Dimitar Lazarov, Lead Graphics Engineer, Treyarch

Introduction

We started pursuing Physically Based Lighting during the development of Call of Duty: Black Ops. The details were presented at SIGGRAPH 2011 as part of the Advances in Real-Time Rendering course [11]. Here we'll give a brief recap of the important bits relevant for this course and then describe the improvements we did during the development of Call of Duty: Black Ops II.

Summary of Physically Based Shading in Call of Duty: Black Ops

For the diffuse response we used the classic Lambertian BRDF. This talk will be all about the specular response.

We calculate our specular response out of two parts: direct and indirect. The direct component is calculated in the pixel shader from analytical sources (point lights), while the indirect component is reconstructed from environment maps, rendered offline at artist-selected locations.

Direct Specular (Analytical light)

We used a microfacet BRDF based on Cook-Torrance[2][3]:

$$BRDF(l,v,h) = \frac{D(h)F(l,h)G(l,v,h)}{4(n\cdot l)(n\cdot v)}$$

Here *D* is the *normal distribution function* (NDF), *F* is the *reflectance* function and *G* is the *shadowing-masking* (or geometry) function.

We factored the BRDF into three components, such that it's easier to swap different formulations and approximations.

Since we're using the BRDF with point lights, we need to multiply it by π[1]. It's convenient to group this and the division by 4 together as D_{pl}:

$$D_{pl}(h) = \frac{\pi}{4} D(h)$$

We also grouped G with the divide by $(n\cdot l)(n\cdot v)$. This is often called the *visibility* function – *V(l,v,h)*.

[1] See Naty Hoffman's notes[6] for a derivation of the punctual light equation.

$$V(l,v,h) = \frac{G(l,v,h)}{(n \cdot l)(n \cdot v)}$$

All combined:

$$BRDF_{pl}(l,v,h) = D_{pl}(h)F(l,h)V(l,v,h)$$

We used Blinn-Phong as our distribution function:

$$D(h) = \frac{\alpha + 2}{2\pi}(n \cdot h)^{\alpha}$$

Which for D_{pl} gives:

$$D_{pl}(h) = \frac{\alpha + 2}{8}(n \cdot h)^{\alpha}$$

Here, α is the specular power parameter.

We didn't paint specular powers directly into our textures but instead we painted gloss values, *g*, a log base-8192 encoding of specular power:

$$\alpha = 8192^{g}$$

We used the Schlick-Fresnel approximation as our reflectance function [12]:

$$F(l,h) = rf_0 + (1 - rf_0)(1 - h \cdot l)^5$$

Here rf_0 is the base reflectance, or *specular color* value. In most materials this was painted directly into textures; however we had a special case dielectric material that used an implicit value of 0.04 linear.

We used the Schlick-Smith approximation as our visibility function [13]:

$$k = \frac{2}{\sqrt{\pi(\alpha + 2)}}$$

$$V(l,v,h) = \frac{1}{((n \cdot l)(1 - k) + k)((n \cdot v)(1 - k) + k)}$$

Despite the extra cost we found that it was very important to have a high quality visibility function for natural-looking specular response.

Indirect Specular (Environment map)

Environment map normalization

Traditionally, when dealing with environment maps, we constantly struggled between the memory cost and the accuracy of reflections for a given location. To alleviate these problems we came up with a technique that let us better "fit" an environment map to the surroundings where it's being applied. We called this technique *environment map normalization*.

First, offline, we "normalized" the environment map image: we divided the radiance at each texel by the average irradiance at the place where the environment map was captured. Then, in the pixel shader, we sampled the environment map and scaled it by the average irradiance calculated[2] at that pixel.

Environment map filtering

We used *CubeMapGen* [1] to pre-filter the environment map's mip-map chain. For each mip level we used angular Gaussian filter with increasing size, based on the reference values provided in the CubeMapGen documentation. In the pixel shader we manually selected the mip level as a function of the gloss value:

```
texCUBElod( uv, float4( R, nMips - gloss * nMips ) )
```

Environment BRDF

In the pixel shader we multiplied the pre-filtered environment map sample with a Fresnel factor. Initially we used the Schlick-Fresnel $(n \cdot v)$based function, but it quickly became obvious this was not enough to match the specular response we were getting from our analytical sources. We derived a new empirical version, which included a gloss-based term, that effectively had the role of a shadowing-masking function.

$$F(l,v) = rf_0 + (1 - rf_0)\frac{(1 - n \cdot v)^5}{4 - 3g}$$

At the time we continued to refer to this as a Fresnel function, but in this talk we'll use the term *Environment BRDF*[3], since it's a better description of its purpose. We'll discuss this in more detail in the following sections.

---

[2] Usually this is easily derived from data that is already present in the lightmap or any other form of light baking.

[3] In other published materials this has also been termed an *Ambient BRDF* [4][5].

Getting More Physical in Call of Duty: Black Ops II

Direct Specular

We were pretty happy with the quality of the direct specular, so for Black Ops II we primarily focused on performance improvements.

We tried to optimize the BRDF parts in isolation and combined. In the end, it appeared individual approximations gave us the best tradeoff between accuracy and speed.

New Fresnel function approximation
We used *Mathematica* to plot and compare candidate curves, and searched for cheap approximations via trial and error[4].
In the end, we managed to optimize the Fresnel function with this approximation[5]:

$$F_{opt}(l,h) = rf_0 + (1 - rf_0)2^{-10(h \cdot l)}$$

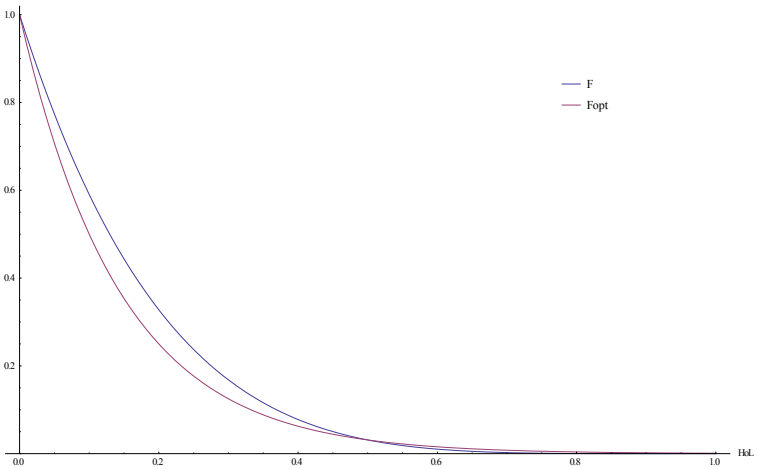


Figure 1 – 2D plot comparing F and F_{opt}

We dialed it for minimum error at small angles, since the vast majority of pixels on screen tend to fall into this category.

New Visibility function approximation

This part was done outside of *Mathematica*. The primary idea was to look for an approximation similar to the Kelemen-Szirmay-Kalos [8] approximation to the Cook-Torrance Shadowing-Masking function; replacing a complex function with a simpler one, which is dependent only on $(v \cdot h)$ or $(l \cdot h)$ – the two are interchangeable. Since the

[4] Refer to the accompanying Mathematica notebook for more details.
[5] Note that F_{opt} saves 1 instruction on current generation GPUs but is not necessarily an optimization on newer GPUs.

original function is multi-dimensional, it's hard to visualize it in *Mathematica*. Instead we used a shader test framework and tried various functions of $(v \cdot h)$ vs. the current expression, on a range of geometry with representative Black Ops II normal and gloss maps. In the end, we found that the following function was a pretty good match over various gloss values:

$$k = \min(1.0, g + 0.545)$$

$$V_{opt}(v,h) = \frac{1}{k(v \cdot h)^2 + (1-k)}$$

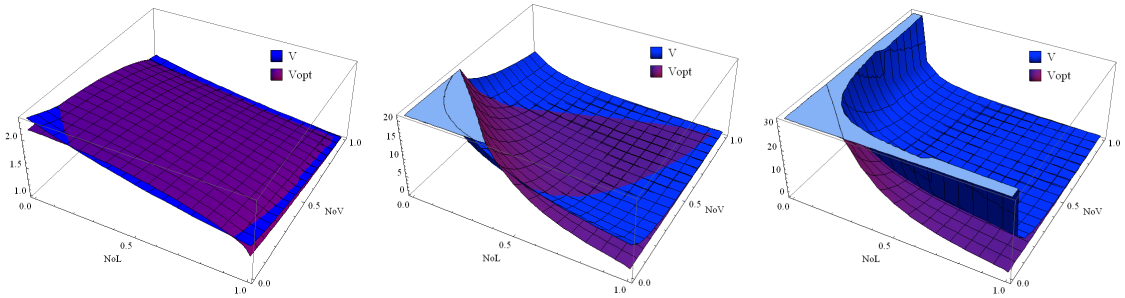


Figure 2 - 3D plot comparing V and V_{opt} with gloss values of 0.0, 0.5 and 1.0

The approximation diverges from the original at large angles, although these cases tend to be rare. In game there was little visual difference and the speed increase was significant, so we decided to use it in the shipping game.

Indirect Specular

All three components of the indirect specular had visual problems that our content creators often complained about. It's fair to say we spent the majority of our effort during the project getting our indirect specular to be a closer match to the direct specular.

Improved environment map normalization

The normalization appeared to have certain inconsistencies in the reflection amount over meshes employing different types of light baking. In particular, lightmapped surfaces seemed to have problems recovering the correct intensity compared to objects that used light probes.

Analyzing the problem revealed that the denormalization step yields incorrect results with light encodings that only capture a hemisphere of irradiance. For example, a reflection probe placed above the ground would see "sky" and "ground", while the ground surface itself would only see "sky". Hence when the ground surface attempts to denormalize the reflection, it has no way to compensate for the "ground" part of the irradiance.

One solution would be to perform the normalization using irradiance – for example against the geometric normal of a surface – instead of average irradiance. By definition, the irradiance covers a hemisphere and as such would solve the problem case described above. However, since a given environment map might be applied to surfaces of different orientations (and hemispheres respectively), we can't bake the normalization into the image anymore and there would be some cost to performing this at run-time.

Fortunately, we came up with a relatively cheap approach that accomplishes this.
First, we capture the diffuse irradiance at the same location where the environment map was captured. We encode it as 3rd-order Spherical Harmonics, or "SH9" (9 terms), which in practice we store as a tint plus 9 scalar SH coefficients for efficiency[6].
During rendering, we set up the SH coefficients as shader constants. In the vertex shader we evaluate the SH against the vertex normal (which gives us the environment map's irradiance in that direction) and then pass it down to the pixel shader as the normalization factor. Finally, the pixel shader samples the environment map, applies the normalization factor, and immediately rescales it by the per-pixel irradiance as before.

In pseudo code:

Old method:
Offline:

```
env_sh9 = capture_sh9(env_pos);
env_average_irradiance = get_sh_coeff(env_sh9, 0);
for_each(texel in environment map)
      texel /= env_average_irradiance;
```

PS:

```
env_color = sample(env_map) * pixel_average_irradiance;
```

New method:
Offline:

```
env_sh9 = capture_sh9(env_pos);
```

[6] The majority of our problem cases were related to incorrect intensity and less so to incorrect color.

VS:

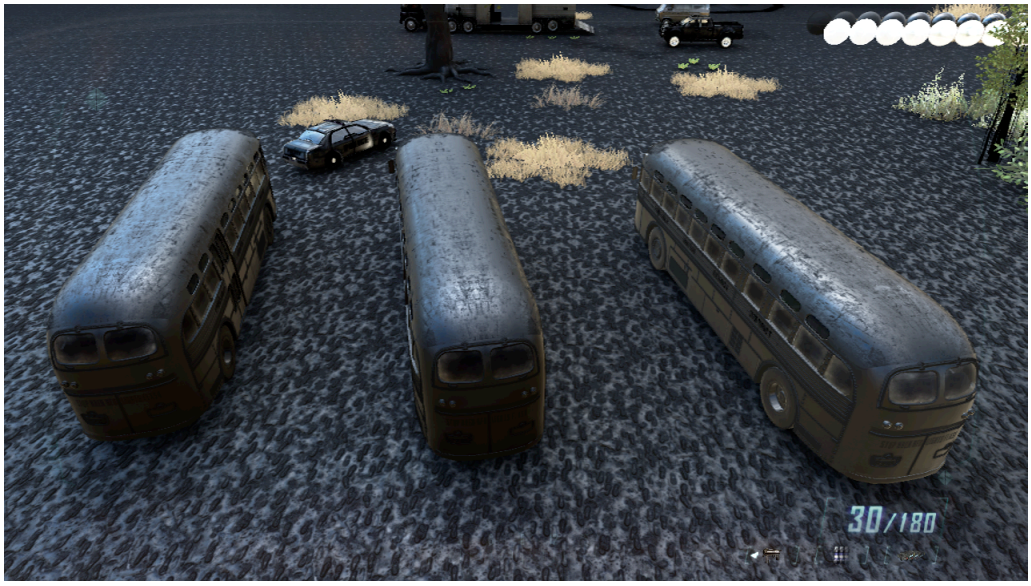
```
env_irradiance = eval_sh(env_sh9, vertex_normal);
```

PS:

```
env_color = sample(env_map) / env_irradiance * pixel_ irradiance;
```

The new method heavily minimized the visual problems we were seeing. There were still very subtle discrepancies due to the approximate nature of the SH encoding, and in particular, the tinted scalar SH we had to use for performance reasons.

The new method added about 8 instructions to the vertex shader, 1 interpolator and 1 instruction to the pixel shader. However, we didn't see a measurable drop in frame rate when we deployed it in the actual game[7].



[7] Our game is heavily pixel shader bound so this was not entirely surprising.

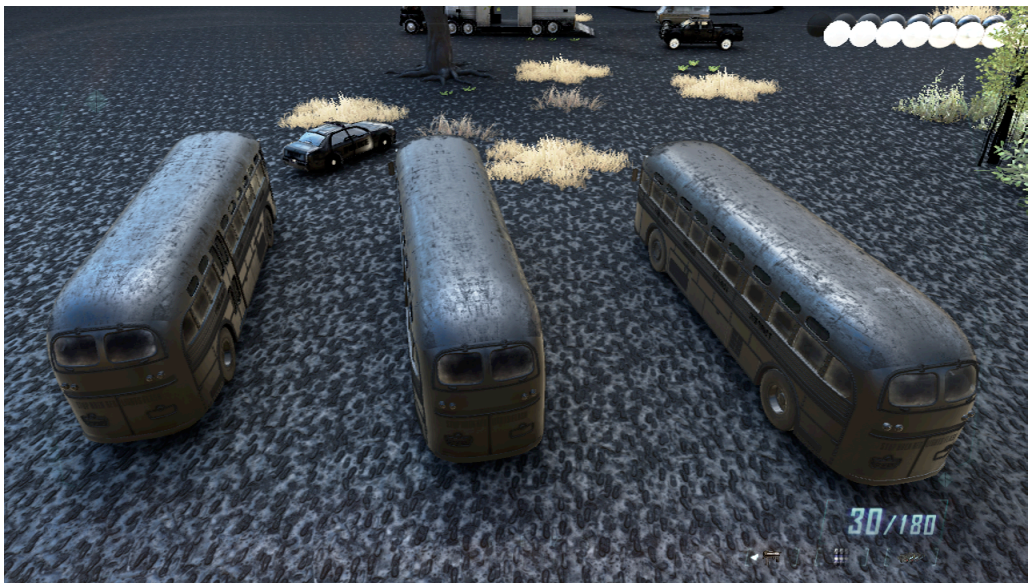


Figure 3 – A pair of in-engine screenshots comparing the old method (above) and the new method (below). The image shows just the environment map's specular contribution overexposed for easier comparison. The three buses use three different light baking methods – lightmap (left), vertex bake (middle), light probe (right). Notice how the new method achieves much closer specular response between the three.

Improved environment map filtering

When accessing the environment map's pre-filtered mip chain, we use a linear function of our gloss value. However, the Gaussian filter we were previously using on Black Ops was not blurring exactly the way our gloss would "blur" the point lights. Fortunately, the CubeMapGen tool was open-sourced around the time we started looking into improving our filtering. We extended the CubeMapGen filtering framework with a new cosine power filter[8].

For each mip level we calculated the corresponding gloss and specular power, which we then used to drive the cosine power filter. Note that this effectively ties the maximum environment map resolution with the maximum specular power.

On Black Ops we used 256x256 resolution environment maps, but for Black Ops II we realized that we could drop the resolution to 128x128, since our maximum specular power of 8192 was over-blurring the 256x256 images, so we were gaining nothing from the higher resolution.

It is also important to note that we need to convert the Blinn-Phong specular powers to Phong specular powers. This is necessary since we perform the environment map lookup via a reflection vector, which follows the Phong shading model. Dividing the Blinn-Phong specular power by 4 appears to be a good approximation to a Phong highlight [10].

[8] Similar work was done independently by Sébastien Lagarde, who has since published the modifications [9].

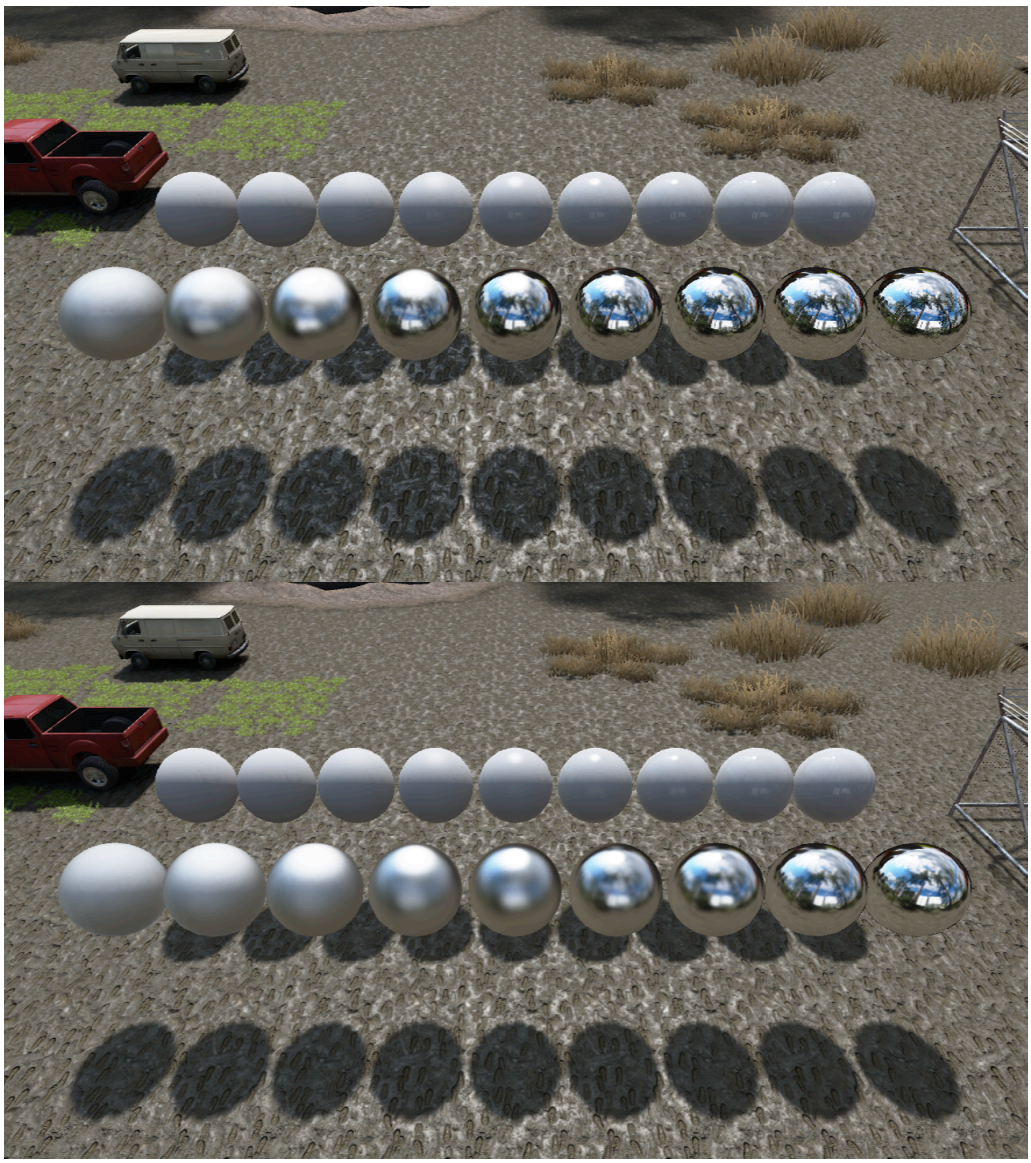


Figure 4 – A pair of in-engine screenshots comparing the old method (above) with the new method (below). The back row of spheres has $rf_0 = 0.04$, while the front row of spheres has $rf_0 = 1$, with gloss values linearly spanning the range between the spheres. Notice how the new method blurs much more linearly across the gloss range and it's a closer match to the sun's hot-spot "blur".

Improved Environment BRDF

The empirical Environment BRDF we used in Black Ops was only a rough estimate. Consequently, our content creators often complained about inconsistent behavior between the direct and indirect specular for low gloss values and oblique angles. We knew this was something we had to address, so early on in the development of Black Ops II we set about deriving a better formulation.

First, let's start with the integral that we're trying to approximate:

$$\int Env(l)BRDF_{env}(l,v,h)\cos(\omega)d\omega$$

Where:

$$D_{env}(h) = \frac{\alpha+2}{8\pi}(n\cdot h)^{\alpha}$$
$$BRDF_{env}(l,v,h) = D_{env}(h)F(l,h)V(l,v,h)$$

Note that the distribution function has a π in the denominator, since we're using the BRDF with environment map radiance values (as opposed to point light values).

One way to compute this integral would be to importance-sample the environment map in the pixel shader and evaluate the BRDF for each sample. This was quite an expensive proposition for our game, so instead we made an approximation by splitting the integral into two parts, with the idea that it would be easier to calculate them separately:

$$\left(4\int Env(l)D_{env}(h)\cos(\omega)d\omega\right)\left(\int BRDF_{env}(l,v,h)\cos(\omega)d\omega\right)$$

The first part represents environment map filtering (blurring), and the second part is the *Environment BRDF*.

It's important to note that this split is exactly correct for the case where the environment map is a constant color. As is typical outdoors, an environment map containing the sky is often low frequency – particularly on a clear or overcast day – so this approximation is not entirely off-base.

The environment map filtering is further approximated via the offline pre-filtering described in the previous section. For the Environment BRDF, we attempted to find cheap analytical expressions for common cases that could be evaluated directly in the pixel shader. Elsewhere in this course, Brian Karis from Epic Games, will show an alternative approach to solving this integral [7].

Our approach was to calculate "ground truth" curves via numerical integration in *Mathematica*.

Based on examination of the Schlick-Fresnel factor in our BRDF, the full integral can be calculated as a linear interpolation between the integrals for base reflectance of 0.0 and base reflectance of 1.0, weighted with the actual value of the base reflectance.

$$F(l,h) = rf_0 + (1-rf_0)(1-h\cdot l)^5$$

$$\int BRDF_{env}\cos(\omega)d\omega = rf_0\int D_{env}V\cos(\omega)d\omega + (1-rf_0)\int D_{env}V(1-h\cdot l)^5\cos(\omega)d\omega$$

Thus, we integrated our cosine-weighted BRDF for varying gloss values (specular powers) into two sets of "ground truth" curves, with base reflectance of 0.0 and 1.0 respectively.

For the exact *Mathematica* expressions please take a look at the accompanying *Mathematica* notebook.

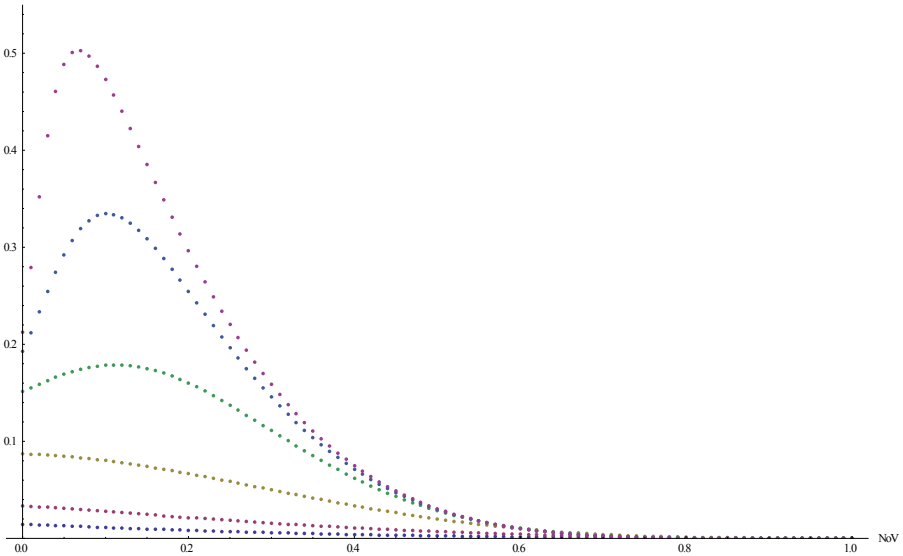


Figure 5 – 2D plot of "ground truth" curves $rf_0 = 0$ with gloss values of 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0

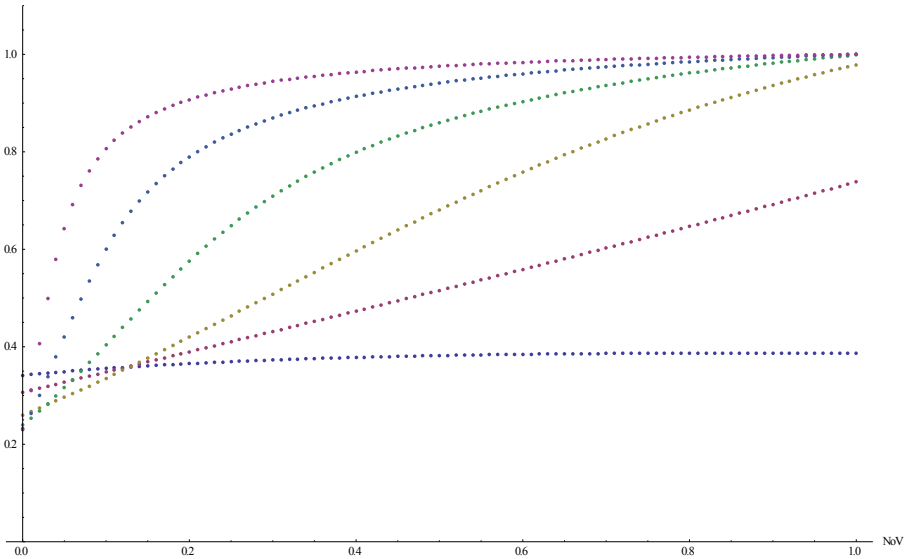


Figure 6 – 2D plot of "ground truth" curves $rf_0 = 1$ with gloss values of 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0

Once we had the "ground truth" curves we proceeded to fit analytical expressions that would match them as closely as possible. Finding these expressions was done mostly via trial and error. For some ideas on how to approach this, please refer to the accompanying *Mathematica* notebook.

First we focused on accurate approximations:

```
float a0( float g, float NoV )
{
	float t1 = 11.4 * pow( g, 3 ) + 0.1;
	float t2 = NoV + ( 0.1 - 0.09 * g );
	return (1 - exp( -t1 * t2 ) ) * 1.32 * exp2( -10.3 * NoV );
}

float a1( float g, gloat NoV )
{
	float t1 = max( 1.336 - 0.486 * g, 1);
	float t2 = 0.06 + 3.25 * g + 12.8 * pow( g, 3 );
	float t3 = NoV + min( 0.125 - 0.1 * g, 0.1 );
	return min( t1 - exp2( -t2 * t3 ), 1 );
}
```

Later we optimized the expressions by noting areas of the curves that had little visual impact in game:

```
float a0f( float g, float NoV )
{
	float t1 = 0.095 + g * ( 0.6 + 4.19 * g );
	float t2 = NoV + 0.025;
	return t1 * t2 * exp2( 1 - 14 * NoV );
}

float a1f( float g, float NoV )
{
	float t1 = 9.5 * g * NoV;
	return 0.4 + 0.6 * (1 - exp2( -t1 ) );
}
```

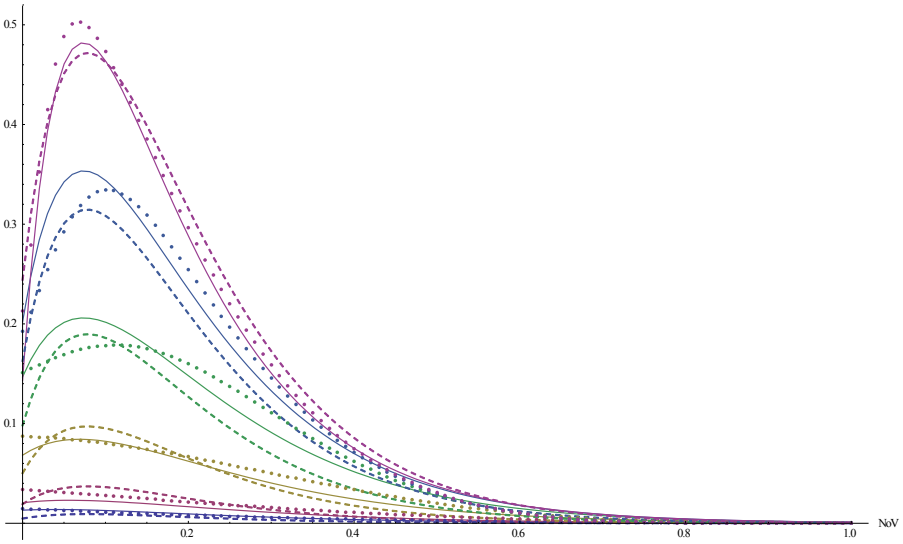


Figure 7 – 2D plot comparing the "ground truth" curves $rf_0 = 0$ (dotted) with the *a0* (solid) and *a0f* (dashed) approximations

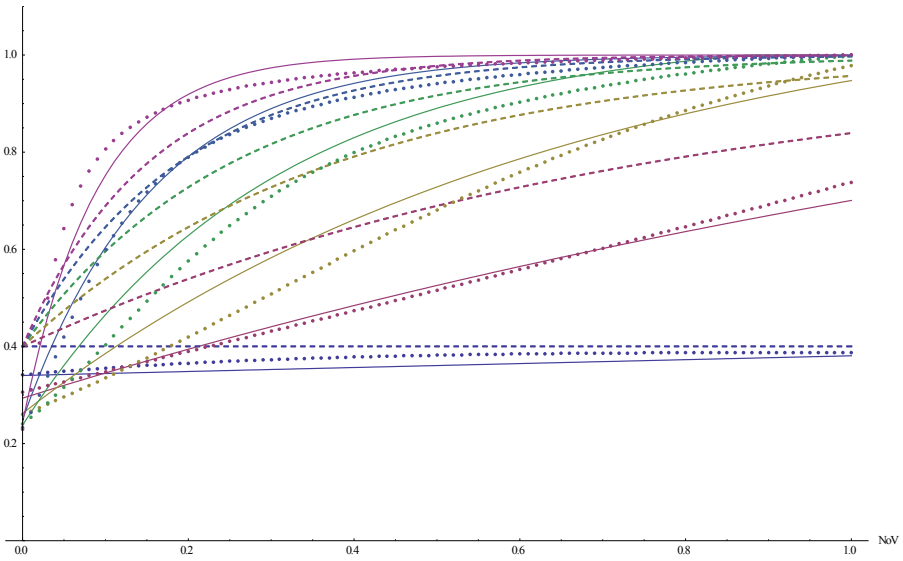


Figure 8 - 2D plot comparing the "ground truth" curves $rf_0 = 1$ (dotted) with the *a1* (solid) and *a1f* (dashed) approximations

The new Environment BRDF approximations solved a good percentage of the cases where our artists complained of the environment reflections being too bright, particularly for dielectric/low-gloss materials. However the approximations were still fairly expensive.

At this point, we decided to focus on the special case of $rf_0 = 0.04$ – a common value for many dielectric materials – especially since we have a higher-performance "simple" (dielectric-only) shader that can avoid the reflectance interpolation entirely.
Again we used *Mathematica* to calculate ground truth curves for the $rf_0 = 0.04$ case, with gloss values of 0.0, 0.5 and 1.0.

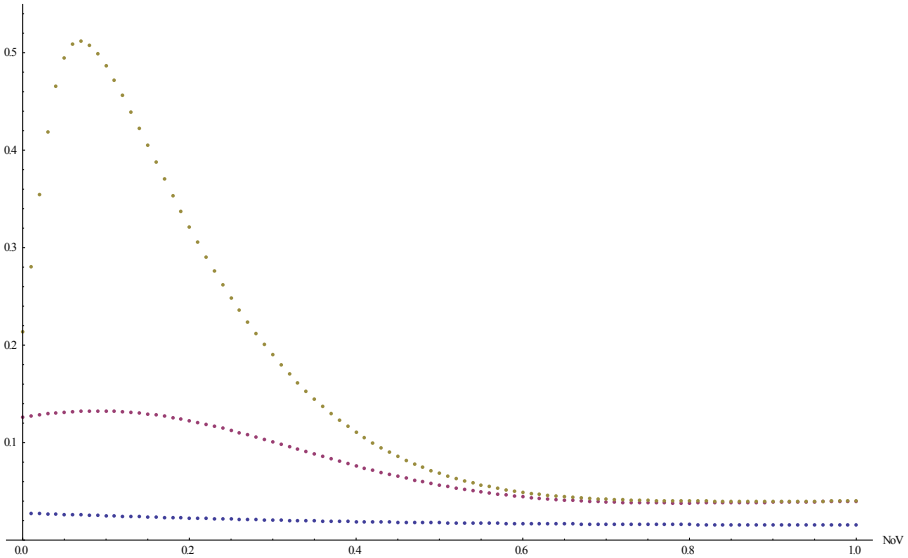


Figure 9 – 2D plot of the "ground truth" curves $rf_0 = 0.04$ with gloss values of 0.0, 0.5 and 1.0

And then we proceeded to find the closest and cheapest fit for this special case.

```
float a004( float g, float NoV )
{
	float t = min( 0.475 * g, exp2( -9.28 * NoV ) );
	return ( t + 0.0275 ) * g + 0.015;
}
```

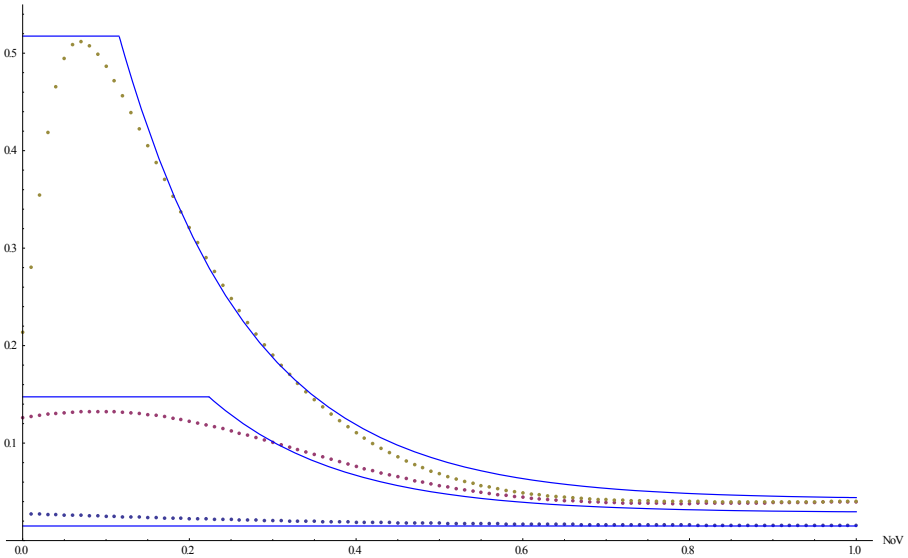


Figure 10 – 2D plot comparing the "ground truth" curves $rf_0 = 0.04$ (dotted) with the *a004n* approximation (solid)

For performance reasons we ended up using a very cheap expression for the $rf_0 = 1.0$ case. Metals are relatively rare in our game and the ones we had looked good even with this very coarse approximation:

```
float a1vf( float g )
{
	return 0.25 * g + 0.75;
}
```

Finally, we reconstructed the $rf_0 = 0$ case by extrapolating from the $rf_0 = 0.04$ case and the $rf_0 = 1.0$ case.

```
float a0r( float g, float NoV )
{
	return ( a004( g, NoV ) - a1vf( g ) * 0.04 ) / 0.96;
}
```

The final approximation:

```
float3 EnvironmentBRDF( float g, float NoV, float3 rf0 )
{
	float4 t = float4( 1/0.96, 0.475, (0.0275 - 0.25 * 0.04)/0.96, 0.25 );
	t *= float4( g, g, g, g );
	t += float4( 0, 0, (0.015 - 0.75 * 0.04)/0.96, 0.75 );
	float a0 = t.x * min( t.y, exp2( -9.28 * NoV ) ) + t.z;
	float a1 = t.w;
	return saturate( a0 + rf0 * ( a1 - a0 ) );
}
```

Figure 11 – A pair of in-engine screenshots comparing the old (above) and new (below) method. Notice with the old method the unnatural amount of environment reflection on the shadowed side of the red truck. The new method provides a much more natural looking specular response.

It's fair to say the final Environment BRDF approximation gave us pretty satisfactory visual results given our tight performance budgets. The dielectric special case was only 5 pixel shader instructions, while the full approximation was 7 instructions. This compared favorably to the Black Ops empirical "Fresnel" which was 8 instructions.

Acknowledgements

I'd like to thank Naty Hoffman for his invaluable contribution to the physically based shading in our game and for helping me understand the intricacies of the math behind it; Marc Olano for answering my numerous questions along the way; Jorge Jimenez and Sebastien Lagarde for the insightful and thought-provoking email conversations; Stephen Hill and Stephen McAuley for reviewing and helping me prepare this talk. And last but not least, I'd like to thank the team at Treyarch for making a fantastic game, without which none of this would've been possible.

References

[1] AMD, CubeMapGen: Cubemap Filtering and Mipchain Generation Tool. http://developer.amd.com/resources/archive/archived-tools/gpu-tools-archive/cubemapgen/

[2] Cook, Robert L., and Kenneth E. Torrance, "A Reectance Model for Computer Graphics", Computer Graphics (SIGGRAPH '81 Proceedings), pp. 307-316, July 1981.

[3] Cook, Robert L., and Kenneth E. Torrance, "A Reectance Model for Computer Graphics", ACM Transactions on Graphics, vol. 1, no. 1, pp. 7-24, January 1982. http://graphics.pixar.com/library/ReflectanceModel/

[4] Drobot, Michal, "Lighting Killzone: Shadow Fall", Digital Dragons, April 2013. http://www.guerrilla-games.com/publications/

[5] Gotanda, Yoshiharu, "Practical Implementation of Physically-Based Shading Models at tri-Ace", part of "Physically-Based Shading Models in Film and Game Product", SIGGRAPH 2010 Course Notes. http://renderwonk.com/publications/s2010-shading-course/

[6] Hoffman, Naty, "Background: Physics and Math of Shading", part of "Physically Based Shading in Theory and Practice", SIGGRAPH 2013 Course Notes. http://blog.selfshadow.com/publications/s2013-shading-course/

[7] Karis, Brian, "Real Shading in Unreal Engine 4", part of "Physically Based Shading in Theory and Practice", SIGGRAPH 2013 Course Notes. http://blog.selfshadow.com/publications/s2013-shading-course/

[8] Kelemen, Csaba, and Laszlo Szirmay-Kalos, "A Microfacet Based Coupled Specular-Matte BRDF Model with Importance Sampling," Eurographics 2001, short presentation, pp. 25-34, September 2001. http://www.fsz.bme.hu/~szirmay/scook_link.htm

[9] Lagarde, Sebastien, "AMD Cubemapgen for physically based rendering", September 2011. http://seblagarde.wordpress.com/2012/06/10/amd-cubemapgen-for-physically-based-rendering/

[10] Lagarde, Sebastien, "Relationship between Phong and Blinn lighting model", March 2012. http://seblagarde.wordpress.com/2012/03/29/relationship-between-phong-and-blinn-lighting-model/

[11] Lazarov, Dimitar, "Physically-based lighting in Call of Duty: Black Ops", part of "Advances in Real-Time Rendering in 3D Graphics and Games", SIGGRAPH 2011 Course Notes. http://advances.realtimerendering.com/s2011/

[12] Schlick, Christophe, "An Inexpensive BRDF Model for Physically based Rendering," Computer Graphics Forum, vol. 13, no. 3, Sept. 1994, pp. 149-162. http://dept-info.labri.u-bordeaux.fr/~schlick/DOC/eur2.html

[13] Akenine-Moller, Tomas, Eric Haines, and Naty Hoffman, "Real-Time Rendering, third edition", A K Peters Ltd., 2008, pp. 262