Material Advances in Call of Duty: WWII

Danny Chan, Sledgehammer Games / Activision Blizzard

1 INTRODUCTION

In this paper, we describe several improvements that were made to our lit surface shaders in *Call of Duty: WWII*. We detail a simple method to mipmap normal and gloss maps, by directly mapping gloss to average normal length of a distribution, bypassing an intermediate representation, like variance (Section 2). We show how rational functions can be a useful tool to approximate many functions that can't be represented analytically or efficiently (Section 3). We discuss how to auto-generate cavity maps for every surface that has a normal map, and show how to handle occlusion and indirect lighting in cavities represented by these textures (Section 4). Finally, we describe how to extend the Diffuse BRDF to model Lambertian microfacets with multiscattering (Section 5), while also showing how the environment split integral precalculation can be easily reused for energy conservation (Section 5.3).

These changes allowed us to achieve a richer ambient look than previously possible, with scenes rendered during "magic hour" (Figure 1).

Figure 1: An early version of Normandy rendered during "magic hour".

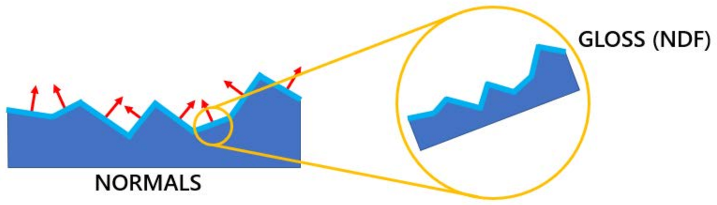

Figure 2: Normal maps and gloss maps represent geometric information at different scales.

1.1 New Gloss Parameterization

Before we dive in, we should point out a change to how we parameterize gloss in *Call of Duty: WWII*. Previously, on *Call of Duty: Advanced Warfare*, we used this parameterization of gloss [Cha15] for GGX [Wal+07]:

$$\alpha = \sqrt{\frac{2}{2 + 2^{16g}}}. \tag{1}$$

This is our new parameterization of gloss:

$$\alpha = \sqrt{\frac{2}{1 + 2^{18g}}}. \tag{2}$$

Notice we allow sharper specular highlights because of the change from 16 to 18. Importantly, we also allow α to reach 1 when gloss is 0, giving us the full range of roughness available with GGX. In other words, our gloss value now expresses a wider range – both rougher and smoother – than before.

2 SPECULAR ALIASING: MIPMAPPING NORMAL AND GLOSS

Normal maps and gloss maps can be thought of as representing geometric information at different scales. Gloss maps store microfacet normal distributions, while normal maps store macroscale normals (Figure 2). Normals become microfacet orientations when viewed from afar. Because of this, it makes sense to push normal map information down into gloss maps when generating mipmaps.

If we mipmap our normal and gloss map independently, what happens is normal maps tend towards flat surfaces as they mip, while gloss averages out. Imagine a very bumpy but highly glossy surface: as it mips into the distance, it turns more and more mirror-like. By pushing the variation in normals into our gloss maps, we can avoid this issue (Figure 3).

Many techniques have been proposed to combat specular aliasing (LEAN [OB10], CLEAN [Bak11], Normal Variance [Tok04], see [Hil11] for an overview). These techniques typically require more channels of information, but simplified variants exist that do not add any extra channels. For example, you can use [Tok04] to estimate variance and then use this to adjust an existing gloss channel.

Similar to these variants, we've taken a simpler approach that does not add any extra channels, but unlike previous approaches, is specialized for the GGX normal distribution function (NDF). In addition, our method filters normals and base gloss together, such that base gloss affects the weighting of our normal directions, similar

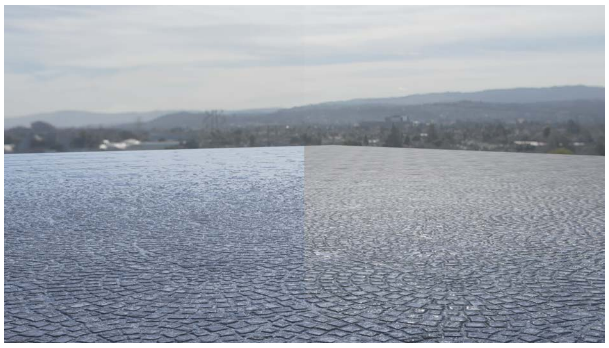

Figure 3: Left half is standard mipmapping of normal and gloss maps, while right half uses our technique for mipmapping normal and gloss.

to [Kar18]. This is different from the other methods, where base gloss is post-convolved with a distribution derived from the normals.

First, notice that averaging a collection of arbitrary normals in the hemisphere (that is, summing the normals and dividing by their count) results in a vector that may no longer be unit length. We call this the "average normal", or "shortened normal" ($\mathbf{n}_s$ in Equation 3 and Figure 4), following [Tok04].

$$\mathbf{n}_s = \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{n}_i . \tag{3}$$

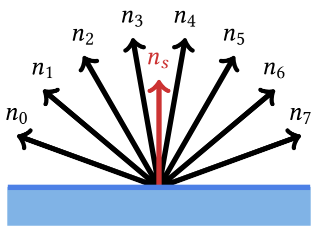

Figure 4: n_s is the average of n_0 through n_7.

If we randomly sample the GGX NDF, generating random microfacet normals following this distribution for specific gloss values [0, 255], and calculate the average normal for these distributions (Macrosurface Area model of this technique described in Section 2.1.3, corresponding pseudocode in Listing 6) then we have a table that relates gloss to "shortened normal" length (Listing 8).

2.1 Randomly Sampling the NDF

Why do we randomly sample the NDF? What probability density function (PDF) do we use to do so? Key to this is understanding what $D(\omega)$, our NDF, represents and how that relates to a normal map.

2.1.1 *NDF relates to microsurface area*

Given a microsurface M and points along that surface p_m with corresponding normal directions ω_m, we can define our $D(\omega)$ as a spatial equation using [Hei14], Equation 2:

$$D(\omega) = \int_M \delta_\omega(\omega_m(p_m))\, \mathrm{d}p_m, \tag{4}$$

where δ_ω is the Dirac-delta function.

From this, we can see $D(\omega)$ represents the area of the microsurface that points in direction ω. Therefore, you can think of $D(\omega)$ as related to the microsurface area of either the normals in our normal maps, or the normals randomly chosen from our NDF.

We can reason about how we calculate our shortened normal lengths mathematically. When accumulating normals for averaging, we can assign a weight to each normal according to the ratio of microsurface area of that normal to the total microsurface area.

Using [Hei14], Equation 5:

$$\text{microsurface area } = \int_M \mathrm{d}p_m = \int_\Omega D(\omega)\, \mathrm{d}\omega, \tag{5}$$

we can calculate our per-normal weight:

$$s_{\omega'} = \frac{D(\omega')}{\int_\Omega D(\omega)\, \mathrm{d}\omega}. \tag{6}$$

The shortened normal length of our distribution of normals (NDF) can be calculated using the integral of this weight times the z-component of the sampled direction. The x- and y-components will cancel out since samples are symmetric in these dimensions, so the z-component is all that is needed to calculate shortened normal length.

$$\begin{align} \left\|\mathbf{n}_{s,micro}\right\| &= \int_\Omega s_{\omega'}\omega'_z \mathrm{d}\omega', \tag{7}\\ &= \int_\Omega \frac{D(\omega')}{\int_\Omega D(\omega)\, \mathrm{d}\omega}\omega'_z\, \mathrm{d}\omega', \tag{8}\\ &= \frac{\int_\Omega D(\omega')\, \omega'_z\, \mathrm{d}\omega'}{\int_\Omega D(\omega)\, \mathrm{d}\omega}. \tag{9} \end{align}$$

Since we use z-up, and because the z-component of the sampled direction is equivalent to the cosine of its polar angle:

$$\omega'_z = \cos\theta', \tag{10}$$

and given the projected area of the microsurface onto the macrosurface is 1:

$$\int_\Omega D(\omega) \cos\theta\, \mathrm{d}\omega = 1, \tag{11}$$

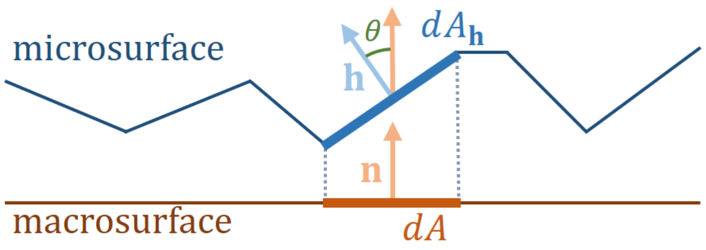

Figure 5: Microsurface facet area dA_h projects to macrosurface area dA.

our shortened normal length is

$$\left\|\mathbf{n}_{s,micro}\right\| = \frac{1}{\int_\Omega D(\omega)\,\mathrm{d}\omega}. \tag{12}$$

We can see that both shortened normal length and total microsurface area relate directly to our NDF. Interestingly, relating the NDF to total microsurface area provides an alternative to relating to shortened normal length. We call this the Microsurface Area model for calculating shortened normals.

2.1.2 Normal map as microsurface

If we imagine a normal map as a microsurface, then what does each texel of the normal map represent? We can think of each texel as a infinitesimal microfacet oriented in the normal direction with area:

$$dA_h = \frac{d_A}{\cos\theta}, \tag{13}$$

where d_A is the texel area associated with the macrosurface (Figure 5). This gives us a way to average normals of a normal map based on microsurface area (Algorithm 1) that relates with our calculation of shortened normal length in Equation 12.

Algorithm 1 Averaging normals by microsurface area

```
1: procedure CalcAverageNormalMicro
2:     $averageNormal \leftarrow (0, 0, 0)$
3:     $totalWeight \leftarrow 0$
4:     for all $n$ in $normals$ do
5:         $averageNormal \leftarrow averageNormal + \frac{n}{\cos\theta}$
6:         $totalWeight \leftarrow totalWeight + \frac{1}{\cos\theta}$
7:     $averageNormal \leftarrow \frac{averageNormal}{totalWeight}$
```

2.1.3 Normal map as macrosurface

We can also think of the normal map as a macrosurface where each texel represents a normal pertubation with equal weighting. If this is the case, we would use a PDF based on macrosurface-projected area:

$$\mathrm{p}_\omega(\omega) = D(\omega)\,\cos\theta, \tag{14}$$

and importance sample using this PDF. This is the standard PDF used in importance-sampling the microfacet equation, so is convenient to use. Using this macrosurface-projected area PDF, we can generate a lookup table using a macrosurface-based weighting function:

$$t_{\omega'} = \frac{D(\omega')\cos\theta'}{\int_\Omega D(\omega)\cos\theta\,\mathrm{d}\omega}, \tag{15}$$

where we calculate the shortened normal length by, once again, taking the integral of this weight times the z-component of the sampled direction:

$$\begin{aligned}\left\|\mathbf{n}_{s,macro}\right\| &= \int_\Omega t_{\omega'}\omega'_z\,\mathrm{d}\omega', &(16)\\ &= \int_\Omega \frac{D(\omega')\cos\theta'}{\int_\Omega D(\omega)\cos\theta\,\mathrm{d}\omega}\omega'_z\,\mathrm{d}\omega', &(17)\\ &= \int_\Omega D(\omega')\cos\theta'\omega'_z\,\mathrm{d}\omega'. &(18)\end{aligned}$$

We call this the Macrosurface Area model for calculating shortened normal length. In this case, we would average our normals without weighting by microsurace area since we are dealing with macrosurface area (Algorithm 2), and we would relate this to the table generated by Equation 18.

Algorithm 2 Averaging normals by macrosurface area

```
1: procedure CalcAverageNormalMacro
2:     $averageNormal \leftarrow (0,0,0)$
3:     $totalWeight \leftarrow 0$
4:     for all $n$ in normals do
5:         $averageNormal \leftarrow averageNormal + n$
6:         $totalWeight \leftarrow totalWeight + 1$
7:     $averageNormal \leftarrow \frac{averageNormal}{totalWeight}$
```

It's important to note the table we generate using Macrosurface Area is different from that generated for Microsurface Area (see Figure 6 for a graph of differences), but we can relate both tables to their corresponding normal averaging techniques.

2.1.4 Converting from macrosurface averaging to microsurface averaging

We can show that converting between macrosurface and microsurface models is a matter of applying $\cos\theta$ or its reciprocal to the weight within the integral.

Starting with a refactored version of the Macrosurface Area shortened normal length (Equation 17):

$$\left\|\mathbf{n}_{s,macro}\right\| = \frac{\int_\Omega D(\omega')\cos\theta'\ \omega'_z \mathrm{d}\omega'}{\int_\Omega D(\omega)\cos\theta\,\mathrm{d}\omega}, \tag{19}$$

we can apply the area conversion of Equation 13 ($\frac{1}{\cos\theta}$) to the integral in the numerator and the denominator to convert this to the shortened normal length using Microsurface Area:

$$\left\|\mathbf{n}_{s,micro}\right\| = \frac{\int_\Omega D(\omega')\cos\theta'\ \omega'_z \frac{1}{\cos\theta'}\,\mathrm{d}\omega'}{\int_\Omega D(\omega)\cos\theta\,\frac{1}{\cos\theta}\,\mathrm{d}\omega}, \tag{20}$$

$$= \frac{\int_\Omega D(\omega')\,\omega'_z\,\mathrm{d}\omega'}{\int_\Omega D(\omega)\,\mathrm{d}\omega}, \tag{21}$$

$$= \frac{1}{\int_\Omega D(\omega)\,\mathrm{d}\omega}. \tag{22}$$

Converting the Microsurface Area equation to the Macrosurface Area equation is a matter of applying the inverse area conversion.

2.1.5 Differences between microsurface and macrosurface models

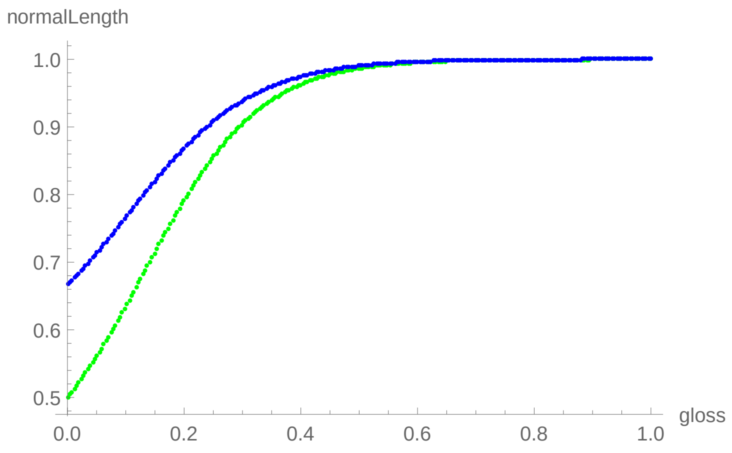

Figure 6: Plot of gloss values to $\left\|\mathbf{n}_{s,micro}\right\|$ in green and $\left\|\mathbf{n}_{s,macro}\right\|$ in blue.

Figure 6 shows the differences in shortened normal length between the microsurface and macrosurface models. These differences motivated further exploration of how we relate normal maps to NDFs via the microsurface and macrosurface models.

In theory, since $D(\omega)$ relates directly to microsurface area, we should be using the Microsurface Area model. But if we were to attempt to correctly filter NDFs (gloss) while rendering, we'd need to account for the microsurface area of the facet associated with the normal (Equation 13). In other words, as we average and filter NDF texels in the pixel shader we would need to account for the microsurface area change due to the normal direction.

An argument in favor of using the Macrosurface Area model – in spite of the fact that our NDF relates directly to microsurface area – is that hardware filtering occurs in the macrosurface domain. In other words, when using GPU bilinear filtering, we are averaging over pixel footprints on the macrosurface.

Assuming we are limiting ourselves to hardware filtering, this leaves us with two solutions, neither of which are ideal – either we use the Microsurface Area model and filter using the incorrect area measure, or we use the Macrosurface Area model and filter the slightly modified, but incorrect, distribution appropriately.

Normal maps are geometrically flat and represent uncorrelated facets, while the microfacet equation has various degrees of correlation through its geometry term – i.e. there is shadowing and masking of microfacets by other microfacets. Further research is required to understand what our goals are in rendering normal maps and the NDFs representing those normal maps at a distance, and which of these models is closer to achieving those goals.

In *Call of Duty: WWII*, we used the Macrosurface Area model and all the values calculated in this paper use macrosurface area normalization, including the values in the table relating gloss to shortened normal length. To generate that table, we importance sampled GGX using the PDF from Equation 14.

2.1.6 *Closed-form solutions to shortened normal length*

While we used tables generated using importance sampling during development of *Call of Duty: WWII*, we can also derive closed-form solutions to shortened normal length using both Microsurface (Equation 12) and Macrosurface Area (Equation 18) methods:

$$\left\|\mathbf{n}_{s,micro}\right\| = \frac{1}{\int_\Omega D_{GGX}(\omega)\,\mathrm{d}\omega}, \tag{23}$$

$$= \frac{a}{a + (1-a^2)\tanh^{-1}(a)}, \tag{24}$$

$$\left\|\mathbf{n}_{s,macro}\right\| = \int_\Omega D_{GGX}(\omega)\cos\theta\,\omega_z\,\mathrm{d}\omega, \tag{25}$$

$$= \frac{a - (1-a^2)\tanh^{-1}(a)}{a^3}, \tag{26}$$

$$\text{where } a = \sqrt{1-\alpha^2}. \tag{27}$$

These closed-form solutions (C implementation in Listing 7) supersede our importance sampling method (Listing 6) since they are faster and more accurate.

2.2 Mipmapping Process

Using our gloss to shortened normal length table, we can construct a shortened normal map where each pixel represents both the normal, through the direction of the vector, and gloss, through the length of the vector. An interesting property of a shortened normal map is that filtering of the map is meaningful – for example, the average of shortened normals is equivalent to averaging the individual (Macrosurface Area model) NDFs associated with the shortened normals. Also notice the filtered normal direction is weighted according to normal distribution since we are not merely filtering unit normals. Averaging between a glossy normal pointing one direction and a rough normal pointing another will result in a normal closer to the glossy normal direction; as it

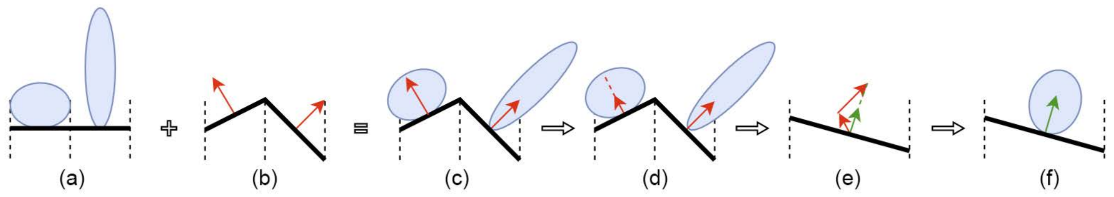

Figure 7: From left to right: (a) Gloss map representing per-pixel NDF (rough left pixel, smooth right pixel), (b) normal map, (c) normal and gloss combined represents the macro and micro-surface details, (d) the broader NDF on the left is represented by a shortened normal, (e) mipmapping generates a shortened normal that is the average of the two original shortened normals, and (f) this shortened normal represents the mipmapped normal direction and gloss (NDF).

should, since the distribution of normals favors that direction (the mipmapped normal points more to the right in Figure 7).

If we were able to encode these shortened normal maps such that we could calculate their lengths down to six base-10 digits of precision (see the last several normal lengths in the table in Listing 8), we could pass these shortened normals directly to the GPU and take advantage of hardware (e.g. bilinear) filtering. Then, in the pixel shader, we could reconstruct gloss from the length of the shortened normal using a fit curve (Section 3.1), or more likely we'd convert straight to α or α^2 for GGX. Unfortunately, half precision floating point with BC6H is not enough and uncompressed 16- and 32-bit formats are too memory intensive.

Instead, after generating the mipmaps of the shortened normals, we calculate the gloss from the length of the shortened normal and renormalize. We chose to store our normals as a BC5S, and our gloss separately as a single channel in a composited texture which also holds metalness and material surface occlusion (Section 4). Run-time filtering of normals and gloss independently is not quite the same as filtering shortened normals, but it's a close approximation. Algorithm 3 outlines the steps performed in mipmapping our normal and gloss maps.

Algorithm 3 Pseudocode for mipmapping normal and gloss maps

```
1: procedure MipmapNormalAndGlossMap($N$, $G$)
2:     for all pixels $N(x, y)$ in $N$ do
3:         $len \leftarrow$ GlossToNormalLength($G(x, y)$)
4:         $N(x, y) \leftarrow len \times N(x, y)$
5:     $N_M \leftarrow$ Mipmap($N$)                                ▷ returns an array of images
6:     for all mipmaps $N_M[i]$ in $N_M$ do
7:         for all pixels $N_M[i](x, y)$ in $N_M[i]$ do
8:             $G_M[i](x, y) \leftarrow$ NormalLengthToGloss($N_M[i](x, y)$)
9:             $N_M[i](x, y) \leftarrow$ Normalize($N_M[i](x, y)$)
10:    return $N_M$, $G_M$
```

Figure 8 shows an authored gloss map of $\frac{255}{255}$ and the generated mipmaps from our technique. The lower mips encode higher normal variation and are darker, representing lower gloss values.

Figure 9 shows the result of pushing normal variation into gloss (left) and a comparison against the same surface rendered up close and bilinearly downsampled (right).

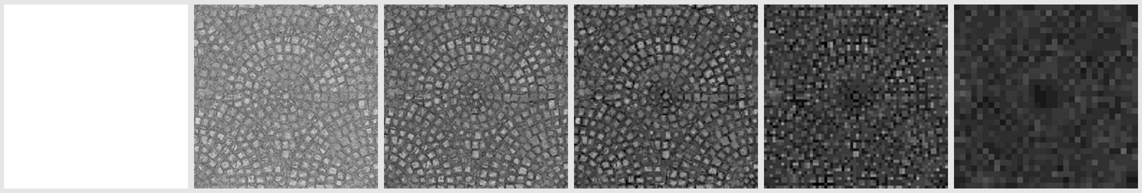

Figure 8: MIP 0 through 5 of a demonstration glossmap. The lower resolution mips to the right get darker to represent more normal map normal variation.

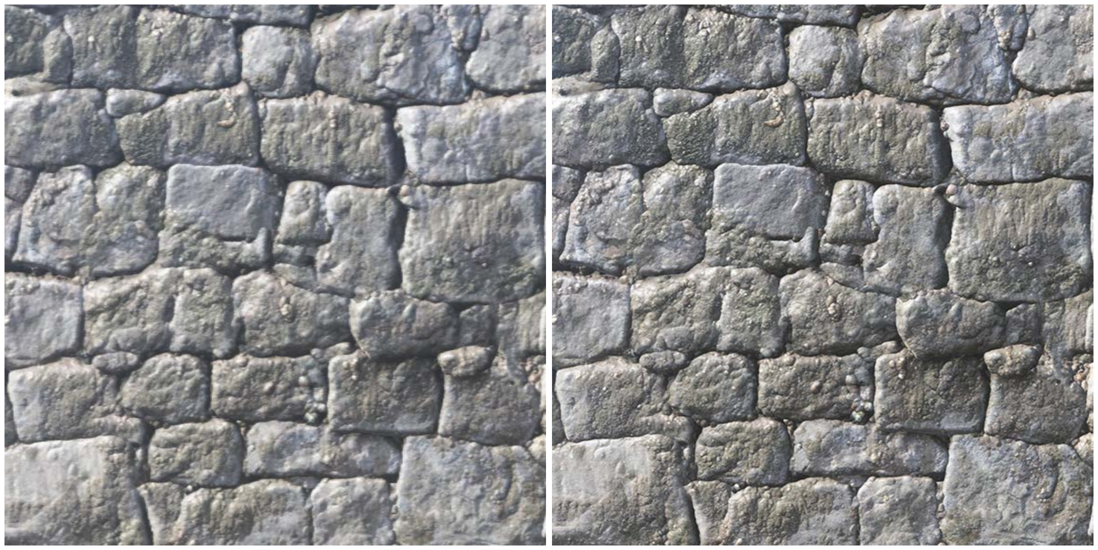

Figure 9: Left: Surface rendered in the distance. Right: Bilinear downsample of surface up close.

2.3 Representation

When thinking about normal maps, what exactly do they represent? Do normal map pixels represent facets of a faceted surface, or are they samples from a continuous smooth surface (Figure 10)? In fact, you can ask the same question about gloss: does it represent consistent gloss across the entire pixel footprint, or does it represent a gloss sample at the pixel center?

The answer to these questions changes how we treat authored normals and gloss at MIP 0, our highest source resolution. For this project, we treated the normals as facets. This was mostly an authoring decision – we didn't want to alter the artist-authored MIP 0 texture, so we consider normals as representing a faceted surface.

You could easily make the argument that normals represent a smooth surface. In that case, you would modify your MIP 0 gloss to take into account the NDF across the entire pixel footprint. A simple approximation is to generate shortened normals at random within the pixel footprint, interpolating the normal directions, then averaging them. Going further, you could use a filter kernel to determine the facetedness/smoothness of pixel areas, e.g. large changes in normal direction represent a faceted edge.

It's important to note the smooth surface idea falls apart under magnification without some further modifications. Under magnification the rendered pixel footprint might not cover the full extent of the source normal and gloss map, so the underlying normal distribution wouldn't be as broad.

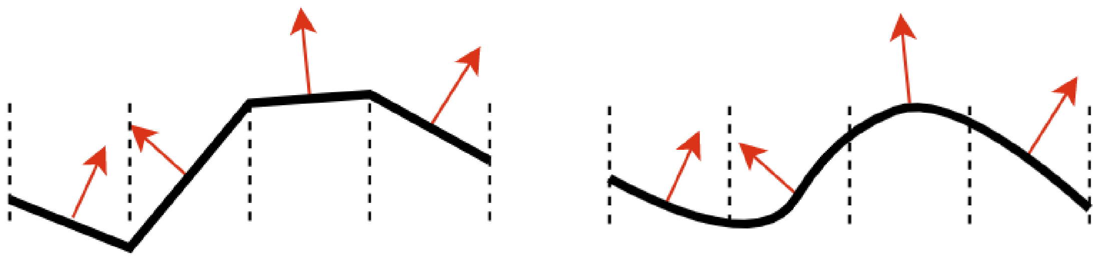

Figure 10: Left: Faceted interpretation of normals. Right: continuous surface interpretation of normals.

The issue with any approximate technique like this is that it can't represent the full composite NDF – it represents a complex NDF with the closest match constrained by our shading model (see Figure 7), which only uses one dimension for gloss. Future directions might be to provide kurtosis (haze) control, anisotropic support, or to model NDFs more fully, e.g. [Yan+14; Jak+14].

2.4 Detail Normal Gloss

We run into an issue when we use detail normal maps. As detail normal maps recede into the distance and mip to lower resolutions, their normal variation is represented in a detail gloss channel. How, then, do we combine this detail gloss with our base gloss?

We need to define the gloss combining step. We imagine this as choosing a normal at random from our detail NDF, rotating our random detail normal into the space of a randomly chosen normal from our base NDF. We do this many times and calculate a shortened normal for this new distribution.

Given two NDFs, A and B, we can randomly sample from each of them individually to calculate our shortened normals, $\mathbf{n}_a$ and $\mathbf{n}_b$:

$$\mathbf{n}_a = \frac{1}{N}\sum_{i=1}^{N}\mathbf{n}_{a,i}, \tag{28}$$

$$\mathbf{n}_b = \frac{1}{M}\sum_{j=1}^{M}\mathbf{n}_{b,j}. \tag{29}$$

Our gloss combining step is equivalent to randomly sampling from one distribution, and rotating into the space defined by randomly sampling from the other distribution:

$$\mathbf{n}_s = \frac{1}{M}\sum_{j=1}^{M}\left(\frac{1}{N}\sum_{i=1}^{N}\mathbf{n}_{a,i|b,j}\right), \tag{30}$$

where $\mathbf{n}_{a,i|b,j}$ is $\mathbf{n}_{a,i}$ rotated into the space of $\mathbf{n}_{b,j}$.

The length of the shortened normal from the inner sum is

$$\left\| \frac{1}{N} \sum_{i=1}^{N} \mathbf{n}_{a,i|b,j} \right\| = \left\| \frac{1}{N} \sum_{i=1}^{N} \mathbf{n}_{a,i} \right\| = \|\mathbf{n}_a\| \,. \tag{31}$$

The outer sum is equivalent to summing vectors with length $\|\mathbf{n}_a\|$:

$$\mathbf{n}_s = \frac{1}{M} \sum_{j=1}^{M} \left(\|\mathbf{n}_a\| \, \mathbf{n}_{b,j} \right) \tag{32}$$

$$= \|\mathbf{n}_a\| \frac{1}{M} \sum_{j=1}^{M} \mathbf{n}_{b,j}. \tag{33}$$

$$\|\mathbf{n}_s\| = \|\mathbf{n}_a\| \, \|\mathbf{n}_b\| \,. \tag{34}$$

In order to combine gloss, then, we multiply shortened normal lengths of A and B and convert back to gloss. This ignores any normal direction variation that comes from the averaging of detail normals, i.e. we assume that the gloss values we are combining are along the same normal.

We generate a 32 × 32 lookup table (LUT) and then perform a rational function fit (Figure 11), which results in the equations below (shader code in Listing 2). We can reduce the degrees of freedom of the rational function fit because the two dimensions/arguments are symmetric/reciprocal, i.e. we can swap the two arguments.

$$P(a, b) = -0.535580 + 1.002204\,(a + b) - 0.223910\,(a^2 + b^2) + 13.323150\,a\,b \tag{35}$$

$$Q(a, b) = 1.0 + 8.259559\,(a + b) + 9.896132\,(a^2 + b^2) - 22.015902\,a\,b \tag{36}$$

$$g = \text{clamp}\left(\frac{P(a, b)}{Q(a, b)}, 0, 1 \right). \tag{37}$$

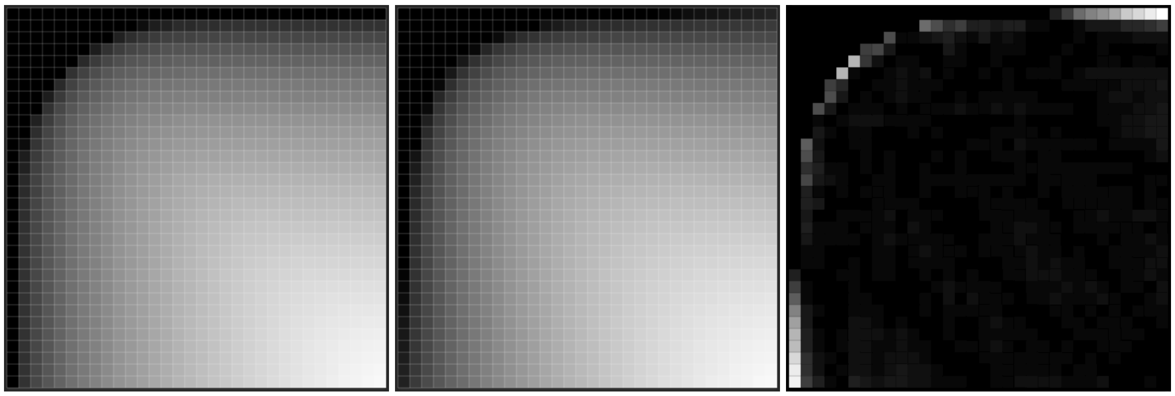
Figure 11: From left to right: Gloss Combine (RNM) lookup table, Gloss Combine (RNM) approximated using rational function, normalized difference between lookup table and rational function.

Our normal maps are not blended using Reoriented Normal Maps (RNM) [BH12], which is the assumption for the LUT built above. Instead our normals are blended by adding the detail normal offsets (X and Y) to the base

Algorithm 4 Combining base and detail normals using BlendXY

```
1: procedure CombineNormals($B$,$D$)
2:     $C_x \leftarrow B_x + D_x$
3:     $C_y \leftarrow B_y + D_y$
4:     $C_z \leftarrow \sqrt{\max\left(1 - C_x^2 - C_y^2, 0\right)}$
5:     return $C$
```

normal offsets, and then recalculating Z – we call this method BlendXY (Algorithm 4). We use our RNM-based LUT in spite of this, since both methods combine gloss similarly; we compared the generated tables for both methods and they are very close.

In order to validate this approach of multiplying shortened normal lengths, we've also generated LUTs using a brute force approach where we randomly sample from both NDFs and combine the normals as we would a base normal and detail normal, using either RNM or BlendXY. Averaging a large number of these normals from the new NDF, and converting to gloss, we end up with the same results, in the case of RNM, and very similar results for BlendXY.

3 RATIONAL FUNCTION FITTING

Fitting functions using rational functions for rendering was introduced in [Sch94].

$$F_{\text{Schlick}}(\mathbf{v}, \mathbf{h}) = F_0 + (1 - F_0)\,(1 - \mathbf{v} \cdot \mathbf{h})^5. \tag{38}$$

This is the paper that introduced Schlick Fresnel (Equation 38), which is widely used in physically based rendering. In my opinion though, the more important takeaway was the use of rational function approximations, and the idea of kernel conditions.

A rational function is an algebraic fraction of polynomials. Here's a simple example:

$$f(x) = \frac{c_0 + c_1 x + c_2 x^2 + c_3 x^3}{1 + c_4 x + c_5 x^2}. \tag{39}$$

Kernel conditions are a key idea when performing a fit, in that we ask: What are the salient characteristics of the function we are fitting? What makes this function special? Then, we shape our approximation to hit these targets. While the paper called these "kernel conditions", we prefer to use the term "constraints".

Notice the rational functions we use have 1.0 as the constant term in the denominator polynomial, to remove an unnecessary degree of freedom. This also gives us an easy way to fix the constraint when $x = 0$, $f(0) = c_0$.

3.1 Example 1D Rational Function Fit

Section 2 discusses our method for normal and gloss mipmapping to combat specular aliasing. As part of that process we generate a table that relates gloss to shortened normal length (Listing 8). We use this table directly to evaluate GlossToNormalLength and NormalLengthToGloss in Algorithm 3.

Notice we can also use the closed-form solution (Equation 26) to evaluate GlossToNormalLength. As an alternative, we are going to perform a rational function fit of the table (Listing 8) purely as a demonstration of the effectiveness of this technique. First, we will fit without constraints, and then we will refit it with constraints. Figure 12 shows this graph of gloss to average normal length, along with the rational function fit.

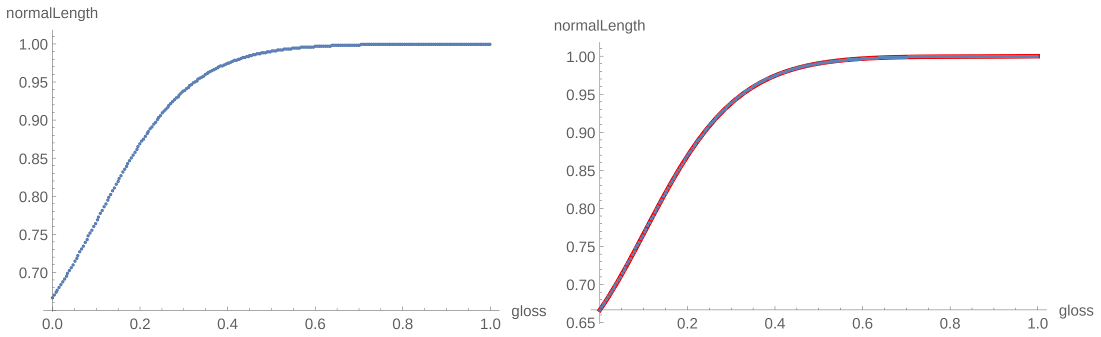

Figure 12: X-axis is gloss, while Y-axis is average normal length. Right image shows rational function fit in red.

Using `FindFit` in Mathematica on our table, and the rational function form of Equation 39, we get this rational function approximation:

$$f(x) = \frac{0.66693 + 0.0609009x + 7.32398x^2 + 0.600464x^3}{1 - 1.13467x + 8.78264x^2}. \tag{40}$$

This is a simple example without constraints, but what if we wanted fit our curve with some constraints? Since $f(0) = 0.66667$, we know that $c_0 = 0.66667$. We are, in effect, fixing the start point of the curve. This removes one degree of freedom. We wouldn't necessarily consider this a true constraint for this function since we can accept some error at this end of the curve, but we are doing this as a demonstration.

We also know $f(1) = 0.999958$, our other constraint. Unlike the previous constraint, this constraint is important since small errors at this end of the curve can lead to large errors when evaluating gloss for shortened normals generated with this curve. With this, we are fixing the right end of the curve.

With some rearranging we can remove c_3 – this is an arbitrary choice, we could remove any of the other coefficients – by specifying c_3 as a function of the remaining coefficients.

$$c_3 = (0.999958 - 0.66667) - c_1 - c_2 + 0.999958c_4 + 0.999958c_5. \tag{41}$$

Using `FindFit` again in Mathematica gives us our final rational function approximation (right image in Figure 12):

$$f(x) = \frac{0.66667 + 0.0942205x + 7.41935x^2 + 0.567111x^3}{1 - 1.0881x + 8.83582x^2}. \tag{42}$$

4 MATERIAL SURFACE OCCLUSION

A common complaint by our artists is that things look flat when in shadow.

We have solutions for geometric occlusion, in the form of SSAO (GTAO and variants), MDAO (medium distance AO, cone tracing + occlusion volume texture) and per-vertex bent normals to encode self visibility, but we are missing proper material surface occlusion. Cavities and bumps defined by the normal map should occlude indirect light and if possible, direct light as well, for self-shadowing.

How do we generate material surface occlusion? We start with the authored normal map and we generate a height map from it using relaxation (Section 4.2). Then, we generate an occlusion map from the height map using GTAO (Section 4.3). Since we do all of this during an offline material conversion step, we've created a C++ wrapper to be able to compile the GTAO shader code as C++.

4.1 Generating a Height Map from a Normal Map

To go from normal map to occlusion map, we first generate an intermediate height map corresponding to the normal map (Figure 13).

It's important to start with a well-authored normal map. The maps that work the best are ones that are captured using photogrammetry. Normal maps that are painted or modified using normal mapping tools can many times be "impossible" normal maps, that is, normal maps that don't correspond to any real surface. In these cases, there is no height map that can be created that matches the normal map. If we then use the generated height map for displacement and the normal map for lighting, as is the case with our Tessellation and Displacement systems, we will get unrealistic lighting.

One solution to the height map from normal map problem is to solve a linear system using traditional means [Cou04], but this is slow and scales poorly with texture size. Another solution, from Dmitriev and Makarov [DM11], assumes that the integral of height differences is zero, and their method processes each pixel individually.

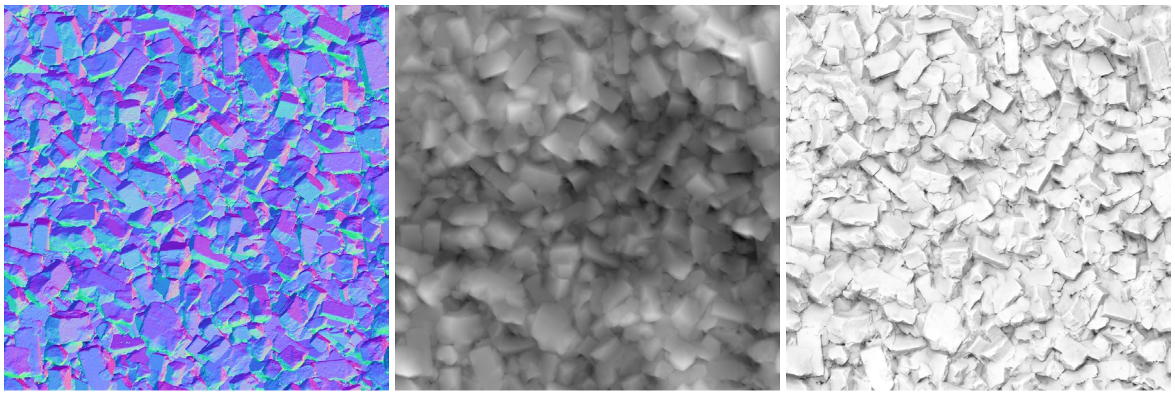
Figure 13: From left to right: Brick rubble normal map, the height map generated from the normal map, and the occlusion map generated from the height map.

4.2 Relaxation

Instead we perform the process iteratively via relaxation, where we choose each pixel randomly and calculate the height it should be at given the heights of its neighbors and the height differences from its neighbors. After many iterations of adjusting single pixels, the height of each pixel converges towards a solution.

This brute force approach can be very slow, so our technique refines its heights up through the mip-chain, solving lower resolution mips and using those solutions as the first estimate for the next higher mip (Algorithm 5). During our discussions, we found [Kol15], which is also very similar and serves as a good description of the technique.

Algorithm 5 Simplified pseudocode for converting normal maps to height maps

```
 1: procedure BuildDxDyMaps(N)
 2:     for all pixels $n \leftarrow N(x, y)$ do
 3:         $Dx(x, y) \leftarrow \frac{-n.x}{n.z}$
 4:         $Dy(x, y) \leftarrow \frac{-n.y}{n.z}$
 5:     return Dx and Dy
 6: procedure BuildHeightFromDxDy(Dx, Dy)
 7:     Downsample $Dx$ and $Dy$ maps to $\frac{1}{2}$-resolution $Dx'$ and $Dy'$
 8:     $H' \leftarrow$ BuildHeightFromDxDy($Dx'$,$Dy'$)
 9:     Upsample $H'$ to full-resolution $H$
10:     repeat
11:         for all pixels $H(x, y)$ do
12:             $L \leftarrow H(x-1, y) + Dx(x-1, y)$
13:             $R \leftarrow H(x+1, y) + Dx(x, y)$
14:             $U \leftarrow H(x, y-1) + Dy(x, y-1)$
15:             $D \leftarrow H(x, y+1) + Dy(x, y)$
16:             $H''(x, y) \leftarrow \frac{1}{4}(L + R + U + D)$        ▷ Normals represent slopes down and to the right.
17:         $H \leftarrow H''$
18:     until convergence
19:     return $H$
```

4.3 Generating an Occlusion Map from a Height Map

After building a height map from normals, we can build the occlusion map from the height map by calling into `GTAO.hlsl` [Jim+16], using a wrapper file, `hlsl2cpp.h`, which provides HLSL features to C++. GTAO normally acts on a depth buffer generated using a perspective projection, but the author of GTAO made modifications to allow it to be used on orthographic depth.

Once we have an occlusion value in the shader – which is actually visibility since 1.0 means "no occlusion" – we can modulate indirect diffuse lighting by this term. Also, since this occlusion term is a cosine-weighted integral of visibility, we can convert to an equivalent cone angle for use in direct lighting attenuation. But first, we're going to discuss an adjustment to our ambient occlusion.

4.4 Adjustment of Ambient Occlusion Value

A problem with the standard formulation of ambient occlusion is that occluded directions are assumed to be black, i.e. there is no incoming light from occluded directions.

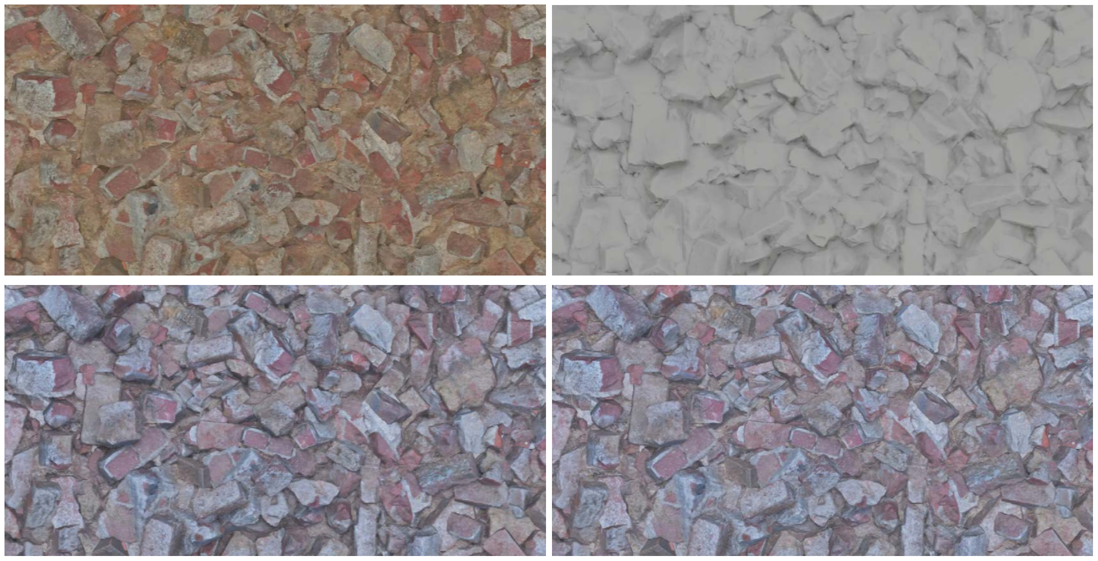

Figure 14: Brick rubble in shadow, clockwise from top-left: Diffuse albedo only, raw material surface occlusion values, render with no material surface occlusion, render with material surface occlusion.

We typically define ambient occlusion at point p as

$$A_p = \frac{1}{\pi} \int_\Omega V_{p,\omega} \cos\theta \, \mathrm{d}\omega, \tag{43}$$

where $V_{p,\omega}$ is the visibility function from p in the direction ω for some finite distance d. Then, assuming our point is in shadow and only affected by indirect light, outgoing radiance for a Lambertian surface would typically be calculated using

$$L_p = \frac{\rho}{\pi} E_p A_p, \tag{44}$$

where ρ is the albedo of the surface and E_p is irradiance. Since A_p is the cosine-weighted integral of visibility over the hemisphere, occluded directions don't contribute at all to its final radiance.

This is a poor approximation to what actually happens in the real world and we can reformulate ambient occlusion using better assumptions about our lighting environment.

To keep our equations simple, we will assume a white furnace environment of radiance 1, where black, the radiance in occluded directions, is 0. We can see that ambient occlusion can be formulated as

$$A'_p = 0\,(1 - A_p) + 1\,(A_p). \tag{45}$$

Now we can reformulate ambient occlusion, depending on the following assumptions about the occluded directions.

Occluded directions have white diffuse albedo and are occluded similarly:

$$A'_p = A_p\,(1 - A_p) + 1\,(A_p). \tag{46}$$

Occluded directions have the same diffuse albedo, but are unoccluded:

$$A_p' = \rho\,(1 - A_p) + 1\,(A_p). \tag{47}$$

Occluded directions have the same diffuse color, and are occluded similarly:

$$A_p' = \rho A_p\,(1 - A_p) + 1\,(A_p). \tag{48}$$

Going further, we can use the interreflection model of [SL96] and assume that the radiance of the occluded directions is equal to the outgoing radiance of the pixel being shaded.

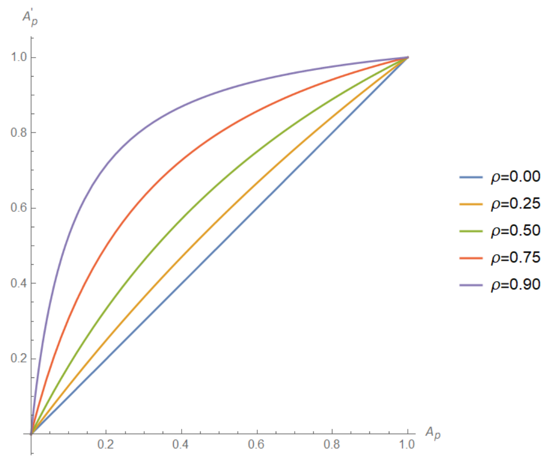

Figure 15: Graph of effect of albedo ρ on adjusted ambient occlusion A_p'.

Figure 15 shows how adjusted ambient occlusion A_p' is affected by the diffuse albedo of the point being shaded, assuming the interreflection model of Equation 48. Notice when albedo ρ is zero, ambient occlusion remains unchanged since occluded directions are still black. As albedo ρ increases, adjusted ambient occlusion A_p' tends to be less pronounced.

$$A_p' = \frac{A_p}{1 - \rho + \rho A_p}. \tag{49}$$

The challenge now is how to author diffuse textures. Previously, artists needed to bake some occlusion into our diffuse textures, since we didn't have material surface occlusion. Now that we auto-generate material surface occlusion, these textures should be strictly diffuse albedo. This is a change in how we visualize and author them.

4.5 Converting Visibility Value to Equivalent Cone Angle

It is fairly straightforward to convert from a visibility term to the equivalent cone angle. [Jim+16] has a formulation using integrals, but we can also reason about this geometrically.

Using the Nusselt analog [Wik], we can relate the area of a disk C to the unit disk as "cosine-weighted visibility"

(Equation 50) and the disk C projected up to the hemisphere representing the equivalent cone (Figure 16).

$$\frac{\text{cos-weighted visibility}}{\text{hemisphere visibility}} = \frac{\text{area of disk } C}{\text{area of unit disk}} \tag{50}$$

$$\frac{V}{1} = \frac{\pi r^2}{\pi} \tag{51}$$

$$V = r^2. \tag{52}$$

r, then, is the radius of disk C and the projected cone and we can relate that to cone angle:

$$r = \sin\theta \tag{53}$$

$$V = \sin^2\theta \tag{54}$$

$$V = 1 - \cos^2\theta \tag{55}$$

$$\cos\theta = \sqrt{1 - V}. \tag{56}$$

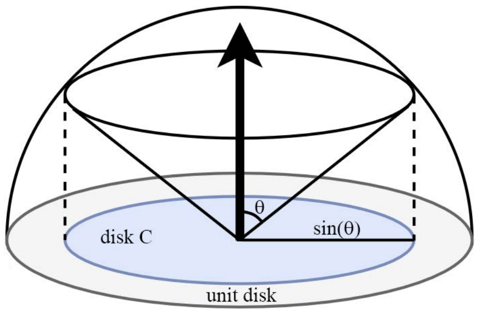

Figure 16: Disk C.

Figure 17 shows the effect of using material surface occlusion for direct light shadowing in the rightmost image.

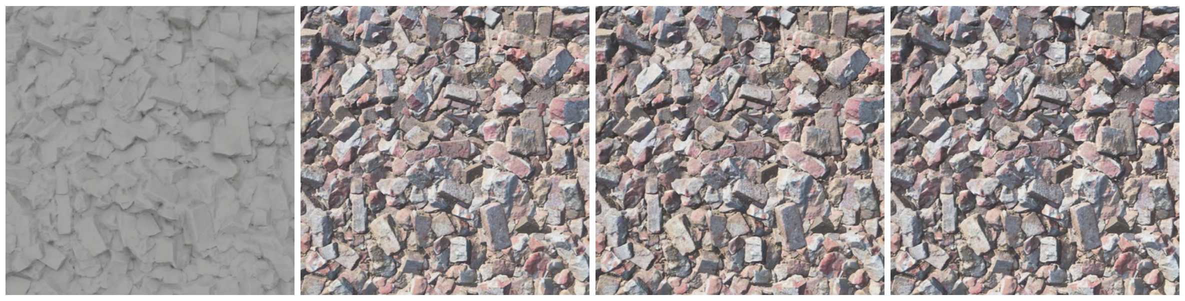

Figure 17: Brick rubble in the sun, from left to right: Raw material surface occlusion values, render with no material surface occlusion, render with material surface occlusion, render with material surface occlusion to approximate primary light shadowing.

4.6 Cone-Based Direct Light Occlusion

Now we have a method of relating ambient visibility to cone angle, we can apply this to shadow direct lighting.

$$vis_{\mathbf{l}} = \begin{cases} 1 & \text{if } \cos\theta < \mathbf{n} \cdot \mathbf{l}, \\ 0 & \text{otherwise.} \end{cases} \tag{57}$$

Since this normal-oriented cone is a poor approximation of the actual directional visibility of the point being rendered, self-shadowing using Equation 57 looks poor (Figure 18, top-right).

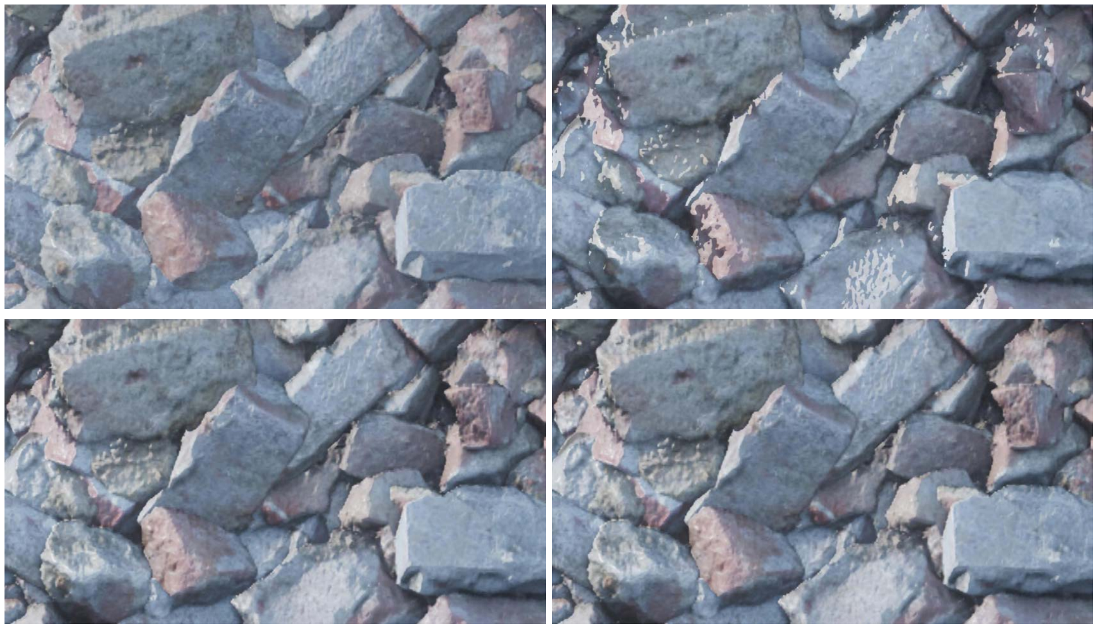

Figure 18: Equivalent cone for direct light shadowing, clockwise from top-left: cone is not used; cone is used as exact visibility function (notice the dark splotches); cone has a smooth falloff: $\frac{\mathbf{n}\cdot\mathbf{l}}{\cos\theta}$; cone has a sharper falloff: $\left(\frac{\mathbf{n}\cdot\mathbf{l}}{\cos\theta}\right)^2$.

Instead, we apply an ad hoc falloff to the lighting following a scaled $\mathbf{n} \cdot \mathbf{l}$ factor from the edge of the cone to 90 degrees (see Listing 4 for shader code).

$$vis_{\mathbf{l}} = \text{clamp}\left(\frac{\mathbf{n} \cdot \mathbf{l}}{\cos\theta}, 0, 1\right)^2. \tag{58}$$

Since this falloff is ad hoc, in the future we could reformulate this as a function of V and avoid the square root in calculating a precise $\cos\theta$.

[BM16] does something similar, applying what they call “Micro Shadows” using their own ad hoc falloff tied to AO maps that are not necessarily authored consistently. Since we generate our AO maps using GTAO, our falloff can be based on the equivalent cone angle.

GPU-based Tessellation and Displacement (T&D) biases shadow map rendering to avoid z-fighting, so using some sort of cone-based occlusion should help with the direct lighting of these surfaces.

Also, in theory, our SSAO technique (we use GTAO) will pick up the geometric occlusion of T&D, so we should omit applying the occlusion map to indirect lighting for GPU T&D. In practice, we take the `min` of occlusion values generated by GTAO and Material Surface Occlusion. This is not ideal since GTAO and Material Surface Occlusion can often represent occlusion at different scales. In the future, we may explore other methods of combining these values.

4.7 Cone-Based Indirect Light Specular Occlusion

Given a visibility cone, we can also attenuate our indirect specular contribution. If the cone represented exact visibility, we'd need to integrate our specular BRDF against this cone to determine how much specular light reaches the eye.

Integrating our Specular BRDF against the visibility cone gives this $32 \times 32 \times 8$ environment lookup table (Figure 19). This table is just an extension of the 2D lookup table of [Kar13], which we used in *Call of Duty: Advanced Warfare*, into a third dimension: cone angle. [Jim+16] presents a similar extension, going further and using bent-normals for more accuracy. We use the same technique for generating our LUT as before, importance sampling the GGX NDF, except now we reject samples that fall outside the cone.

The left-most slice in the table represents a completely unoccluded cone (the full hemisphere) and is equivalent to the 2D lookup table we used previously. The right-most slice represents a completely occluded cone ($\theta = 0$); notice it is completely black.

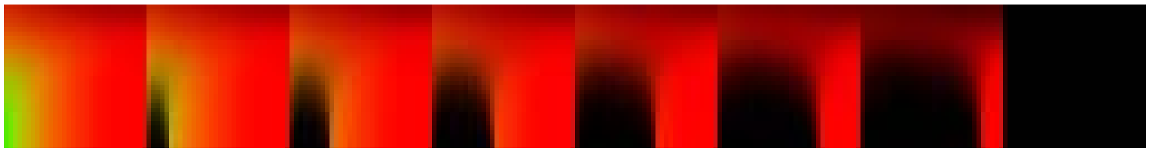

Figure 19: An environment BRDF lookup table that takes specular occlusion into account. Each slice represents a different cone angle.

Unfortunately, a normal-centered cone is not a good approximation to actual directional-dependent occlusion. For low gloss surfaces, it works well enough since there is significant overlap between the BRDF lobe and the cone at most angles, but as gloss increases, the probability and extent of overlap decreases. We see this issue in practice as well, with glossy surfaces getting unnaturally occluded (Figure 20).

Because of this, we use an ad hoc adjustment of cone angle to bias the lookup towards a completely unoccluded cone as gloss increases. It also takes into account the viewing angle of the surface. This adjustment was tuned by eye.

$$\cos\theta' = \cos\theta\,(1 - gloss^2)(1 - (1 - (\mathbf{n}\cdot\mathbf{v})^4)). \tag{59}$$

Unfortunately, using this adjustment alone still resulted in over-occlusion of glossy player (viewmodel) weapons, because we always view their surfaces at extreme grazing angles. Because of this, we biased our *gloss* before applying Equation 59 using

$$gloss' = \text{clamp}(1.5\,gloss, 0, 1). \tag{60}$$

While this fixes issues with glossy player weapons, this results in the right image in Figure 20, which shows a lack of specular occlusion. This is what we shipped with in game, but in the future, we may apply Equation 60 selectively only on materials that require it. See Listing 3 for shader code. This ad hoc adjustment

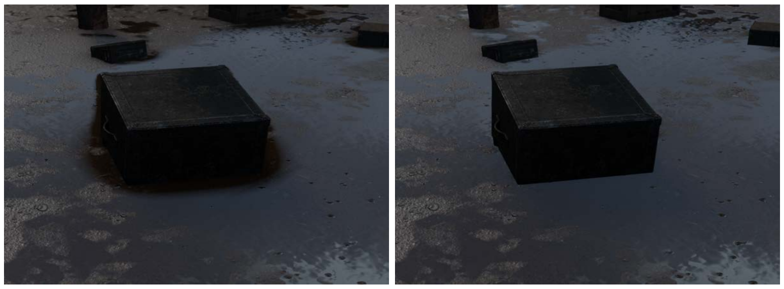

Figure 20: A box on a puddle of water. Notice the differences in specular response on the water. Left: an example of an over-occlusion artifact from using a normal-centered cone as an approximation to occlusion, Right: using an adjustment to the cone angle based on gloss and viewing angle and biasing gloss before the adjustment so that glossy surfaces don't apply specular occlusionm, which leads to under-occlusion.

might be improved by reasoning about the probability that the cone occlusion is a good approximation to actual directional-dependent occlusion, given gloss.

4.8 Future Directions

The behavior of our cone-based direct light attenuation should match the implicit microfacet shadowing that occurs in our Multiscattering Diffuse BRDF. Ideally, an area spot meter measurement of a bumpy surface in our game which includes the microshadowing from our cone-based occlusion will match that same bumpy surface in the distance after the normal map detail gets pushed into the lower resolution gloss maps (Section 2). This may require some amount of parallax mapping so that micro-masked areas are not rendered.

We can also validate our cone-based specular occlusion against reference renders to improve the accuracy of our approximation.

5 MULTISCATTERING DIFFUSE BRDF

5.1 Rough Surfaces

During *Call of Duty: Advanced Warfare*, we measured the brightness of rough surfaces using a spotmeter from different angles and noticed behavior that is at odds with our expectations [Cha15].

Normally, we'd expect surfaces to be brightest when the light is glancing off the surface and we are looking towards the light; the specular reflection in this case is strongest. But this is not consistent with our brightness measurements for rough surfaces. The right-most image in Figure 21 represents this lighting setup and the rough surface appears darkest here.

How do we explain this? Figure 22 shows how a bumpy surface can both cause shadowing and a retroreflective effect, and visual inspection of the bumpy wall in Figure 21 seems to confirm that these effects are present.

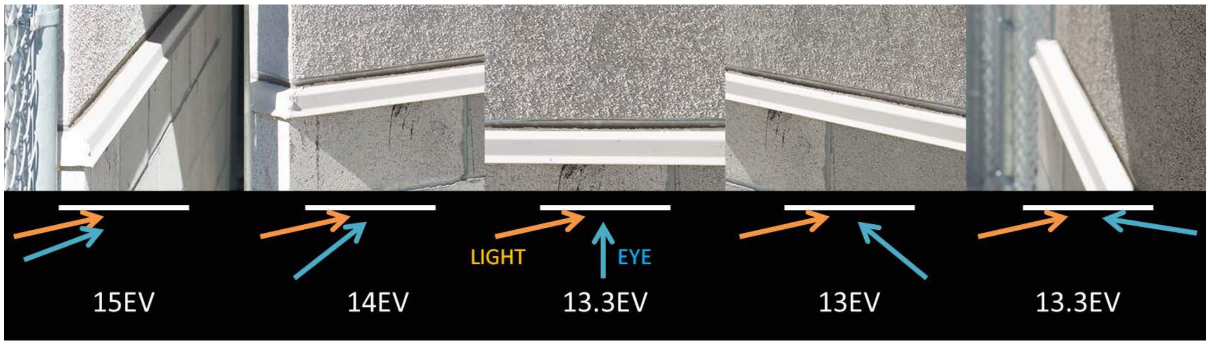

Figure 21: Unexpected measurements made at different angles to a bumpy wall. At far left, the wall is brightest (15EV) when light is behind the viewer.

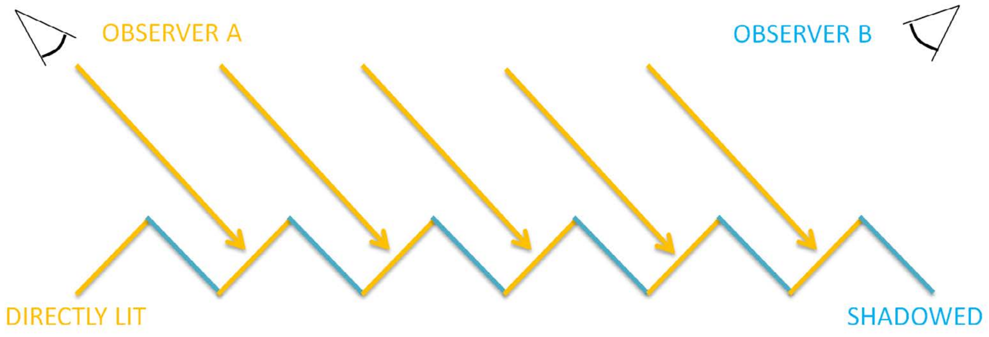

Figure 22: Extreme example of lighting a bumpy surface. Observer A sees the surface as lit while observer B sees the surface as shadowed.

5.2 Smooth Surfaces

[WNO98] had several observations about the diffuse response of rough and smooth surfaces. They noticed a flattening of lighting for rough surfaces, which is by-product of the retroreflective and shadowing effects discussed earlier. In addition, they saw the opposite of flattening, a "rounding", of lighting for smooth surfaces.

We can explain this rounding using energy conservation: reflected specular light is unavailable for diffuse lighting, therefore for smooth surfaces at grazing angles, where specular reflectance is highest, the light available for diffuse reflectance is lowest.

5.3 Energy Conservation

In order to achieve a rounding of lighting, we tried applying energy conservation to our BRDF with Lambertian diffuse, where energy reflected specularly is not available for diffuse reflectance. Figure 23 shows the effect of this energy conservation, where glossy materials have a smoother light falloff. Our final Multiscattering Diffuse BRDF has energy conservation built in, assuming an $F_0 = 0.04$, and it has similar properties to Lambert with energy conservation.

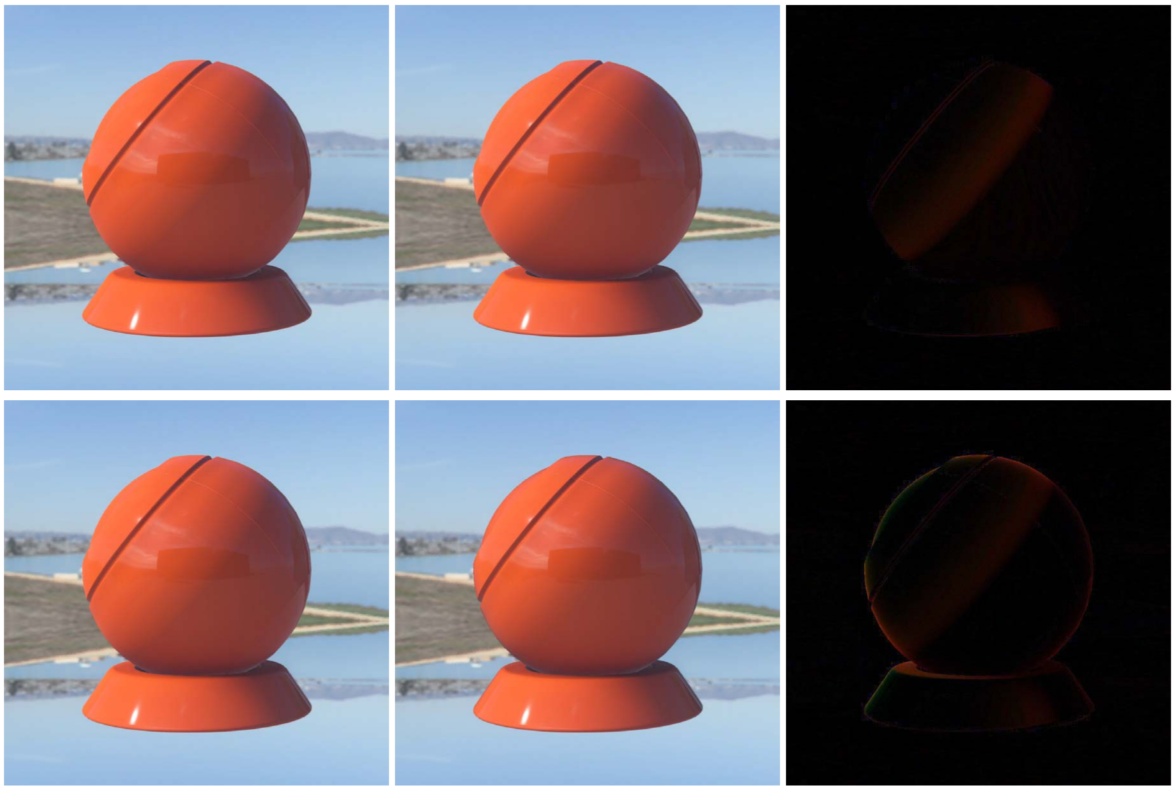

Figure 23: Top row, from left to right: Lambertian diffuse with no energy conservation, Lambertian diffuse with energy conservation, and amplified difference. Bottom row, from left to right: Lambertian diffuse with no energy conservation, Multiscattering diffuse, which models energy conservation with fixed $F_0 = 0.04$, and amplified difference. Notice the smoother light falloff in the energy conserving models because of the glossy surface.

We sample from our `EnvBrdfLut` to determine how much energy is reflected towards the eye during our calculation of indirect specular contribution [Kar13]. In order to calculate the direct light energy lost to specular reflection, we can sample from our `EnvBrdfLut` a second time, this time passing in $\mathbf{n} \cdot \mathbf{l}$ instead of $\mathbf{n} \cdot \mathbf{v}$. We can do this because BRDFs are reciprocal, so our `EnvBrdfLut`, which we normally use to calculate the percentage of gathered light reflected towards the eye, can also be used to calculate the percentage of scattered light reflected from the light source.

Since we ended up using the Multiscattering Diffuse BRDF described next, we didn't apply this explicit energy conservation. See Listing 5 for energy conservation shader code.

5.4 Multiple-Scattering Microfacet Model

What do we want from a new diffuse lighting model? In addition to the properties mentioned above, we'd like the normal distribution of the surface for diffuse to be the same as that for specular. In this way, the glossiness of the surface affects both specular and diffuse response.

Heitz and Dupuy [HD15] chose to model rough diffuse response using the microfacets themselves as Lambertian diffuse reflectors. There is an argument against modeling diffuse reflectance in this way: depending on scale, diffusion distance means the receiving microfacet may not be the re-emitting microfacet for a photon. In spite of this, we use this model as a foundation since it is consistent in its treatment of microfacets.

We used a multiple-scattering microfacet model [Hei+16] to simulate multiscattering diffuse with microfacets following the GGX normal distribution function. Since we simulate multiscattering, energy reflected diffusely can be reabsorbed by neighboring microfacets and reflected diffusely again.

There are a couple of differences between our simulation and that of [Hei+16]. Our multiscattering simulation always bounces 100% of the light, so doesn't have the successive color saturation for later bounces that [Hei+16] had. This is a simplification of the model to reduce the dimensionality of the fit that we apply later. We also modified the model by adding a diffuse Fresnel component to each bounce, simulating the energy loss to specular reflectance.

For this diffuse Fresnel component, we look to [Shi+97], which models a perfectly polished surface with an underlying "matte component", which is exactly what we are attempting to model when we reflect light at the individual microfacet level.

Importantly, we discovered that Equation 4 of [Shi+97] has an error[1] and the correct version is

$$k = \frac{21}{20\pi}(1 - F_0). \tag{61}$$

Our microfacet diffuse Fresnel response is then

$$F_d = \frac{21}{20\pi}(1 - F_0)(1 - (1 - \mathbf{n} \cdot \mathbf{l})^5)(1 - (1 - \mathbf{n} \cdot \mathbf{v})^5). \tag{62}$$

After simulation, we realized the characteristics we are looking for naturally fall out of this new model:

- Flattening of diffuse lighting for rough surfaces from grazing angle retroreflectivity (Figure 24)
- Rounding of diffuse lighting for glossy surfaces from loss of energy to specular reflection (Figure 25)

We also evaluated two other diffuse models: Oren-Nayar [ON94; Fuj12] and Disney [Bur12]. While Oren-Nayar gave a good retroreflective effect for rough surfaces, it becomes Lambertian for smooth surfaces, thereby failing to give us the rounding of lighting that we want. Disney's diffuse model (hereafter *Disney Diffuse*) has both these characteristics, but is unrealistic because its albedo exceeds one. Disney Diffuse is a good compromise between efficiency, physically plausibility and aesthetic concerns though, and we use the model for our sand shader since our artists thought sand looked more appealing in shadow using it, due to enhanced contrast.

5.5 Generating the BRDF Tables

Since simulating the bounce of photons using the multiple-scattering microfacet model is so processor intensive, `SN-DBS` was used to generate the large tables corresponding to the BRDF response over the hemisphere. We generated files using the MERL [Mat+03] parameterization so we could load them into *BRDF Explorer*.

[1] There should be a multiplication by $1 - F_0$ instead of a division.

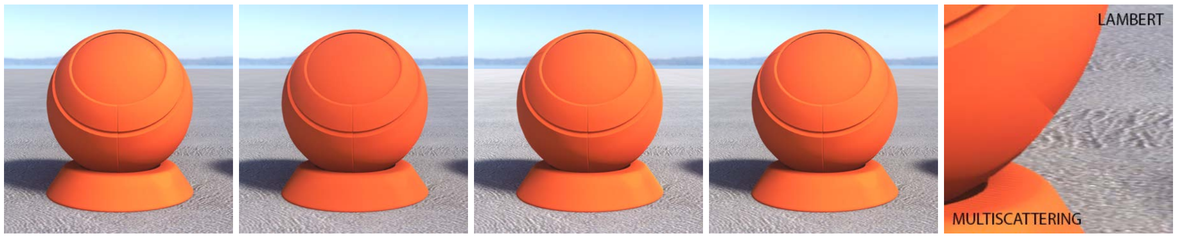

Figure 24: From left to right: Rough diffuse (gloss=0) Lambert, Oren-Nayar, Disney, Multiscattering and a scaled-up comparison of Multiscattering (bottom-left) and Lambert (upper-right). Notice the flattening of lighting in the non-Lambertian models compared to Lambert. Also notice the relative brightnesses of the rough concrete floor in all the images. This lighting scenario is closest to the left-most image in Figure 21.

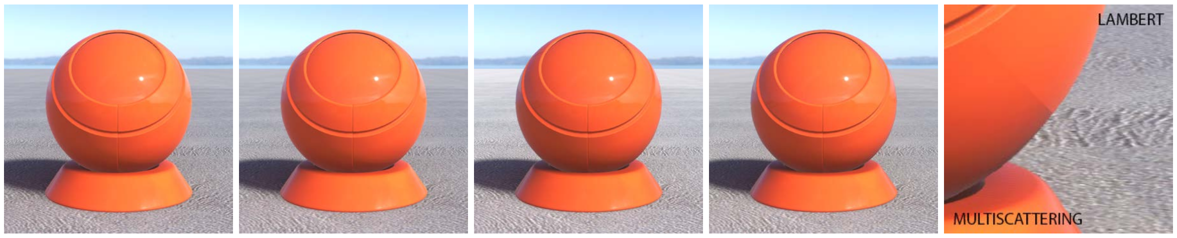

Figure 25: From left to right: Smooth diffuse (gloss=1) Lambert, Oren-Nayar, Disney Diffuse, Multiscattering and a scaled-up comparsion of Multiscattering (bottom-left) and Lambert (top-right). Notice the rounding of lighting in Disney and Multiscattering compared to Lambert, resulting in a darkening of the underside of the sphere and the top of the base on the right side.

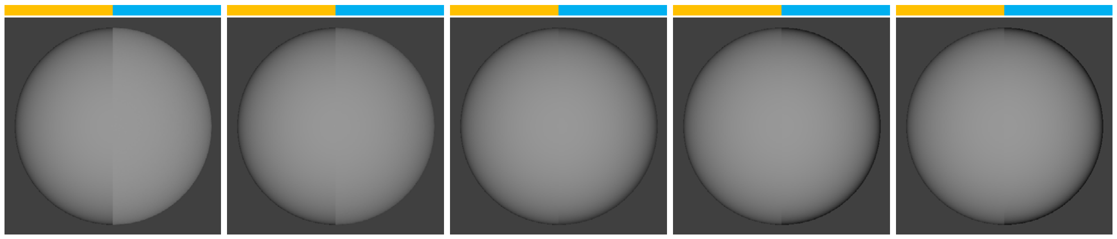

Figure 26: The left-side (yellow) of each rendered sphere is Lambertian Diffuse, while the right-side (cyan) is the full Multiscattering Diffuse BRDF. From left to right: gloss = $\frac{0}{255}, \frac{0}{255}, \frac{120}{255}, \frac{180}{255}, \frac{240}{255}$.

Figure 26 shows a comparison of Lambertian Diffuse with the full Multiscattering Diffuse BRDF. Notice the "flattening" of lighting with rougher surfaces, while the opposite occurs on smoother surfaces – what we call a "rounding" of lighting.

In Figures 27, 28, and 29, notice the stepped nature of the full BRDF table. This is because of the MERL parameterization, which assigns more samples towards the reflected direction. In contrast, our BRDF slice parameterization is equal angle. We've augmented *BRDF Explorer* to read 90×90 pixel Portable Float Map (PFM) files representing the BRDF slice, which we discuss below.

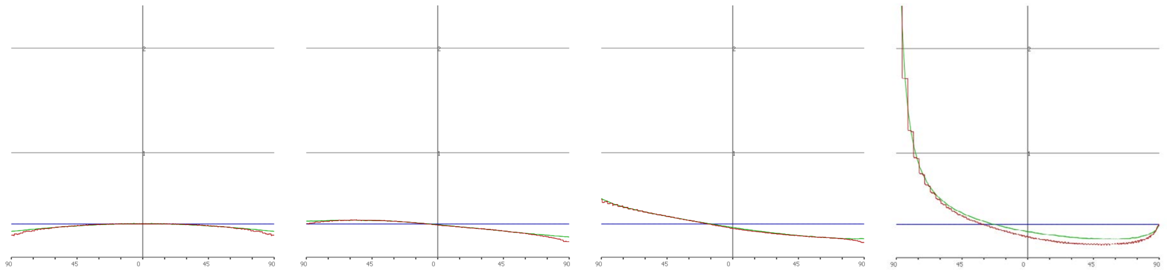

Figure 27: From left to right: Rough diffuse (gloss=0) θ_v plots, 0°, 30°, 60°, 90°. Constant blue line is Lambert, red is full BRDF table, and green is approximation using BRDF slice.

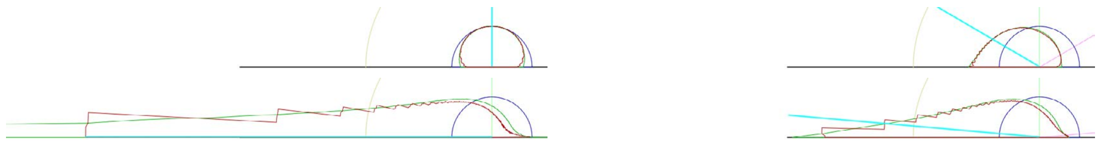

Figure 28: Clockwise from top-left: Rough diffuse (gloss=0) polar plots, incident light 0°, 60°, 85°, 90°. Cyan is incident light direction. Blue is Lambert, red is full BRDF table, and green is approximation using BRDF slice. Notice the grazing angle retroreflectivity.

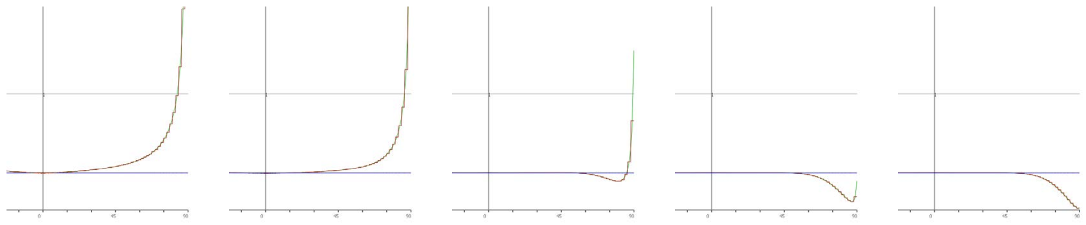

Figure 29: From left to right: θ_h plots, showing high grazing angle retroreflectivity for low gloss, gloss = $\frac{0}{255}$, $\frac{60}{255}$, $\frac{120}{255}$, $\frac{180}{255}$, $\frac{240}{255}$. Constant blue line is Lambert, red is full BRDF table, and green is approximation using BRDF slice.

5.6 BRDF Slice

Our first step was to reduce the dimensionality of the problem. Using BRDF slices is touched upon briefly in [Bur12]. It turns out that parameterizing the BRDF as a two-dimensional slice on θ_h, i.e. $\arccos(\mathbf{n} \cdot \mathbf{h})$, and θ_d, i.e. $\arccos(\mathbf{l} \cdot \mathbf{h})$, results in a good approximation to the BRDF. Most of the behavior of the BRDF comes from this slice at $\phi_d = 90$. If you view the BRDF slice in *BRDF Explorer* you will notice that this is mostly true, that other slices of the BRDF – representing other ϕ_d – are just squished versions of this slice.

The left slice in Figure 30 shows the BRDF slice of our Diffuse Multiscattering BRDF approximation. This slice representation lends itself to visual interpretation. The important characteristic of our diffuse BRDF for rough surfaces is the grazing retroreflection. The right slice in Figure 30 represents our non-cloth BRDF, including GGX specular. In that slice the left edge represents the specular peak, while the top edge represents the Fresnel peak.

Other papers have explored parameterizing other BRDFs in this way. Notably, [Pac+12] did this and then approximated the slice using a rational function. This is the strategy we initially employed – in this case, applying

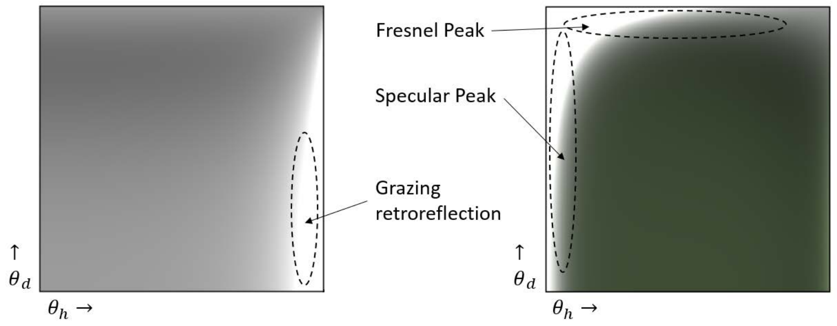

Figure 30: Left: An example BRDF slice of our Multiscattering BRDF approximation. Notice the high grazing retroreflectivity. Right: A BRDF slice of our full BRDF, including GGX specular for a moderately glossy material. Notice the specular and Fresnel peaks.

it to our Multiscattering Diffuse BRDF.

5.7 2D Rational Function Fit of the BRDF Slice

Given the effectiveness of rational functions at approximating many of the functions we encounter in rendering, our first experiments involved using rational functions to fit the BRDF slice. We performed these fits using C++ code utilizing both NLopt [Joh], and a Differential Evolution (for an overview see [PI15]) routine. This lead to almost perfect matches for gloss = 0 when we were fitting to a 2D rational function (Equation 63, and Figure 31). Unfortunately, once we added another dimension for gloss, i.e. we tried to create a function $f(\mathbf{n}\cdot\mathbf{h}, \mathbf{l}\cdot\mathbf{h}, gloss)$, the coefficient count went up dramatically if we wanted to maintain low error.

$$f(x,y) = \frac{0.568 - 0.692x - 0.938y + 1.326x^2 + 0.301y^2 + 1.96xy - 0.98x^3 + 1.533y^3}{1.0 + 1.731x - 0.997y - 0.998x^2 - 0.843y^2 + 9.905xy - 0.971x^3 + 0.948y^3}, \tag{63}$$

where $x = \mathbf{n}\cdot\mathbf{h}$ and $y = \mathbf{l}\cdot\mathbf{h}$.

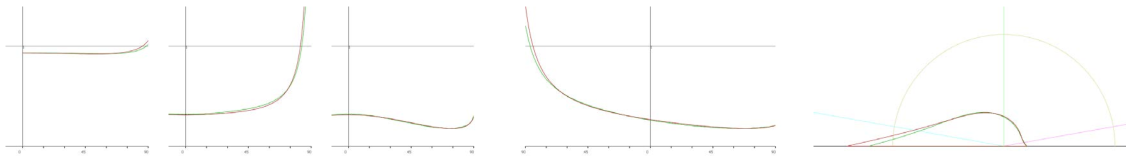

Figure 31: From left to right: albedo, θ_h, θ_d, θ_v, polar plots of Diffuse BRDF with gloss = 0. Green is BRDF slice, red is rational function fit.

5.8 Hand-Fitting the BRDF Slice

After some experimenting, we found that hand-fitting the BRDF provided good results, allowing larger errors where it is less obvious, and maintaining the important characteristics of the function:

$$f_0 = \mathbf{l}\cdot\mathbf{h} + (1 - \mathbf{l}\cdot\mathbf{h})^5, \tag{64}$$
$$f_1 = (1 - 0.75(1 - \mathbf{n}\cdot\mathbf{l})^5)(1 - 0.75(1 - \mathbf{n}\cdot\mathbf{v})^5), \tag{65}$$
$$t = \text{clamp}(2.2g - 0.5, 0, 1), \tag{66}$$
$$f_d = f_0 + (f_1 - f_0)t, \tag{67}$$
$$f_b = (34.5g^2 - 59g + 24.5)\,(\mathbf{l}\cdot\mathbf{h})\cdot 2^{-\max(73.2g-21.2,8.9)\sqrt{\mathbf{n}\cdot\mathbf{h}}}, \tag{68}$$
$$f_r = \frac{\rho}{\pi}(f_d + f_b), \tag{69}$$

where g represents gloss, f_0 represents the rough foundation when $gloss = 0$, f_1 represents the smooth diffuse BRDF, f_b is an extra retroreflective bump at grazing angles, ρ is diffuse albedo, and f_r is the final Multiscattering Diffuse BRDF approximation.

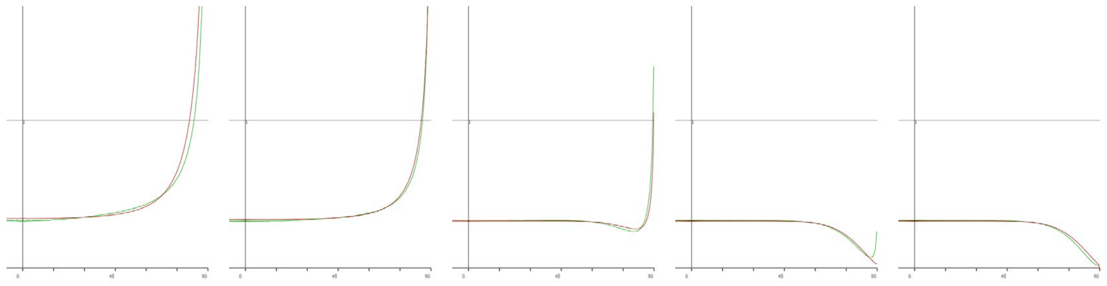

Figure 32: From left to right: θ_h plots for gloss = $\frac{0}{255}, \frac{60}{255}, \frac{120}{255}, \frac{180}{255}, \frac{240}{255}$. Green is an approximation using BRDF slice, and red is our analytical model.

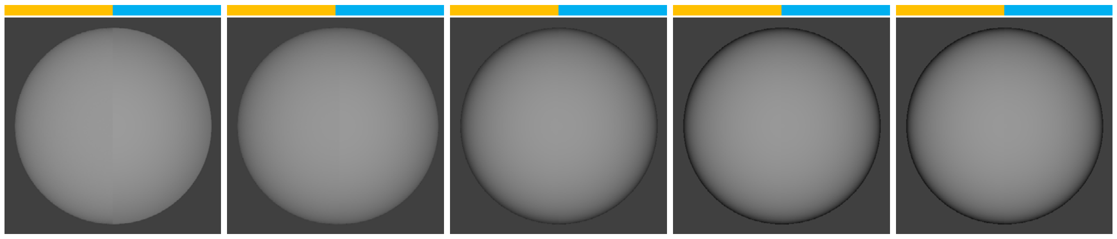

Figure 33: The left-side of the rendered sphere is the full Multiscattering Diffuse BRDF, while the right-side is the hand-fit Multiscattering Diffuse BRDF approximation. From left to right: gloss = $\frac{0}{255}, \frac{60}{255}, \frac{120}{255}, \frac{180}{255}, \frac{240}{255}$.

Figure 33 shows a comparison of the full Multiscattering Diffuse BRDF with our hand-fit approximation, while Figure 34 compares Lambertian Diffuse with our hand-fit Multiscattering Diffuse BRDF approximation.

Figure 35 shows the difference between Lambert and Multiscattering on skin. Since the underlying gloss value is low, there is a general flattening of lighting across the face with Multiscattering Diffuse. While skin seems like an unlikely candidate for this Multiscattering Diffuse model, it does appeal to our character artists, who

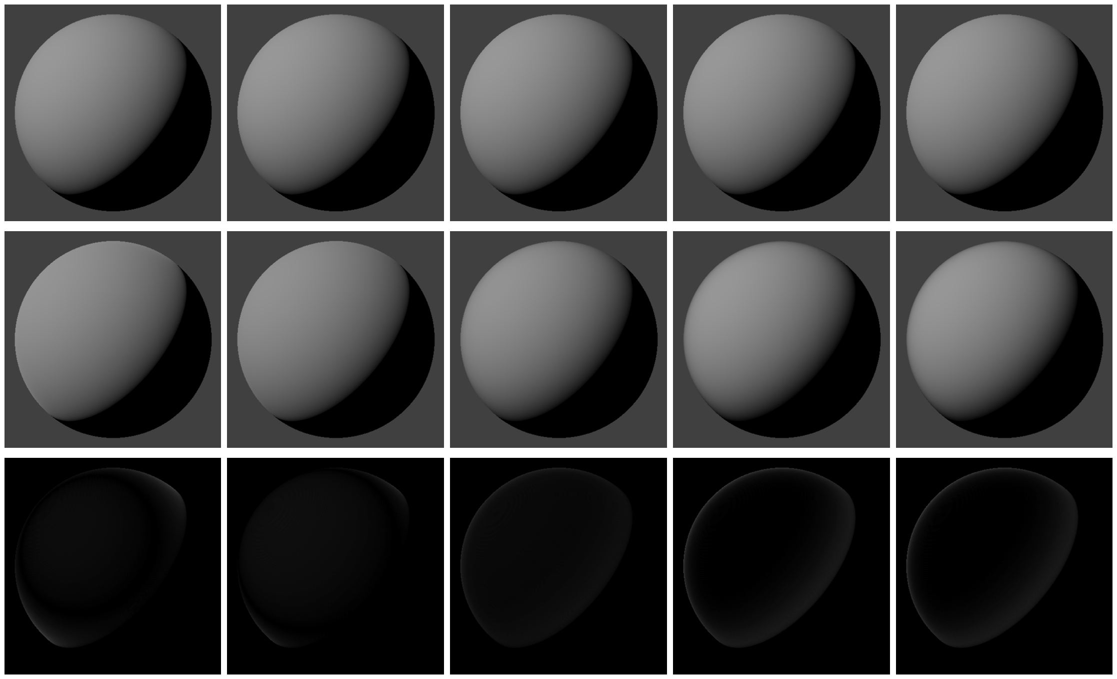

Figure 34: Top row is Lambertian diffuse, second row is our analytical fit to the BRDF slice approximation, and third row is amplified difference. From left to right: gloss = $\frac{0}{255}$, $\frac{60}{255}$, $\frac{120}{255}$, $\frac{180}{255}$, $\frac{240}{255}$.

were first to ask to use the model when it was still in its experimental stage. This does seem to fix the unnatural darkening of the face silhouette when using Lambert. We intend to validate this effect against photographs in the future.

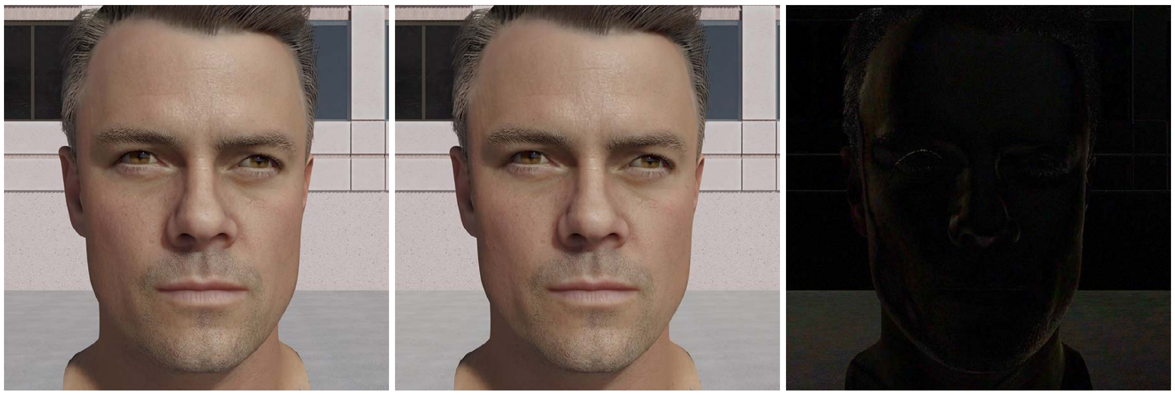

Figure 35: Example of new Multiscattering Diffuse BRDF on skin. Left is Lambertian diffuse, middle is Multiscattering diffuse, right is amplified difference. Notice Lambert diffuse exhibits an unnatural darkening of the face along the silhouette, while Multiscattering diffuse avoids this issue.

5.9 Future Directions

There are several areas in our approximation that could be improved.

We assume 100% albedo on diffuse reflectance when simulating multiscattering. If we were to add more degrees of freedom to our approximation, we could model the saturation that the [Hei+16] model achieves with rough surfaces.

We also assume 4% specular incident reflectance (F_0), a value that lies in the middle of the dielectric range. This too could be another degree of freedom to consider in order to follow the original model more closely.

We parameterize our interpolation variable t using gloss, but as we saw, errors were not evenly distributed across the gloss range (Figure 32, $gloss = \frac{180}{255}$). We could explore using α to calculate t instead.

We never performed real performance analysis of a full rational function solution against our hand-tuned approximation. Rational functions are composed of many multiplies and multiply-adds and a single divide, so can be competitive compared to a moderate number of quarter-rate instructions.

6 LIGHTING STUDIES

Figure 36 shows the result of a lighting study rendered in-game, done to mimic the lighting of well-known photographs, using our star character as the subject.

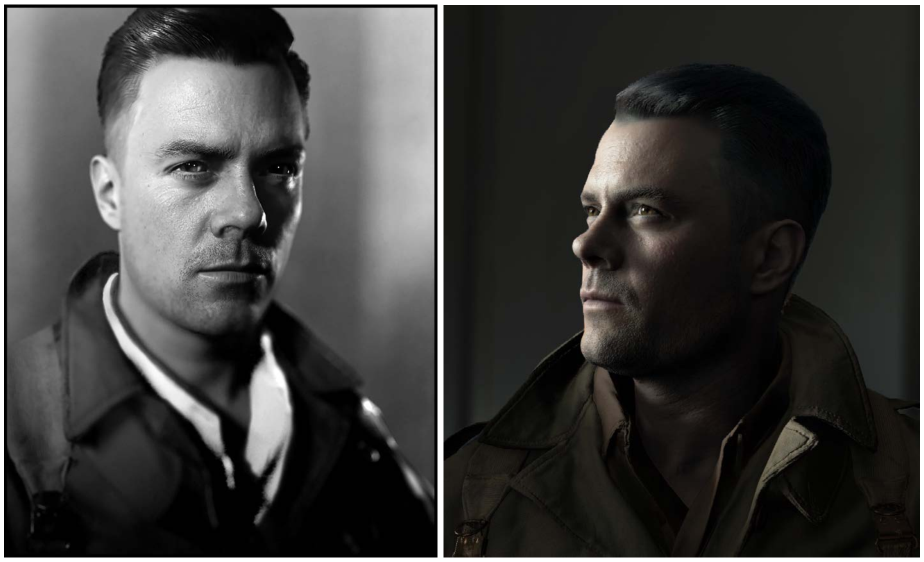

Figure 36: Lighting studies.

7 CONCLUSION

When compared against the original photographs the lighting studies were based on, the images in Figure 36 show a level of photorealism that was not quite possible in our previous project.

Several techniques were outlined which contributed to this improvement in image quality. Our method for *Mipmapping Normal and Gloss* (Section 2) ensured that surface bumpiness transitions into an appropriate look of roughness in the distance. Our technique for *Material Surface Occlusion* (Section 4) was motivated by our goal to render at "magic hour" when lighting is dominated by skylight, a lighting scenario that our rendering traditionally had trouble with. This resulted in richer shadow detail in all lighting conditions. Our *Multiscattering Diffuse BRDF* (Section 5) was motivated by our observations and measurements of real-world lighting, and importantly, ended up giving us better material differentiation through our gloss range.

8 ACKNOWLEDGMENTS

Thanks to Jorge Jiménez, Peter-Pike Sloan, Angelo Pesce, Michał Iwanicki and Paul Edelstein for discussions related to these topics. Jorge Jiménez suggested the interreflection model of Equation 49 in [SL96] based on conversations with Naty Hoffman about [Jim+16]. Michał Iwanicki suggested the mathematical formulation of shortened normal length calculation in Equation 9 that sparked further exploration of how NDFs relate to normal maps. Angelo Pesce wrote the Differential Evolution routine that was widely used in our experiments. Thanks to Stephen Hill, Stephanus, Keng Hua Sing, Brian Karis, Ari Silvennoinen and Josiah Manson for feedback, editing and notes. Stephen Hill derived the closed-form solutions to shortened normal length (Equation 24 and 26 and Listing 7). Atsushi Seo, James Wagner and Terry Barnett helped provide beautiful in-game screenshots. Special thanks to Dan Kendall and Jennifer Velazquez. Finally, thanks to Natalya Tatarchuk for organizing a great course.

REFERENCES

[Bak11] D. Baker. *Spectacular Specular - LEAN and CLEAN specular highlights.* 2011. URL: https://www.gdcvault.com/play/1014557/Spectacular-Specular-LEAN-and-CLEAN.

[Ber23] E. M. Berry. "Diffuse Reflection of Light from a Matt Surface". In: *J. Opt. Soc. Am.* 7.8 (1923), pp. 627–633. URL: http://www.osapublishing.org/abstract.cfm?URI=josa-7-8-627.

[BH12] C. Barré-Brisebois and S. Hill. *Blending in Detail.* 2012. URL: http://blog.selfshadow.com/publications/blending-in-detail/.

[BM16] W. Brinck and A. Maximov. "The Technical Art of Uncharted 4". In: *ACM SIGGRAPH Talks.* 2016. URL: http://advances.realtimerendering.com/other/2016/naughty_dog/index.html.

[Bur12] B. Burley. "Physically Based Shading at Disney". In: *Practical Physically Based Shading in Film and Game Production, ACM SIGGRAPH Courses.* 2012. URL: http://selfshadow.com/publications/s2012-shading-course/.

[Cha15] D. M. Chan. "Real-World Measurements for Call of Duty: Advanced Warfare". In: *Physically Based Shading in Theory and Practice, ACM SIGGRAPH Courses.* 2015. URL: http://blog.selfshadow.com/publications/s2015-shading-course/.

[Cha17] D. M. Chan. *Material Capture Experiments for S2.* 2017.

[Cou04] N. Cournia. *Creating Height Maps from Normal Maps.* 2004. URL: http://www.cournia.com/devnull/n2h/n2h.pdf.

[DM11] K. Dmitriev and E. Makarov. "Generating Displacement from Normal Map for use in 3D Games". In: *ACM SIGGRAPH Talks.* 2011. URL: http://developer.download.nvidia.com/assets/gamedev/docs/nmap2displacement.pdf.

[Fuj12] Y. Fujii. *A tiny improvement of Oren-Nayar reflectance model.* 2012. URL: http://mimosa-pudica.net/improved-oren-nayar.html.

[HD15] E. Heitz and J. Dupuy. *Implementing a Simple Anisotropic Rough Diffuse Material with Stochastic Evaluation.* Tech. rep. 2015. URL: https://eheitzresearch.wordpress.com/research/.

[Hei+16] E. Heitz, J. Hanika, E. d'Eon, and C. Dachsbacher. "Multiple-scattering Microfacet BSDFs with the Smith Model". In: *ACM Trans. Graph.* (2016). URL: https://eheitzresearch.wordpress.com/240-2/.

[Hei14] E. Heitz. "Understanding the Masking-Shadowing Function in Microfacet-Based BRDFs". In: *Journal of Computer Graphics Techniques (JCGT)* 3.2 (June 2014), pp. 48–107. ISSN: 2331-7418. URL: http://jcgt.org/published/0003/02/03/.

[Hil11] S. Hill. *Specular Showdown in the Wild West.* 2011. URL: http://blog.selfshadow.com/2011/07/22/specular-showdown/.

[Jak+14] W. Jakob, M. Hašan, L.-Q. Yan, J. Lawrence, R. Ramamoorthi, and S. Marschner. "Discrete Stochastic Microfacet Models". In: *ACM Trans. Graph.* (2014). URL: http://www.cs.cornell.edu/projects/stochastic-sg14/.

[Jim+16] J. Jiménez, X.-C. Wu, A. Pesce, and A. Jarabo. *Practical Realtime Strategies for Accurate Indirect Occlusion.* Tech. rep. Activision R&D, 2016.

[Joh] S. G. Johnson. *The NLopt nonlinear-optimization package.* URL: https://nlopt.readthedocs.io/en/latest/.

[Kar13] B. Karis. "Real Shading in Unreal Engine 4". In: *Physically Based Shading in Theory and Practice, ACM SIGGRAPH Courses.* 2013. URL: http://blog.selfshadow.com/publications/s2013-shading-course/.

[Kar18] B. Karis. *Normal map filtering using vMF (part 3).* 2018. URL: https://graphicrants.blogspot.com/2018/05/normal-map-filtering-using-vmf-part-3.html.

[Kol15] K. Kolasinski. *AwesomeBump v1.0.* 2015. URL: http://awesomebump.besaba.com/wp-content/uploads/2015/01/ABoverwiev.pdf.

[Löw+12] J. Löw, J. Kronander, A. Ynnerman, and J. Unger. "BRDF Models for Accurate and Efficient Rendering of Glossy Surfaces". In: *ACM Trans. Graph.* (2012). URL: http://vcl.itn.liu.se/publications/2012/LKYU12/.

[Mat+03] W. Matusik, H. Pfister, M. Brand, and L. McMillan. "A Data-driven Reflectance Model". In: *ACM Trans. Graph.* (2003).

[McA11] S. McAuley. *Energy Conserving Wrapped Diffuse.* 2011. URL: http://blog.stevemcauley.com/2011/12/03/energy-conserving-wrapped-diffuse/.

[NP13] D. Neubelt and M. Pettineo. "Crafting a Next-Gen Material Pipeline for The Order: 1886". In: *Physically Based Shading in Theory and Practice, ACM SIGGRAPH Courses.* 2013. URL: http://blog.selfshadow.com/publications/s2013-shading-course/.

[OB10] M. Olano and D. Baker. "LEAN Mapping". In: 2010. URL: https://www.csee.umbc.edu/~olano/papers/lean/.

[ON94] M. Oren and S. K. Nayar. "Generalization of Lambert's Reflectance Model". In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH '94. 1994. URL: http://www1.cs.columbia.edu/CAVE/publications/pdfs/Oren_SIGGRAPH94.pdf.

[Pac+12] R. Pacanowski, O. S. Celis, C. Schlick, X. Granier, P. Poulin, and A. Cuyt. "Rational BRDF". In: *IEEE Transactions on Visualization and Computer Graphics* (2012). URL: http://manao.inria.fr/perso/~pac/.

[PH10] M. Pharr and G. Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation.* 2nd. Morgan Kaufmann Publishers Inc., 2010.

[PI15] A. Pesce and M. Iwanicki. "Approximate Models For Physically Based Rendering". In: *Physically Based Shading in Theory and Practice, ACM SIGGRAPH Courses.* 2015. URL: http://blog.selfshadow.com/publications/s2015-shading-course/.

[Sch94] C. Schlick. "An Inexpensive BRDF Model for Physically-based Rendering". In: *Computer Graphics Forum* 13 (1994). URL: https://www.cs.virginia.edu/~jdl/bib/appearance/analytic%20models/schlick94b.pdf.

[Shi+97] P. Shirley, B. Smits, H. Hu, and E. Lafortune. "A practitioners' assessment of light reflection models". In: *Computer Graphics and Applications, 1997. Proceedings., The Fifth Pacific Conference on.* IEEE. 1997, pp. 40–49.

[SL96] A. J. Stewart and M. S. Langer. "Towards Accurate Recovery of Shape from Shading under Diffuse Lighting". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 1996. URL: http://www.cim.mcgill.ca/%7Elanger/pubs-by-year.html.

[Tok04] M. Toksvig. *Mipmapping Normal Maps.* 2004. URL: https://developer.download.nvidia.com/whitepapers/2006/Mipmapping_Normal_Maps.pdf.

[Wal+07] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. "Microfacet Models for Refraction Through Rough Surfaces". In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques.* EGSR'07. 2007.

[Wik] Wikipedia. *Nusselt analog.* URL: https://en.wikipedia.org/wiki/View_factor#Nusselt_analog.

[WNO98] L. B. Wolff, S. K. Nayar, and M. Oren. "Improved Diffuse Reflection Models for Computer Vision". In: *Int. J. Comput. Vision* (1998), pp. 55–71.

[Yan+14] L.-Q. Yan, M. Hašan, W. Jakob, J. Lawrence, S. Marschner, and R. Ramamoorthi. "Rendering Glints on High-resolution Normal-mapped Specular Surfaces". In: *ACM Trans. Graph.* (2014). URL: https://rgl.epfl.ch/publications/Yan2014Rendering.

APPENDIX

```
float Fresnel_SchlickFrac( const float u )
{
    const float oneMinusU = 1.0f - u;
    const float oneMinusUSq = oneMinusU * oneMinusU;
    return oneMinusUSq * oneMinusUSq * oneMinusU;
}

float Diffuse_Multiscattering_Primary( const float NdotL,
                                       const float NdotV,
                                       const float NdotH,
                                       const float LdotH,
                                       const float gloss )
{
    float oneMinusLdotH = 1 - LdotH;
    float oneMinusLdotH2 = oneMinusLdotH * oneMinusLdotH;
    float oneMinusLdotH4 = oneMinusLdotH2 * oneMinusLdotH2;
    float oneMinusLdotH5 = oneMinusLdotH4 * oneMinusLdotH;

    float Fd0 = ( LdotH + oneMinusLdotH5 );
    float Fd255 = lerp( 1, 0.25, Fresnel_SchlickFrac( NdotL ) ) * lerp( 1, 0.25,
        Fresnel_SchlickFrac( NdotV ) );

    float t = saturate( 2.2 * gloss - 0.5 );
    float Fd = lerp( Fd0, Fd255, t );

    // Extra retroreflective bump
    float A = 34.5 * gloss * gloss - 59 * gloss + 24.5;
    float B = max( 73.2 * gloss - 21.2, 8.9 );
    Fd += A * exp2( -B * sqrt( NdotH ) ) * LdotH;

    // Notice this does not include albedo/PI
    return Fd;
}
```

Listing 1: Multiscattering Diffuse BRDF shader.

```
float CombineGloss( float a, float b )
{
    // Fit for RNM, using our parameterization of gloss
    float p = -0.535580 + ( 1.002204 * ( a + b ) ) +
        ( -0.223910 * ( a * a + b * b ) ) + ( 13.323150 * a * b );
    float q = 1.0 + ( 8.259559 * ( a + b ) ) +
        ( 9.896132 * ( a * a + b * b ) ) + ( -22.015902 * a * b );
    return saturate( p / q );
}
```

Listing 2: Gloss combining shader.

```
float AdjustCosConeAngle( float cosConeAngle, float gloss, float NdotV )
{
    // The cone is an especially poor approximation to actual visibility for high gloss values.
    // This is an ad hoc adjustment.
    // Gloss above 0.67 is unaffected by the cone, i.e.\ we set to full cone angle.
    gloss = saturate( gloss * 1.5 );    // Optionally, omit this line.
    float gloss2 = gloss * gloss;
    float gloss4 = gloss2 * gloss2;
    float gloss8 = gloss4 * gloss4;
    float oneMinusNdotV2 = ( 1 - NdotV ) * ( 1 - NdotV );
    float oneMinusNdotV4 = oneMinusNdotV2 * oneMinusNdotV2;
    // We lerp towards full cone angle based on gloss and NdotV.
    cosConeAngle = lerp( 0, cosConeAngle, ( 1 - gloss8 ) * ( 1 - oneMinusNdotV4 ) );
    return cosConeAngle;
}
```

Listing 3: Specular occlusion cone adjustment.

```
float GetVisibilityFromMaterialOcclusion( float materialOcclusion,
                                          float gloss,
                                          float NdotV,
                                          const float3 worldNormal,
                                          const float3 lightDir )
{
    float NdotL = dot( worldNormal.xyz, lightDir.xyz );
    float vis = materialOcclusion;
    float cosConeAngle = sqrt( 1 - vis );
    float adjustedCosConeAngle = AdjustCosConeAngle( cosConeAngle, gloss, NdotV );
    adjustedCosConeAngle = max( adjustedCosConeAngle, 0.001 );
    vis = saturate( NdotL / adjustedCosConeAngle );
    return vis * vis;
}
```

Listing 4: Light visibility from material surface occlusion.

```
#define ENV_BRDF_SCALE      0
#define ENV_BRDF_OFFSET     1

float3 Fresnel_Indirect( const float NdotV,
                         const float cosConeAngle,
                         const float3 F0,
                         const float gloss )
{
    float2 envBrdf;
    const float ENV_BRDF_XY_RES = 32.0;
    const float ENV_BRDF_Z_RES = 8.0;
    const float tcXYScale = ( ( ENV_BRDF_XY_RES - 1.0 ) / ENV_BRDF_XY_RES );
    const float tcXYOffset = rcp( 2.0 * ENV_BRDF_XY_RES );
    const float tcZScale = ( ( ENV_BRDF_Z_RES - 1.0 ) / ENV_BRDF_Z_RES );
    const float tcZOffset = rcp( 2.0 * ENV_BRDF_Z_RES );
    float4 texcoord;

    texcoord.xy = float3( NdotV, gloss, cosConeAngle ) * tcXYScale + tcXYOffset;
    texcoord.z = cosConeAngle * tcZScale + tcZOffset;
    texcoord.w = 0;
    envBrdf = envBrdfConeLut.SampleLevel( linearClampSampler, texcoord.xyz, texcoord.w ).rg;
    return F0 * envBrdf[ENV_BRDF_SCALE] + envBrdf[ENV_BRDF_OFFSET];
}

float3 Fresnel_Diffuse_Direct( const float NdotL,
                               const float3 specColor,
                               const float gloss )
{
    float cosConeAngle = 0;
    // BRDFs are reciprocal so we can pass in NdotL instead of NdotV and get
    // the energy scattered due to specular reflection. To keep things simple,
    // we assume the point is fully visible (cosConeAngle = 0).
    return 1 - Fresnel_Indirect( NdotL, cosConeAngle, specColor, gloss );
}
```

Listing 5: Energy conservation.

```
float3 GGX_ImportanceSample( const float2 qr, const float alpha )
{
    float alpha2 = alpha * alpha;
    float phi = 2 * M_PI * qr.x;
    float cosTheta = sqrt( ( 1 - qr.y ) / ( 1 + ( alpha2 - 1 ) * qr.y ) );
    float sinTheta = sqrt( 1 - cosTheta * cosTheta );
    float3 H;

    H.x = sinTheta * cos( phi );
    H.y = sinTheta * sin( phi );
    H.z = cosTheta;

    return H;
}

float GlossToAverageNormalLength( float gloss )
{
    float alpha = GGX_AlphaFromGloss( gloss );
    float3 averageNormal = float3( 0.0, 0.0, 0.0 );
    for ( int sampleIndex = 0; sampleIndex < sampleCount; sampleIndex++ )
    {
        // GetQuasiRandomFloat2( seed ) could be a Low-Discrepancy Sequence
        // e.g. returns float2( HaltonSequence( seed, 2 ), HaltonSequence( seed, 3 );
        float2 qr = GetQuasiRandomFloat2( sampleIndex );
        float3 H = GGX_ImportanceSample( qr, alpha );
        averageNormal += H;
    }
    averageNormal /= sampleCount;
    return length( averageNormal );
}
```

Listing 6: Gloss to average normal length (GGX) using Macrosurface Area model by Importance Sampling.

```
float AlphaToMicroNormalLength(float alpha)
{
    if (alpha == 0.0f) return 1.0f;
    if (alpha == 1.0f) return 0.5f;

    float a = sqrtf(1.0f - alpha*alpha);

    return a / (a + (1 - a*a)*atanhf(a));
}

float AlphaToMacroNormalLength(float alpha)
{
    if (alpha == 0.0f) return 1.0f;
    if (alpha == 1.0f) return 2.0f / 3.0f;

    float a = sqrtf(1.0f - alpha*alpha);

    return (a - (1 - a*a)*atanhf(a)) / (a*a*a);
}
```

Listing 7: Closed-form solutions for alpha to average normal length (GGX).

```
// Each entry is {gloss, normalLength}:
glossToNormalLength = {{0.000000, 0.666670}, {0.003922, 0.669966}, {0.007843, 0.673329},
 {0.011765, 0.676758}, {0.015686, 0.680251}, {0.019608, 0.683807}, {0.023529, 0.687424},
 {0.027451, 0.691099}, {0.031373, 0.694832}, {0.035294, 0.698620}, {0.039216, 0.702459},
 {0.043137, 0.706349}, {0.047059, 0.710286}, {0.050980, 0.714267}, {0.054902, 0.718291},
 {0.058824, 0.722353}, {0.062745, 0.726451}, {0.066667, 0.730582}, {0.070588, 0.734743},
 {0.074510, 0.738930}, {0.078431, 0.743141}, {0.082353, 0.747373}, {0.086275, 0.751621},
 {0.090196, 0.755882}, {0.094118, 0.760155}, {0.098039, 0.764434}, {0.101961, 0.768717},
 {0.105882, 0.773001}, {0.109804, 0.777282}, {0.113725, 0.781557}, {0.117647, 0.785823},
 {0.121569, 0.790078}, {0.125490, 0.794317}, {0.129412, 0.798539}, {0.133333, 0.802739},
 {0.137255, 0.806916}, {0.141176, 0.811067}, {0.145098, 0.815189}, {0.149020, 0.819279},
 {0.152941, 0.823336}, {0.156863, 0.827357}, {0.160784, 0.831339}, {0.164706, 0.835281},
 {0.168627, 0.839181}, {0.172549, 0.843036}, {0.176471, 0.846846}, {0.180392, 0.850607},
 {0.184314, 0.854320}, {0.188235, 0.857982}, {0.192157, 0.861592}, {0.196078, 0.865149},
 {0.200000, 0.868652}, {0.203922, 0.872099}, {0.207843, 0.875491}, {0.211765, 0.878826},
 {0.215686, 0.882103}, {0.219608, 0.885321}, {0.223529, 0.888482}, {0.227451, 0.891583},
 {0.231373, 0.894625}, {0.235294, 0.897607}, {0.239216, 0.900530}, {0.243137, 0.903393},
 {0.247059, 0.906196}, {0.250980, 0.908939}, {0.254902, 0.911623}, {0.258824, 0.914248},
 {0.262745, 0.916814}, {0.266667, 0.919321}, {0.270588, 0.921770}, {0.274510, 0.924161},
 {0.278431, 0.926494}, {0.282353, 0.928771}, {0.286275, 0.930992}, {0.290196, 0.933157},
 {0.294118, 0.935267}, {0.298039, 0.937324}, {0.301961, 0.939326}, {0.305882, 0.941276},
 {0.309804, 0.943175}, {0.313726, 0.945022}, {0.317647, 0.946818}, {0.321569, 0.948566},
 {0.325490, 0.950265}, {0.329412, 0.951916}, {0.333333, 0.953520}, {0.337255, 0.955079},
 {0.341176, 0.956592}, {0.345098, 0.958061}, {0.349020, 0.959487}, {0.352941, 0.960871},
 {0.356863, 0.962214}, {0.360784, 0.963516}, {0.364706, 0.964779}, {0.368627, 0.966002},
 {0.372549, 0.967189}, {0.376471, 0.968338}, {0.380392, 0.969452}, {0.384314, 0.970530},
 {0.388235, 0.971574}, {0.392157, 0.972585}, {0.396078, 0.973564}, {0.400000, 0.974511},
 {0.403922, 0.975427}, {0.407843, 0.976313}, {0.411765, 0.977171}, {0.415686, 0.978000},
 {0.419608, 0.978801}, {0.423529, 0.979576}, {0.427451, 0.980324}, {0.431373, 0.981047},
 {0.435294, 0.981746}, {0.439216, 0.982421}, {0.443137, 0.983073}, {0.447059, 0.983703},
 {0.450980, 0.984311}, {0.454902, 0.984897}, {0.458824, 0.985464}, {0.462745, 0.986010},
 {0.466667, 0.986537}, {0.470588, 0.987046}, {0.474510, 0.987537}, {0.478431, 0.988010},
 {0.482353, 0.988466}, {0.486275, 0.988906}, {0.490196, 0.989330}, {0.494118, 0.989739},
 {0.498039, 0.990133}, {0.501961, 0.990513}, {0.505882, 0.990879}, {0.509804, 0.991232},
 {0.513726, 0.991571}, {0.517647, 0.991898}, {0.521569, 0.992213}, {0.525490, 0.992517},
 {0.529412, 0.992809}, {0.533333, 0.993090}, {0.537255, 0.993361}, {0.541176, 0.993622},
 {0.545098, 0.993873}, {0.549020, 0.994114}, {0.552941, 0.994346}, {0.556863, 0.994570},
 {0.560784, 0.994785}, {0.564706, 0.994992}, {0.568627, 0.995191}, {0.572549, 0.995382},
 {0.576471, 0.995567}, {0.580392, 0.995744}, {0.584314, 0.995914}, {0.588235, 0.996078},
 {0.592157, 0.996235}, {0.596078, 0.996386}, {0.600000, 0.996531}, {0.603922, 0.996671},
 {0.607843, 0.996806}, {0.611765, 0.996935}, {0.615686, 0.997059}, {0.619608, 0.997178},
 {0.623529, 0.997292}, {0.627451, 0.997402}, {0.631373, 0.997508}, {0.635294, 0.997609},
 {0.639216, 0.997707}, {0.643137, 0.997801}, {0.647059, 0.997890}, {0.650980, 0.997977},
 {0.654902, 0.998060}, {0.658824, 0.998139}, {0.662745, 0.998216}, {0.666667, 0.998289},
 {0.670588, 0.998360}, {0.674510, 0.998427}, {0.678431, 0.998492}, {0.682353, 0.998554},
 {0.686275, 0.998614}, {0.690196, 0.998671}, {0.694118, 0.998726}, {0.698039, 0.998779},
 {0.701961, 0.998830}, {0.705882, 0.998879}, {0.709804, 0.998925}, {0.713726, 0.998970},
 {0.717647, 0.999013}, {0.721569, 0.999054}, {0.725490, 0.999094}, {0.729412, 0.999132},
 {0.733333, 0.999168}, {0.737255, 0.999203}, {0.741176, 0.999236}, {0.745098, 0.999268},
 {0.749020, 0.999299}, {0.752941, 0.999328}, {0.756863, 0.999357}, {0.760784, 0.999384},
 {0.764706, 0.999410}, {0.768627, 0.999435}, {0.772549, 0.999459}, {0.776471, 0.999481},
 {0.780392, 0.999503}, {0.784314, 0.999524}, {0.788235, 0.999545}, {0.792157, 0.999564},
 {0.796078, 0.999582}, {0.800000, 0.999600}, {0.803922, 0.999617}, {0.807843, 0.999633},
 {0.811765, 0.999649}, {0.815686, 0.999664}, {0.819608, 0.999678}, {0.823529, 0.999692},
 {0.827451, 0.999705}, {0.831373, 0.999718}, {0.835294, 0.999730}, {0.839216, 0.999741},
 {0.843137, 0.999753}, {0.847059, 0.999763}, {0.850980, 0.999773}, {0.854902, 0.999783},
 {0.858824, 0.999792}, {0.862745, 0.999801}, {0.866667, 0.999810}, {0.870588, 0.999818},
 {0.874510, 0.999826}, {0.878431, 0.999833}, {0.882353, 0.999841}, {0.886275, 0.999848},
 {0.890196, 0.999854}, {0.894118, 0.999860}, {0.898039, 0.999866}, {0.901961, 0.999872},
 {0.905882, 0.999878}, {0.909804, 0.999883}, {0.913725, 0.999888}, {0.917647, 0.999893},
 {0.921569, 0.999898}, {0.925490, 0.999902}, {0.929412, 0.999906}, {0.933333, 0.999911},
 {0.937255, 0.999914}, {0.941176, 0.999918}, {0.945098, 0.999922}, {0.949020, 0.999925},
 {0.952941, 0.999929}, {0.956863, 0.999932}, {0.960784, 0.999935}, {0.964706, 0.999938},
 {0.968627, 0.999940}, {0.972549, 0.999943}, {0.976471, 0.999945}, {0.980392, 0.999948},
 {0.984314, 0.999950}, {0.988235, 0.999952}, {0.992157, 0.999954}, {0.996078, 0.999956},
 {1.000000, 0.999958}};
```

Listing 8: Gloss to average normal length (GGX) table for our parameterization of gloss. We generated this table using importance sampling of the macrosuface-area GGX NDF.