 (June, 1994) http://users.tricity.wsu.edu/~bobl/personal/mypubs/1993_plausible.pdf
[9] Akienne-Möller, Haines and Hoffmann, "Real-Time Rendering" book http://www.realtimerendering.com/
[10] Pharr, Humphreys, "Physically-Based Rendering" book http://www.pbrt.org/
[11] Nathaniel Hoffmann, "Crafting Physically Motivated Shading Models for Game Development", SIGGRAPH 2010 http://renderwonk.com/publications/s2010-shading-course/

This entry was posted in **Uncategorized** and tagged **math**, **physically based shading**, **spellforce**, **velvet assassin** by **christian**. Bookmark the **permalink [http://www.thetenthplanet.de/archives/255]** .

33 THOUGHTS ON "THE BLINN-PHONG NORMALIZATION ZOO"

GaameGame
on **2011-09-09 at 18:03** said:

Can you help me understand the derivation of irradiance under the section “Two Pi Or Not Two Pi”? It seems like if we decrease r_0 towards 0, then the equation for irradiance E also goes to 0, and the factor of π becomes sort of pointless. The irradiance equation would make sense to me if $\frac{r_0}{r}$ does not approach 0, and pi’s cancel out, but we still need to multiply the reflectance with $(\frac{r_0}{r})^2$ when evaluating the rendering equation.

My understanding when evaluating irradiance from a single point light is that it’s 0 everywhere else and 1 for the dw coming from the point light so the irradiance integral can be done by directly evaluating the incoming radiance times the cosine factor. There would be no pi’s to cancel and we would still need to divide by pi for a single outgoing dw.

Please correct me if I missed something, I want to understand this as it has always confused me. Thanks.

christian
on **2011-09-10 at 02:17** said:

Hallo Gaame,
no problem. Think of the light source as being a distant star. It has some radius r_0 and some surface temperature, giving the surface a specific radiance L. If you reduce the radius, and therefore the surface area, the radiance of the surface must increase (the star must get hotter) for that the light source keeps its intensity! In the limit of a point light, you would have zero r_0 and infinite L, but the product, the intensity, and therefore the irradiance received, stays finite. It is true that in the end, a point light is evaluated as a directional delta function, but you must not forget π, as it will stay there when you do the limit.

GaameGame
on **2011-09-18 at 16:51** said:

It makes a lot more sense now, but I am still slightly confused about the pi. Let me see if I understood this correctly. Assuming the light source is a small distant sphere as stated originally and inside the upper hemispherical range of the receiver, the light source is actually emitting the radiant flux from its part of the hemisphere facing the receiving point. So when calculating the radiant flux coming from the light source seen by the receiver, the pi comes from integrating the light source’s hemisphere? Or is the pi actually coming from integrating the receiver’s hemisphere?

Thanks again for replying.

christian
on **2011-09-18 at 21:24** said:

Neither of both, the example was derived from the solid angle of the light source as seen in the hemisphere of the irradiant plane. It starts with the integral

$$E = \int_{\phi=0}^{2\pi} \int_{\theta=0}^{\beta} L \cos\theta \sin\theta \, d\theta \, d\phi,$$

which is integrating L over the cap of a sphere with half-angle β, times the cosine factor. For this problem, the plane for the irradiant flux is conveniently set to be normal to the light direction, so the spherical cap is centered around $\theta = 0$. (The $\mathbf{N} \cdot \mathbf{L}$ later converts this to the irradiance received by the surface itself). The solution of the integral is

$$E = \pi L \sin^2\beta.$$

By trigonometry, $\sin\beta = \frac{r_0}{r}$, which arrives at the formula stated in the article,

$$E = \pi L \left(\frac{r_0}{r}\right)^2.$$

I'm now turning this reply more into a footnote for the article. Let's go all the way and reduce the expression to the one that will appear in the shader. The radiance L' of the shaded surface (aka. pixel color) is the BRDF times $\mathbf{N} \cdot \mathbf{L}$ times irradiance E. Assuming, without loss of generality, a constant Lambertian BRDF, $f_r = \frac{k_d}{\pi}$, we have

$$L' = \frac{k_d}{\pi} (\mathbf{N} \cdot \mathbf{L}) \, E$$

for the pixel color. The π from the BRDF cancels with the π from the irradiance, so the radiance-to-radiance relationship does not involve π:

$$L' = k_d \, (\mathbf{N} \cdot \mathbf{L}) \, L \left(\frac{r_0}{r}\right)^2.$$

When the light source is now shrunk to an idealized point, we can define a constant, call it I', that is the product of L and r_0^2, and we can recreate the usual formula for a diffuse light

$$L' = k_d \, (\mathbf{N} \cdot \mathbf{L}) \, \frac{I'}{r^2}.$$

GaameGame
on **2011-09-19 at 00:40** said:

This makes perfect sense now! Calculating irradiance in the plane normal to the light source makes a lot of sense, it is the piece that I was missing. I want to make sure I got one last small detail correctly: $\sin\beta \simeq \beta \simeq \frac{r_0}{r}$ for large r?

Thanks for your patience. Much appreciated.

> christian
> on **2011-09-19 at 00:52** said:
>
> There is not even a need for an approximation, $\sin\beta = \frac{r_0}{r}$ is always true. Just draw a right triangle ABC with A the light source center, B the observer and C a point tangent on the light source surface (it's a sphere), with angle β in B!
>
> PS, I hope you don't mind I prettied up your comment a little bit. LaTeX code for inline math via \$…\$ is enabled blog-wide.

GaameGame
on **2011-09-19 at 02:05** said:

I am feeling embarrassed to even ask this, but isn't $\frac{r_0}{r} = \tan\beta$ and $\frac{r_0}{r} = \sin\beta$ only for large r? Thanks!

P.S. Feel free to prettify my comments.

> GaameGame
> on **2011-09-19 at 06:39** said:
>
> Nevermind got it.

Earl Ashfield
on **2011-11-22 at 14:25** said:

Your site is actually outstanding. Thank you for that.

Pingback: Readings on physically based rendering | Light is beautiful

Motohaugh
on **2012-01-22 at 21:36** said:

Hello! Just want to say thank you for this interesting article! =) Peace, Joy.

Joyce
on **2012-01-26 at 12:35** said:

Like the blog

Steve Marton
on **2012-09-10 at 21:08** said:

I recently started toying with PBS, and right away got annoyed by artifacts in transitions between materials. Here's an example from your game:

http://i50.tinypic.com/2hnqjia.jpg

I circled one of the instances of the artifact I'm talking about: bright specular fringes in transitions between rough and smooth materials. It's unavoidable due to texture filtering, anti-aliasing in texture authoring, etc. The issue is that you end up with an unrealistic "semi-rough" value in the transition region, which picks up a broader and brighter specular highlight than the smooth and rough neighboring surfaces, respectively. (BTW, I tried both "real" Cook-Torrance and your "minimalist" approximation and got similar artifacts).

There's a similar issue transitioning between metal and dielectric (ex: rust on metal), where you get dark fringing if you try to reduce the diffuse contribution on the metal all the way to 0. Again, this is an unrealistic situation, because in real life you don't have a "semi-metal" in between the metal and the rust.

The root issue is that filtering introduces unrealistic in-between BRDF attributes.

The more correct solution to both of these issues would be to evaluate the BRDFs for the two materials separately, and blend the results, but that is prohibitive in terms of resources and likely very inconvenient for artists (manage separate texture sets and blend maps). The art workflow could maybe be worked around with extensive specialized paint tools, and the extra resource requirement might be acceptable on dx11 hardware, but neither sounds appealing.

So I'll throw the question out there for people who have had production experience with PBS. How did you get around texture filtering related artifacts?

> christian
> on **2012-09-12 at 16:43** said:
>
> Hallo Steve
> what you mention is indeed an issue, and I too do not have a solution for it. For our game we just accepted it as is and the artists try to work around the issue. Other people have this issue too. In the sildes from Call Of Duty, you can see on the wet vs dry parts on the roads the exact same issue. As it currently stands, the negatives are outweighed by the fact that a large variety of materials is batchable into a single texture.

Steve Marton
on **2012-09-13 at 02:15** said:

I see. Were you able to achieve a decent look for something as simple as rust on metal?

> christian
> on **2012-09-16 at 14:09** said:
>
> I think yes, at least I can't remember the artists complain. Any artifacts that appearead at the transitions of glossiness (for instance, very pronounced at the borders of wet puddles) seem to be accepted as a fact of life.

Pingback: Velvet Assassin is on Mac | The Tenth Planet

Ben Klumaster

on **2013-04-05 at 12:04** said:

One thing is confusing me here – you describe the RDF as the probability that a photon from the given direction will reflect towards the camera, but the normalisation factor for high gloss values seems to go above 1. Is there another factor that cancels this out?

> christian
> on **2013-04-07 at 17:44** said:
>
> The RDF (and the NDF) itself is not a probability but a *distribution*, aka probability density. To get a probability you need to integrate over a region. The normalization takes care that this integral is at most 1.

Jens
on **2013-04-24 at 22:04** said:

Do you have some references, guidelines or tutorials about how you factored the image into diffuse, specular and glossiness textures? Are you using any custom tools or plugins?

> christian
> on **2013-04-24 at 22:23** said:
>
> Hallo Jens,
> besides what I wrote in the article, there is no magic here. I also don't know of any resource that would specifically teach about this topic. One idea is to always take photos when the sky is overcast, so there is as little lighting as possible in the photos to begin with. Another thing is that if you have factored one image (for instance, you think you have a good specular map) then you can try to *linearly* subtract that from the photo to get the diffuse map. Linearly means here, that the subtraction is done in linear lighting space, not in RGB space. Hope this helps!

GaameGame
on **2013-09-23 at 19:42** said:

Sorry for resurrecting an old discussion, but I am recently coming back to the math and realized that there is one thing I am still missing during the irradiance E substitution.

Based on your derived equations, plugging E into L':

$$L' = \frac{k_d}{\pi}(N \cdot L)\pi L \left(\frac{r_0}{r}\right)^2$$

cancelling out π, we have

$$L' = k_d(N \cdot L)L \left(\frac{r_0}{r}\right)^2$$

The final equation is

$$L' = k_d(N \cdot L)\frac{I}{r^2}$$

which means

$$I = Lr_0^2$$

but how is this final equation derived? I can understand that
$E = \frac{I}{r^2}$, and $L = \frac{I}{A}$, but isn't
$Lr_0^2 = \frac{dI}{dw}$ (by substituting $dw = \frac{dA}{r^2}$)?
Please help.

christian
on **2013-09-28 at 15:23** said:

Hi Gaame,
welcome back, that's a long time; I had to reread my own posts to get up to the context. So that's correct, the radiant intensity appears as the product of L and r_0^2 *in the limit of a point light*. Remember that when the light source is shrunk to a point, we need to make it infinitely bright. So r_0 goes to zero, and L goes to infinity, but their product approaches the finite value I. It would happen in real life if a distant star is smaller than the resolution of the camera. Then we can only measure I but not L, since the solid angle of a camera pixel is known but not the actual surface area of the light source.

GaameGame
on **2013-10-11 at 03:43** said:

Hi Christian,

Thanks for the reply. I have been banging my head trying to understand/prove that $Lr_0^2 = I$ without much success. I can agree that

$L r_0^2$ is a finite value as $r_0 \to 0$, but how does it become the radiant intensity? Can you help me figure this out? Thanks!

> christian
> on **2013-10-14 at 17:23** said:
>
> Simple, just do a dimensional analysis. The units of radiant intensity is Watts per steradian (Wsr^{-1}); while the units of radiance is Watts per meter squared and steradian ($\mathrm{Wm}^{-2}\mathrm{sr}^{-1}$). Multiplying L with r^2 (or dividing I by r^2) will do the conversion.

Pingback: Readings on Physically Based Rendering | Interplay of Light

christian
on **2014-03-19 at 23:02** said:

I edited some wordings in the article and also added a paragraph that explains the decomposition of the Minimalist Cook Torrance formula into the D, F, V factors.

Pingback: Highlights from GDC 2014 presentations | The Tenth Planet

Pingback: Elite Dangerous: Impressions of Deep Space Rendering | The Tenth Planet

IRYOKU
on **2015-01-09 at 15:16** said:

Hi Christian, thanks for the great post and blog. I found it very useful.

I've a doubt with regards to the pi removal, I'd appreciate if you could help me with it.

From this equation:

$$L' = \frac{k_d}{\pi} (\mathbf{N} \cdot \mathbf{L}) \, E$$

If I just plug the point light irradiance equation, as found in various other sources:
E = I / r^2

Then the pi doesn’t cancel out. Why is that happening, and why it didn’t happened in you original derivation?

Following other references:
http://renderwonk.com/publications/s2010-shading-course/hoffman/s2010_physically_based_shading_hoffman_a_notes.pdf
Page 18

It is suggested that this pi disappears as a convenient way to describe point light sources in such a way that the color of light matches the color of a surface lit by it, when the normal and the light direction matches.

From what I can understand, the pi naturally disappears for area lights, as you showed in your derivation, but it looks to me that for point lights you actually need add a “times pi” correction (for more intuitive edition of light values).

If you can comment on this, it would be pretty much appreciated 🙂

christian
on **2015-01-15 at 15:41** said:

Hi Iryoku,
the reason is because the π is also in the formula for the irradiance, I.
That’s why I have shown the example of the ‘disk shrinking to a point’.
I have also corrected a slightly misleading wording, in my comment post from 2011-09, so now it should be clear.

The formula for E for a disk shaped light source was derived as

$$E = \pi L \left(\frac{r_0}{r}\right)^2 .$$

In this framework, $I = \pi L r_0^2$, so there is your π.

Pingback: Energy Conservation |

Pingback: Physically Based Shading in Games | A Graphics guy's note