Crafting Physically Motivated Shading Models for Game Development

by Naty Hoffman

In this section of the course notes, we discuss the design of shading models that are both physically based and appropriate for game development use.

Motivation and Infrastructure

Motivation

The first question many game developers ask in connection with physically-based shading models is "Why bother?". This is a valid question, since games do not aim at an exact physical simulation of light transport[1]. However, we shall see many practical advantages for games in adopting these models.

With shading models that are based on physical principles, it is easier to achieve photorealistic and cinematic looks. Objects that use such shading models retain their basic appearance when the lighting and viewing conditions change; they have a robustness that is often not afforded by ad-hoc shading hacks. Art asset creation is also easier; less "slider tweaking" and adjustment of "fudge factors" is needed to achieve high visual quality, and the material interface exposed to the artists is simple, powerful and expressive.

For graphics programmers and shader writers, physically based shaders are easier to troubleshoot. When something appears too bright, too dark, too green, too shiny, etc. then it is much easier to reason about what is happening in the shader when the various terms and parameters have a physical meaning. It is also easier to extend such shaders to add new features, since physical reasoning can be used to determine e.g., which subexpression in the shader should be affected by ambient occlusion, or how an environment map should be combined with existing shading terms.

There have been several articles in the press over the last few years [28, 29, 30] highlighting these advantages in the case of film production; most of the content of these articles applies equally to game development.

Infrastructure

There are several basic features a game rendering engine needs to have to get the most benefit from physically-based shading models. Shading needs to occur in linear space, with inputs and outputs

[1] Rendering applications which do have exact simulation as their goal are called *predictive rendering* applications, since they aim to predict the image that would have been created if the scene and lighting environment existed in the real world. Such applications include architectural previsualization and some types of CAD (Computer-Aided Design).

correctly transformed (*gamma-correct rendering*), the engine should have some support for lighting and shading values with high dynamic range (*HDR*), and there needs to be an appropriate transformation from scene to display colors (*tone mapping*).

Gamma-Correct Rendering

When artists author shader inputs such as textures, light colors, and vertex colors, a *non-linear encoding* is typically used for numerical representation and storage. This means that physical light intensities are not linearly proportional to numerical values. Pixel values stored in the frame buffer after rendering use similar nonlinear encodings.

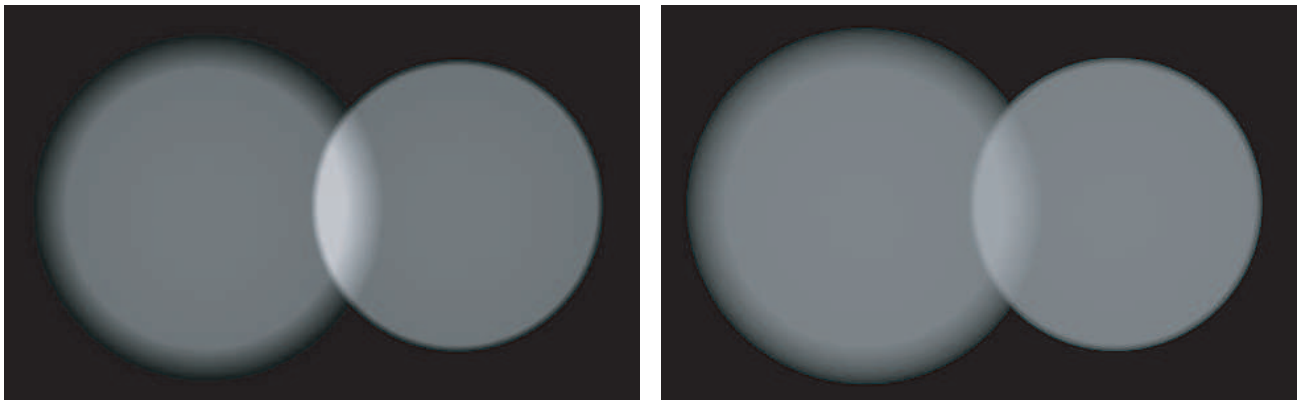

Figure 1: This figure shows a grey flat surface illuminated by two overlapping spotlights. In the left image, shading computations have been performed on nonlinear (sRGB) encoded values, so the addition of the two lights in the overlapping region results in an overly bright region that does not correspond to the expected brightness from summing the two lights. On the right the shading computations are performed on linearly encoded values, and the result appears correct. (*image from "Real-Time Rendering, 3rd edition" used with permission from A K Peters*)

These encodings are primarily used to make efficient use of limited bit precision. Although such nonlinear encodings have steps between successive integer values which are not *physically uniform* (the amount of light energy added at each step varies over the range), they are (somewhat) *perceptually uniform* (the perceived change in brightness at each step does not vary much over the range). This allows for fewer bits to be used without banding.

The two nonlinear encodings most commonly used in game graphics are *sRGB* (used by computer monitors) and *ITU-R Recommendation BT.709* (used by HDTV displays). The official sRGB specification is the IEC 61966-2-1:1999 standard, a working draft of which is available online [18]. Both standards are described in detail elsewhere on the Internet, and in a comprehensive book on video encoding by Charles Poynton [26].

Since shading inputs and outputs use nonlinear encodings, then by default shading computations will be performed on nonlinearly encoded values, which is incorrect and can lead to "$1 + 1 = 3$" situations such as the one shown in Figure 1. To avoid this, shading inputs need to be converted to linear values, and the shader output needs to be converted to the appropriate nonlinear encoding. In principle, these conversions can be done in hardware by the GPU (for textures and render targets) or in a post-process (for shader constants and vertex colors). However, if blending operations are performed in the frame buffer, then difficulties may ensue with platforms like the Playstation 3, which performs alpha blending incorrectly with sRGB render targets. sRGB texture filtering is also not always performed correctly. However, MIP-maps can (and should) always be generated in the correct space. Problems are also caused by the fact that the exact nonlinear encoding varies from platform to platform, especially in the case of consoles.

Although converting a game engine to linear shading generally improves visuals, there are often unintended consequences that need to be addressed. Light distance falloff, Lambert falloff, spotlight angular falloff, soft shadow edge feathering, vertex interpolation all will appear differently now that they are happening in linear space. This may require some retraining or readjustment by the artists, and a few rare cases (like vertex interpolation) might need to be fixed in the shader.

More details on how to convert a game engine to be gamma-correct can be found online [10, 16, 35].

HDR Values

Realistic rendering requires handling much higher intensity values than display maximum white (this maximum is typically mapped to a pixel intensity of 1.0). Values with a much larger range than the display range are referred to as *High Dynamic Range* (HDR) values. HDR values are needed before shading (e.g., lightmaps, environment maps) and shading can also produce such values (e.g., specular highlights). Although these values cannot be displayed directly, they can still affect the final image via effects such as bloom, fog, depth of field and motion blur.

The most straightforward way to handle HDR values is to store them in a wide format with 16 or even 32 bits per channel. However, such wide formats can be prohibitively expensive to use for textures and render targets on current-generation consoles.

One popular solution is to use some type of compressed encoding to store HDR values in low-precision (typically 8 bits per channel) render targets [8, 20]. A second approach is to render multiple (typically two) exposures into different render targets [35]. Unlike compressed encodings, this option has the advantage of hardware blending and filtering support. The cheapest approach is to tone-map to an LDR buffer at the end of the pixel shader. This approach is typically combined with hacks to extract bloom masks out of LDR data. Careful scaling of HDR values to fit in low-precision render targets can improve the results of this approach [19]. Textures such as light maps and environment maps can also be scaled to fit in low-precision texture formats. With careful management of lighting and exposure, ranges greater than $25 - 100$ times display white are rarely needed. In sRGB space this corresponds to just a $4 - 8$ range, which can fit well in 10-bit-per-channel textures (8-bit-per-channel or even DXT in a pinch). Giving artists manual control over the exposure often works better than a more automatic approach.

Tone Mapping

Tone Mapping is the process of converting HDR scene intensity values to display intensities in a perceptually convincing manner. It is common in computer graphics to do with with a smooth curve of some kind [27]. The most effective curves are derived from film emulsion characteristics; these "S-shaped" curves produce pleasing and realistic images. Another term for this mapping "from scene to screen" is *color rendering*. The notes from a recent SIGGRAPH course [17] go into detail on color rendering practice in game and film production; a GDC presentation [11] and subsequent blog posts [12, 13, 14] by John Hable discuss how filmic tone mapping curves were used in *Uncharted 2: Among Thieves*.

Making an Ad-hoc Game Shading Model Physically Plausible

The initial generations of graphics accelerators did not have programmable shaders, and imposed a fixed-function shading model on games for several years. Once programmable shading was introduced, game developers were used to the fixed-function models and often extended them instead of developing new models from scratch. For this reason, many physically incorrect properties of the old fixed-function model persist in common usage today.

Figure 2: Conditionally setting the specular term to 0 when the light is behind the surface can introduce distracting discontinuities (*image from "Real-Time Rendering, 3rd edition" used with permission from A K Peters*).

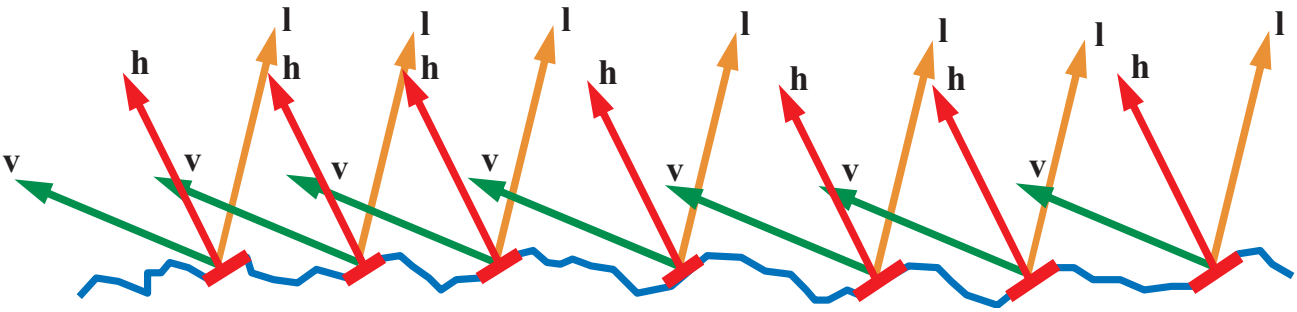

Figure 3: Microfacets with $\mathbf{m} = \mathbf{h}$ are oriented to reflect $\mathbf{l}$ into $\mathbf{v}$—other microfacets do not contribute to the BRDF. (*image from "Real-Time Rendering, 3rd edition" used with permission from A K Peters*).

We will start with a fairly representative game shading model based on Phong's original model [25]. We will show here the equation for a single punctual light source (note that a game will typically have multiple punctual lights and additional terms for ambient light, environment maps, etc.):

$$L_o(\mathbf{v}) = \left(\mathbf{c}_{\text{diff}}\underline{(\mathbf{n}\cdot\mathbf{l_c})} + \begin{cases}\mathbf{c}_{\text{spec}}\underline{(\mathbf{r_v}\cdot\mathbf{l_c})}^{\alpha_p}, & \text{if } (\mathbf{n}\cdot\mathbf{l_c}) > 0\\ 0, & \text{otherwise}\end{cases}\right) \otimes \mathbf{c}_{\text{light}}. \tag{1}$$

The notation is the same as in the background talk: $L_o(\mathbf{v})$ is the outgoing radiance in the view direction, $\mathbf{v}$ is the view vector, $\mathbf{c}_{\text{diff}}$ is the diffuse color, $\mathbf{n}$ is the normal vector, $\mathbf{l_c}$ is the punctual light's direction vector, $\mathbf{c}_{\text{spec}}$ is the specular color, α_p is the specular power, $\mathbf{c}_{\text{light}}$ is the punctual light color, $\otimes$ denotes RGB vector multiplication, and the line under the dot product $\underline{(\mathbf{n}\cdot\mathbf{l_c})}$ is the notation for clamping to 0. There is one new vector— $\mathbf{r_v}$ is the view vector reflected about the normal.

Like the clamp on the diffuse dot product, the conditional on the specular term is there to remove the contributions of punctual lights behind the surface. However, this conditional does not make physical sense and worse, can introduce severe artifacts (see Figure 2).

We will modify the shader to avoid specular from backfacing lights in a different way. Instead of a conditional, we will multiply the specular term by $\underline{(\mathbf{n}\cdot\mathbf{l_c})}$. This makes sense since this cosine term is not actually part of the BRDF, but of the rendering equation. Recall the punctual light rendering

equation from the background talk in this course:

$$L_o(\mathbf{v}) = \pi f(\mathbf{l_c}, \mathbf{v}) \otimes \mathbf{c}_{\text{light}} \underline{(\mathbf{n} \cdot \mathbf{l_c})}. \tag{2}$$

After replacing the conditional with multiplication by the cosine term, we get the following equation, which is simpler, faster to compute, and does not suffer from discontinuity artifacts:

$$L_o(\mathbf{v}) = \left(\mathbf{c}_{\text{diff}} + \mathbf{c}_{\text{spec}} \underline{(\mathbf{r_v} \cdot \mathbf{l_c})}^{\alpha_p}\right) \otimes \mathbf{c}_{\text{light}} \underline{(\mathbf{n} \cdot \mathbf{l_c})}. \tag{3}$$

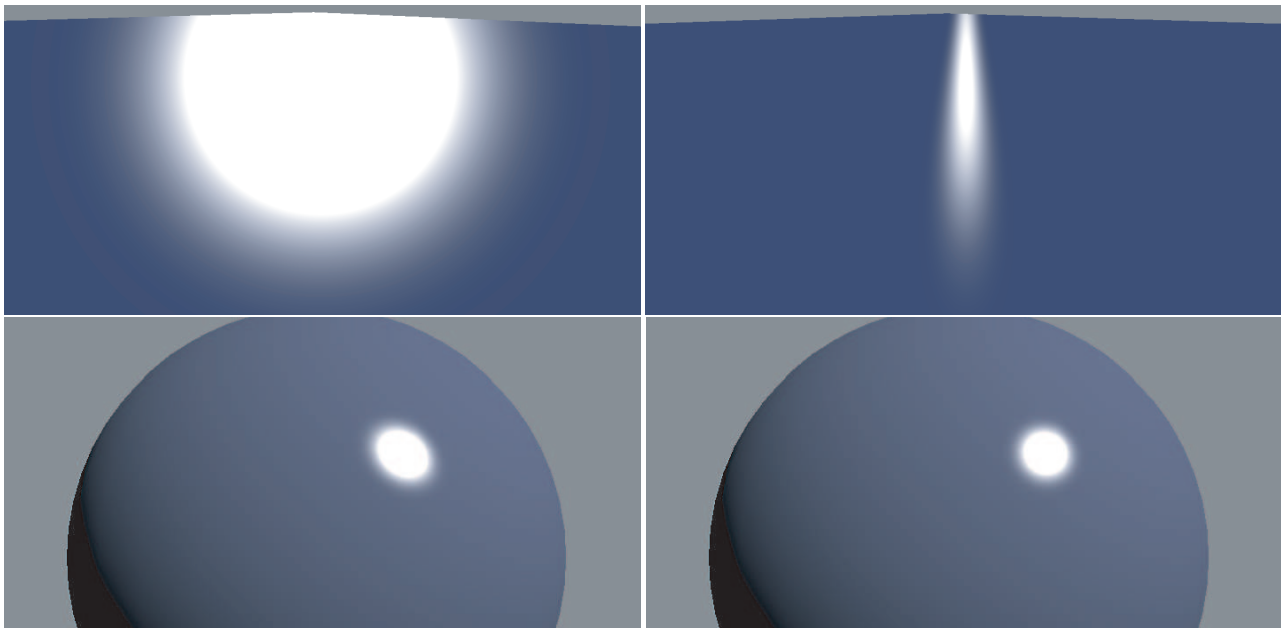

Figure 4: On the left, we see two scenes rendered with the original Phong shading model. On the right, we see the same scenes rendered with the Blinn-Phong model. Although the differences are subtle on the bottom row (sphere), they are very noticeable on the top row (flat plane). (*image from "Real-Time Rendering, 3rd edition" used with permission from A K Peters*).

Let's now focus on the specular term. What is the physical meaning of the dot product between the reflected view vector and the light? It doesn't seem to correspond to anything from microfacet theory. Blinn's modification [2] to the Phong model (typically referred to as the *Blinn-Phong model*) is very similar to Equation 3, but it uses the more physically meaningful half-vector. Recall (from the background talk): the half-vector is the direction to which the microfacet normals $\mathbf{m}$ need be oriented to reflect $\mathbf{l}$ into $\mathbf{v}$ (see Figure 3)—the reflection vector has no such physical significance. Changing from Phong to Blinn-Phong gives us the following model:

$$L_o(\mathbf{v}) = \underline{(\mathbf{n} \cdot \mathbf{h})}^{\alpha_p} \mathbf{c}_{\text{spec}} \otimes \mathbf{c}_{\text{light}} \underline{(\mathbf{n} \cdot \mathbf{l_c})}. \tag{4}$$

Although Blinn-Phong is more physically meaningful than the original Phong, it is valid to ask whether this makes any practical difference for production shading. Figure 4 compares the visual appearance of the two models. For round objects the two are similar, but for lights glancing off flat surfaces like floors, they are very different. Phong has a round highlight and Blinn-Phong has an elongated thin highlight. If we compare to real-world photographs (Figure 5) then it is clear that Blinn-Phong is much more realistic.

So far using microfacet theory to improve our game shading model has been successful. Let's try some more microfacet theory, starting by comparing our current shading model with a microfacet BRDF model lit by a punctual light source:

Figure 5: The real world displays elongated thin highlights, similar to those predicted by the Blinn-Phong model and very different from those predicted by the original Phong model. (*photographs from "Real-Time Rendering, 3rd edition" used with permission from A K Peters and the photographer, Elan Ruskin*).

$$L_o(\mathbf{v}) = \pi \frac{D(\mathbf{h})\, G(\mathbf{l_c}, \mathbf{v}, \mathbf{h})}{4(\mathbf{n}\cdot\mathbf{l_c})\,(\mathbf{n}\cdot\mathbf{v})}\, F(\mathbf{l_c}, \mathbf{h}) \otimes \mathbf{c}_{\text{light}}(\mathbf{n}\cdot\mathbf{l_c}) \tag{5}$$

$$L_o(\mathbf{v}) = (\mathbf{n}\cdot\mathbf{h})^{\alpha_p}\, \mathbf{c}_{\text{spec}} \otimes \mathbf{c}_{\text{light}}(\mathbf{n}\cdot\mathbf{l_c}).$$

There appear to already be several important similarities; we have highlighted the parts that correspond most closely with matching colors. What are the minimal changes required to turn our model into a full-fledged microfacet BRDF?

First, we see that the cosine power term already resembles a microfacet distribution function evaluated with $\mathbf{m} = \mathbf{h}$. However, to convert the cosine power term into a microfacet distribution function it must be correctly normalized. Any microfacet distribution needs to fulfill the requirement that the sum of the microfacet areas is equal to the macrosurface area. More precisely, the sum of the signed projected areas of the microfacets needs to equal the signed projected area of the macroscopic surface; this must hold true for any viewing direction [37]. Mathematically, this means that the function must fulfil this equation for any $\mathbf{v}$:

$$(\mathbf{v}\cdot\mathbf{n}) = \int_{\Theta} D(\mathbf{m})(\mathbf{v}\cdot\mathbf{m})\, d\omega_m. \tag{6}$$

Note that the integral is over the entire sphere, not just the hemisphere, and the cosine factors are not clamped. This equation holds for any kind of microsurface, not just heightfields. In the special case, $\mathbf{v} = \mathbf{n}$:

$$1 = \int_{\Theta} D(\mathbf{m})(\mathbf{n}\cdot\mathbf{m})\, d\omega_m. \tag{7}$$

The Blinn-Phong cosine power term can be made to obey this equation by multiplying it with a simple normalization factor:

$$D_{\text{BP}}(\mathbf{m}) = \frac{\alpha_p + 2}{2\pi}(\mathbf{n}\cdot\mathbf{m})^{\alpha_p}. \tag{8}$$

The next term that needs to be modified is $\mathbf{c}_{\text{spec}}$. As we saw in the background talk, although the specular reflectance of a given material stays almost constant over a wide range of directions, it always goes to 100% white at extremely glancing angles. This effect can be simply modeled by replacing $\mathbf{c}_{\text{spec}}$ with $F_{\text{Schlick}}(\mathbf{c}_{\text{spec}}, \mathbf{l}, \mathbf{h})$ (the background talk course notes give more details on the Schlick approximation). Not all games use a constant $\mathbf{c}_{\text{spec}}$ as in our example "game shading model". Many games do use the Schlick approximation for Fresnel, but unfortunately it is often used incorrectly. The most common error is to use the Schlick equation to interpolate a scalar "Fresnel factor" to 1 instead of interpolating $\mathbf{c}_{\text{spec}}$ to 1. This interpolated "Fresnel factor" is then multiplied with $\mathbf{c}_{\text{spec}}$. This is bad for several reasons. Instead of interpolating the surface specular color to white at the edges, this "Fresnel term" instead darkens it at the center. The artist has to specify the edge color instead of the much more intuitive center color, and in the case of colored specular there is no way to get the correct result. Worse still, the superfluous "Fresnel factor" parameter is added to the ones the artist needs to manipulate, sometimes even stored per-pixel in a texture, wasting storage space. It is true that this "Fresnel model" is slightly cheaper to compute than the correct one, but given the lack of realism and the awkwardness for the artist, the tiny performance difference is not worth it.

Another common error is to use the wrong angle for the Fresnel term. Both environment maps and specular highlights require that the specular color be modified by a Fresnel term, but it is not the same term in both cases. The appropriate angle to use when computing environment map Fresnel is the one between $\mathbf{n}$ and $\mathbf{v}$, while the angle to use for specular highlight Fresnel is the one between $\mathbf{l}$ and $\mathbf{h}$ (or equivalently, between $\mathbf{v}$ and $\mathbf{h}$). This is because specular highlights are reflected by microfacets with surface normals equal to $\mathbf{h}$. Either out of unfamiliarity with the underlying theory or out of temptation to save a few cycles, it is common for developers to use the angle between $\mathbf{n}$ and $\mathbf{v}$ for both environment map Fresnel and specular highlight Fresnel. This temptation should be resisted—when this angle is used for specular highlight Fresnel, any surface which is glancing to the view direction will receive brightened highlights regardless of light direction. This will lead to overly bright highlights throughout the scene, often forcing the use of some hack factor to darken the highlights back down and dooming any chance of achieving realistic specular reflectance.

Looking back at Equation 5, we see that part of the microfacet model has no corresponding term in our modified game specular model. This "orphan term" is the shadowing / masking, or geometry term $G(\mathbf{l_c}, \mathbf{v}, \mathbf{h})$ divided by the "foreshortening factors" $\underline{(\mathbf{n} \cdot \mathbf{l_c})}\,\underline{(\mathbf{n} \cdot \mathbf{v})}$. We refer to this ratio as the *visibility term* since it combines factors accounting for microfacet self-occlusion and foreshortening. Since our modified specular model has no visibility term, we will simply set it to 1. This is the same as setting the geometry term to be equal to the product of the two foreshortening factors, defining the following *implicit geometry term*:

$$G_{\text{implicit}}(\mathbf{l_c}, \mathbf{v}, \mathbf{h}) = \underline{(\mathbf{n} \cdot \mathbf{l_c})}\,\underline{(\mathbf{n} \cdot \mathbf{v})}. \tag{9}$$

This is actually a plausible geometry term for a heightfield microsurface (which is what the Blinn-Phong normal distribution function corresponds to, since it is zero for all backfacing microfacets). $G_{\text{implicit}}()$ is equal to 1 when $\mathbf{l} = \mathbf{n}$ and $\mathbf{v} = \mathbf{n}$, which is correct for a heightfield (no microfacets are occluded from the direction of the macrosurface normal). It goes to 0 for either glancing view angles or glancing light angles, which again is correct (the probability of a microfacet being occluded by other microfacets increases with viewing angle, going to 100% in the limit). Given that this geometry factor actually costs less than zero shader cycles to compute (it cancels out the foreshortening factors so we don't need to divide by them), it has very good "bang per buck".

When comparing $G_{\text{implicit}}()$ to other geometry terms from the literature, we find that it goes to 0 too quickly–it is slightly too dark at moderately glancing angles. In other words, adding a more accurate geometry factor will have the result of somewhat brightening the specular term.

If we plug all these terms (Schlick Fresnel approximation, correctly normalized Blinn-Phong normal distribution function, and implicit geometry term) into the microfacet BRDF in Equation 5, we get

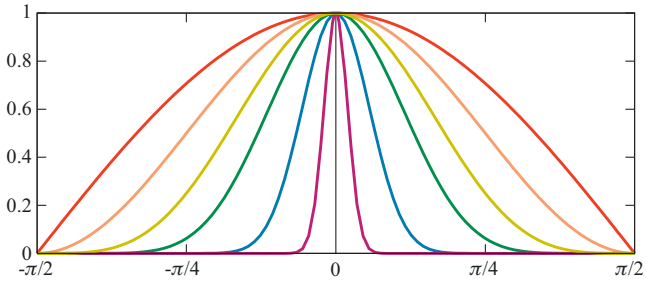

Figure 6: This graph shows the relationship between highlight brightness (y-axis) and angle (x-axis) for a Blinn-Phong term without a normalization factor; the different colors indicate various values of α_p. Note that the center of the highlight is always the same brightness regardless of the value of α_p, which is unrealistic. Furthermore, the overall reflected energy (which can be thought of as roughly corresponding to the volume under the surface created by rotating the curve around the y-axis) decreases as the specular power increases. This is undesirable; the overall reflected energy should only be affected by the parameter $\mathbf{c}_{\text{spec}}$, not α_p. (*image from "Real-Time Rendering, 3rd edition" used with permission from A K Peters*).

the following shading model:

$$\frac{\alpha_p + 2}{8}\underline{(\mathbf{n} \cdot \mathbf{h})}^{\alpha_p} F_{\text{Schlick}}(\mathbf{c}_{\text{spec}}, \mathbf{l}_\mathbf{c}, \mathbf{h}) \otimes \mathbf{c}_{\text{light}}\underline{(\mathbf{n} \cdot \mathbf{l}_\mathbf{c})}. \tag{10}$$

Besides the Fresnel term, the only difference between this model and the one in Equation 4 is the $(\alpha_p + 2)/8$ normalization factor, which results from multiplying the $(\alpha_p + 2)/2\pi$ normal distribution function normalization factor with the $\pi/4$ constant from Equation 5.

This normalization factor is hugely important for realism and ease of artist control. Unfortunately, it is not commonly used in game development, so we will spend some time discussing its advantages[2].

Values for the specular power parameter α_p commonly range from 0 to tens of thousands. This means that without the normalization factor, the specular term will be anywhere from four times too bright to thousands of times too dark. This error is large enough to make considerations of correct Fresnel values irrelevant. Omitting the normalization factor makes it extremely difficult for artists to create realistic-looking materials, especially when α_p varies per pixel (as it should). This is one of the primary reasons why the materials in many games look either like plastic or like chrome.

The graphs in Figures 6 and 7 show how highlight brightness varies with angle and specular power, with and without the normalization factor. Figure 8 shows the effect of the normalization factor on simple rendered images; unfortunately, we didn't have time to prepare comparison images with more complex materials, where the difference is even more noticeable.

The normalization factor also has significant advantages for art asset creation. It clearly separates the surface *material* (controlled by $\mathbf{c}_{\text{spec}}$) from its *roughness* (controlled by α_p). With this factor, varying the value of α_p by reading it from a texture (typically called a *roughness map* or *gloss map*) becomes a very effective way to control surface appearance. The values in this texture will simultaneously control highlight width and intensity, as opposed to just controlling the width as in a non-normalized shader.

Another advantage of the normalization factor is that it enables using real-world $F(0°)$ Fresnel values for $\mathbf{c}_{\text{spec}}$ (see the appropriate table in the background talk course notes), resulting in a realistic appearance similar to the desired material. Recall from the background talk that the vast majority of

[2]The fact that the normalization factor causes the reflected intensity $L_o(\mathbf{v})$ to be higher than the light intensity $\mathbf{c}_{\text{light}}$ may seem like a violation of energy conservation, but it is not. The apparent oddity results from the definition of $\mathbf{c}_{\text{light}}$. It is true that the reflection of a light source can never be more intense than the radiance observed when looking directly at the light. However, the definition of $\mathbf{c}_{\text{light}}$ is not related to the brightness of a light source when observed directly, but to the brightness of a white diffuse surface illuminated by the light source.

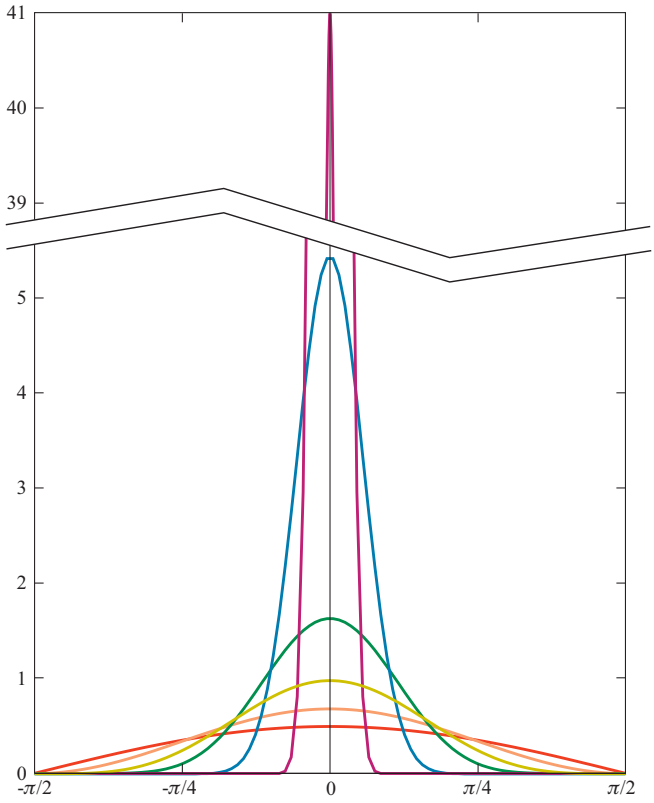

Figure 7: This graph shows the relationship between highlight brightness (y-axis) and angle (x-axis) for a Blinn-Phong term with a normalization factor; the different colors indicate various values of α_p. Note that the center of the highlight becomes brighter as α_p increases, which corresponds to the behavior of real-world surfaces. The overall reflected energy stays constant as α_p is changed. (*image from "Real-Time Rendering, 3rd edition" used with permission from A K Peters*).

real-world materials (anything that isn't a gem, crystal or metal) has a narrow range of $F(0°)$ values, between 0.02 and 0.06. For surfaces comprised of such materials, the variation in α_p will have a much greater effect on highlight intensity then the exact value of $\mathbf{c}_{\text{spec}}$. These materials can get very good results with a constant value of $\mathbf{c}_{\text{spec}}$ somewhere in the appropriate range, perhaps 0.04. This removes one texture from the shader, leaving only the normal map, diffuse color map, and gloss map. Such a reduction in textures is good for several reasons; it saves storage and texture read instructions, and perhaps more importantly it saves the artist from having to author another texture. Given the large impact of art asset creation cost on game budgets, this benefit is not to be underestimated.

A texture for $\mathbf{c}_{\text{spec}}$ thus becomes an "advanced" shader feature, and the specular power or gloss map is a "basic feature" which all shaders should have (this is, sadly, the reverse of current practice). For these "advanced" materials the artist needs to take care in painting values for $\mathbf{c}_{\text{spec}}$, using tables of real-world values as reference. It should also be noted that there is no such thing as "a surface without specular". Shaders without specular terms are commonly used in games for "matte-appearing" materials. However, in reality such materials have $\mathbf{c}_{\text{spec}}$ values around 0.03-0.06, and very low values of α_p (around 0.1-2.0). At glancing angles, even the most "matte' surfaces have noticeable specular appearance; the lack of this effect is another reason why so many game environments appear unrealistic.

As mentioned above, all objects should use roughness maps to vary α_p per-pixel. Artists should paint fine detail into these maps; real-world surfaces are covered with scratches, uneven wear patterns, pores, grooves, and other features which cause the microscopic roughness (modeled by α_p) to vary. The gloss map is closely tied to the normal map; when generating MIP-maps for both maps the variation in

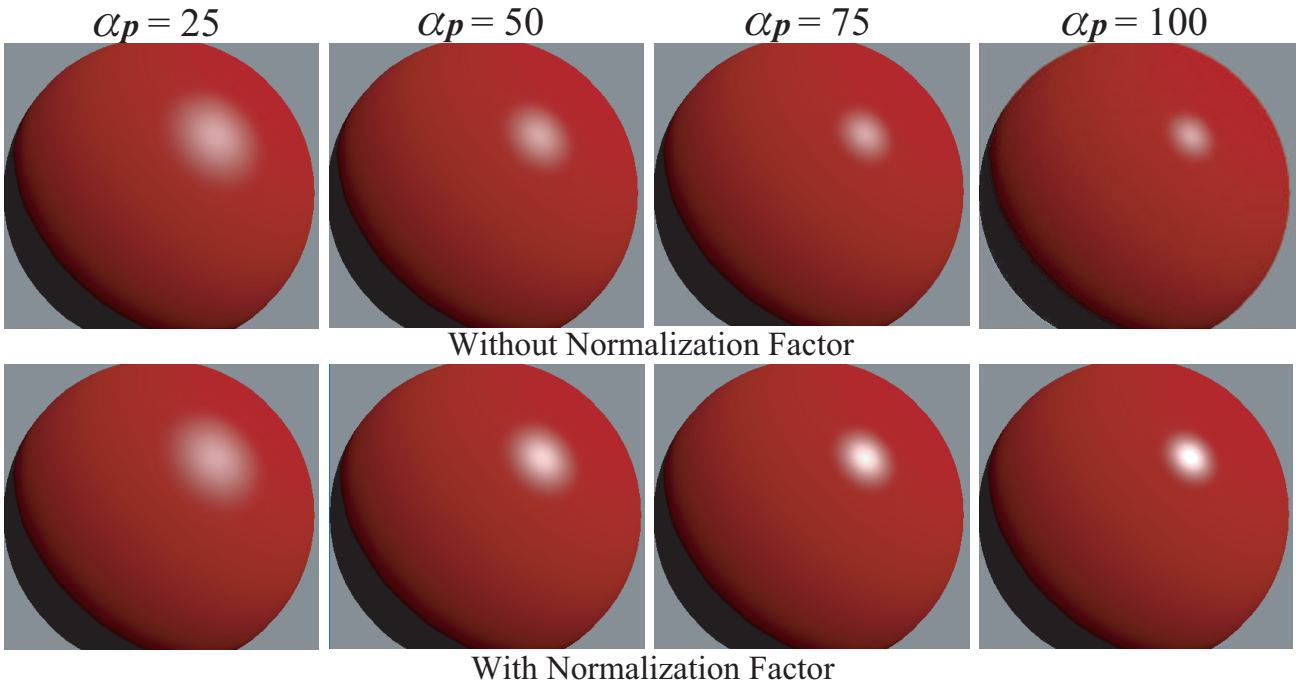

Figure 8: Rendered images of a red plastic sphere. The bottom row of images was rendered with a normalization factor applied to the specular term, using $\mathbf{c}_{\text{spec}} = 0.05$ (an appropriate value for plastic). The top row of images was rendered without a normalization factor, using a value of $\mathbf{c}_{\text{spec}}$ chosen so that the two leftmost images match. The intent is to render spheres made of the same material (red plastic) but with differing surface smoothness. It can be seen that in the bottom row, the highlight grows much brighter as it gets narrower, which is the correct behavior—the outgoing light is concentrated in a narrower cone. In the top row, the highlight remains equally bright as it narrows, so there is a loss of energy and surface reflectance appears to decrease. (*image from "Real-Time Rendering, 3rd edition" used with permission from A K Peters*).

normals should be used to modify the values of α_p [15, 23, 31, 32, 36]. If done correctly, this technique can greatly improve visuals at little or no runtime cost.

For best results, we have found that storing a nonlinear function of α_p in the gloss map helps to utilize limited precision and makes them more intuitive to paint. A good example function is $\alpha_p = (\alpha_{\text{max}})^s$ where α_{max} is a constant set to the highest specular power that will be used in the game, and s is a $0 - 1$ value read from the gloss map.

Environmental and Ambient Light

Environment maps (typically cube maps in game development) are important when using physical shading models. Since they have no diffuse color, all exposed metal surfaces should use environment maps, but it is worth considering using them everywhere, even on "matte" surfaces. The exact content of the environment map typically does not matter. With a few exceptions (such as a racing game where there is a smooth curved object in the center of the player's attention), incorrectly-shaped reflections are rarely noticed by players. However, it is important for the average color and intensity of the environment map to match the diffuse ambient or indirect lighting, otherwise the material's appearance will change. If both are derived from local samples in the game level (typically precomputed), then they will match by default.

However, it is much easier to vary diffuse ambient lighting continuously over the game environment than to do the same for environment maps. For this reason a way to "track" the environment map

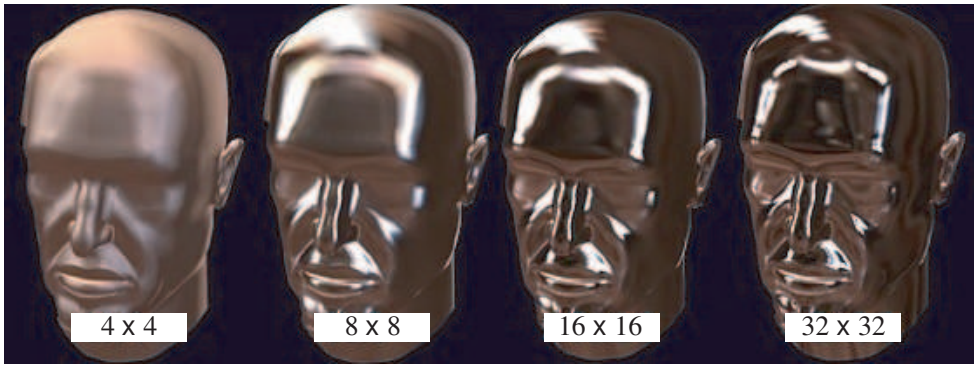

Figure 9: An example of a cube map mip chain filtered with Gaussian lobes of increasing width. Note that this creates a visual impression of changing surface roughness. (*CubeMapGen image from "Real-Time Rendering, 3rd edition" used with permission from A K Peters and AMD.*

to the diffuse ambient is useful. This can be done in a straightforward manner by "normalizing" the environment map (dividing it by its average value) in a pre-process, and then multiplying it by the diffuse ambient in the shader. The diffuse ambient value used should be averaged over all normal directions, in the case of spherical harmonics this would be the 0-order SH coefficient. This can produce quite good results even if the original environment map does not contain an image of the level or even of the same game (e.g., real-world light probes can be used).

Shading with environment maps is reasonably straightforward. The same $\mathbf{c}_{\text{spec}}$ value used for specular highlights should be applied, albeit with a slightly different Fresnel factor. As mentioned earlier, $F_{\text{Schlick}}(\mathbf{c}_{\text{spec}}, \mathbf{v}, \mathbf{n})$ should be used for environment maps and $F_{\text{Schlick}}(\mathbf{c}_{\text{spec}}, \mathbf{l}, \mathbf{h})$ (or the equivalent $F_{\text{Schlick}}(\mathbf{c}_{\text{spec}}, \mathbf{v}, \mathbf{h})$) for specular highlights. If it is desired to use the environment map directly for diffuse shading (instead of "tracking" it to some other diffuse ambient representation as described above), then the environment map should be prefiltered using a cosine term and stored in either a separate low-resolution environment map, the bottom MIP of the specular environment map, spherical harmonics coefficients, or some similar representation.

It is crucial to blur the environment map based on the value of α_p. For low values of α_p the environment map should be very blurry, and for very high values it should be sharp. Low specular powers should blur the environment map and specular highlight by the same amount. Fortunately, most of the hard work of blurring the environment map can be done in a preprocess. The blurring (filtering) should use full HDR values (they can be clipped to a lower range after filtering). It is recommended to use AMD's CubeMapGen library [6]; it has several important features for filtering cube maps that other texture processing libraries lack.

Once the environment map has been properly prefiltered, the shader just needs to select the appropriate MIP level based on the value of α_p. This is particularly effective when combined with per-pixel variation of α_p via a gloss map. Figure 9 shows a simple example of the visual results that can be achieved by prefiltering the cube map and selecting the MIP level in the shader.

If the $\alpha_p = (\alpha_{\max})^s$ gloss-map-to-specular-power mapping is used (as discussed above), then the desired MIP level is a simple linear function of s. The exact function can be calibrated by comparing a prefiltered black environment map with a single HDR texel to a directional light source.

When selecting MIP level in the shader, it is important to compare the desired MIP (calculated from α_p or s) to the MIP level which would be computed automatically by the hardware for a regular cube map lookup. The lower-resolution of the two MIP levels should be used. A straightforward method is to store the MIP level either in a separate cubemap or in the alpha channel of the environment map, and perform two cubemap lookups; one to get the automatic MIP level, and the final lookup. If the alpha channel of the environment map is used, the RGB from the first lookup (which corresponds to

an un-blurred reflection) can be used for effects like the "clear coat" double reflection found in metallic car paint.

In principle, environment maps should contain HDR data. It is most important to perform filtering on the full range. Some extended range is needed in the shader as well, but this can typically be handled by scaling to fit into low-precision formats. Since specular color will never be darker than 0.02, the environment map will saturate to white after it reaches a value 50 times brighter than display white (you may need a bit more if you want bloom from environment maps). As discussed in the HDR section, in sRGB space the range required tends to be quite low (100 in linear space corresponds to around 8 in sRGB space).

Other representations of indirection and environmental lighting, such as ambient terms or spherical harmonics can be applied to specular BRDFs. Yoshiharu Gotanda's talk in this course, *Practical Implementation of Physically-Based Shading Models at tri-Ace* gives a specular implementation for constant and SH ambient, a recent presentation by Bungie [3] discusses applying the Cook-Torrance [4, 5] specular term to SH lighting, and a ShaderX^7 article by Schüler [33] describes an implementation of a physically-based specular term with hemispherical lighting.

Fine-Tuning and Future Directions

This section discusses various lessons learned when helping game teams with the transition to physically-based shading models.

Overbright Highlights

Perhaps the most frequent complaint which arises after changing the shading models and reflectance values to physically correct ones, is that the specular is "too bright". There are several reasons for this; we will discuss the two most common ones.

The first reason is related to the behavior of the Fresnel term. In games, fine cracks and divots are typically modeled as normal maps rather than geometry. Since computing bump self-shadowing in the shader is too expensive for most games, it is common for artists to manually darken the diffuse and specular colors in the crevices instead. The aim is to avoid bright shading and highlights from deep crevices that should by all rights remain dark. However, a Fresnel term such as F_{Schlick} will brighten even the darkest specular colors at glancing angles, causing bright highlights to appear in deep cracks.

We are aware of two solutions to this problem. The first solution, proposed by Schüler [33], is to modify the Schlick approximation so that any values under a certain threshold are unaffected. Since we know that no real-world material has a value of $F(0°)$ lower than 0.02, any values of $\mathbf{c}_{\text{spec}}$ lower than this can be assumed to be the result of "prebaked" bump occlusion and left as is, without applying the Fresnel effect. This technique is effective, but it cannot handle more subtle or partial occlusions—occlusion is "all or nothing". For games which have separate ambient occlusion (AO) textures, another possible solution is to apply the AO texture to the specular term. While applying AO to direct lighting is technically incorrect, as long as care is taken to only apply small-scale occlusion (like AO or cavity maps) and not large-scale occlusion (like screen-space ambient occlusion—SSAO) to the specular term then the results are not too bad.

On the other hand, environment maps are more similar to ambient lighting and should have all AO terms applied to them, regardless of scale. Perceptual studies [9] have found that applying AO to environment maps is a reasonable compromise when more accurate (and expensive) forms of reflection occlusion are impractical (which is almost always the case for games).

Another common reason for over-bright specular highlights is related to the diffuse color ($\mathbf{c}_{\text{diff}}$) values. If material artists are not careful, it is easy for them to make the diffuse colors much too dark,

making the specular term appear too bright. If the game engine supports setting manual exposure values, dark diffuse values will typically cause the artists to over-expose, again making the specular values appear too bright. To avoid this, it is important to set exposure values using well-known principles such as the Ansel Adams zone system [1]. Basically an unshadowed diffuse white surface should expose to a value a little under display white to leave headroom for specular highlights. Any photo references used for diffuse textures should be carefully calibrated ("dividing out" the lighting). Textures painted from scratch should be carefully visualized as they will appear in game (the OpenColorIO [24] open source project includes relevant workflow examples).

Unsolved Problems and Future Work

Specular highlights and regular (non-prefiltered) environment maps have a well-defined Fresnel terms. In each case there is only one normal vector of interest; $\mathbf{n}$ for environment maps and $\mathbf{h}$ for microfacet specular highlights. However, in the case of prefiltered environment maps representing reflections from rough surfaces, there are many different microfacet orientations that contribute to the final color. A cheap Fresnel approximation that represents this case with reasonable accuracy would be a useful development.

Another unsolved problem occurs in the context of very smooth surfaces with high specular powers. Such materials are important to model e.g., wet surfaces. However, the punctual light approximation breaks down in this case, yielding extremely intense highlights of subpixel size that are unrealistic and alias badly. What we would like to see is a sharp reflection of the shape of the light source, which requires some kind of area light approximation which is fast enough to use in games.

A third problem is related to the visual differences between original Phong and Blinn-Phong that were discussed in a previous section. With a single environment map lookup, the visual results are similar to original Phong. Is there an inexpensive way to get stretched "Blinn-Phong-style" reflections from environment maps?

And finally there are a variety of geometry terms in the literature. Do any of them provide a visual improvement over the "cheaper-than-free" implicit geometry function G_{implicit} that is worth the extra cost? One candidate is the geometry factor proposed by Kelemen et. al. [21]. This is an approximation to the Cook-Torrance geometry factor [4, 5] but it is far cheaper to compute:

$$\frac{G_{\text{CT}}(\mathbf{l_c}, \mathbf{v}, \mathbf{h})}{\underline{(\mathbf{n} \cdot \mathbf{l_c})}\,\underline{(\mathbf{n} \cdot \mathbf{v})}} \approx \frac{1}{(\mathbf{l_c} \cdot \mathbf{h})^2}. \tag{11}$$

Just divide by the square of a dot product (which needs to be computed in any case for Fresnel) to get a reasonably close approximation to the full Cook-Torrance geometry factor divided by the foreshortening terms.

Another geometry factor that could be of interest is the one by Smith [34]. It is considered to be more accurate than the Cook-Torrance one, and takes account of surface roughness. Walter [37] gives an approximation to this factor. In Adam Martinez's talk in this course, *Faster Photorealism in Wonderland: Physically-Based Shading and Lighting at Sony Pictures Imageworks*, the use of this approximation for film production shading is discussed. It should be noted that even this approximation is significantly more costly than the Kelemen one; perhaps a cheaper one could be found for game use?

Acknowledgments

The author would like to thank A K Peters for permission to use images from the book *Real-Time Rendering, 3rd edition*, Elan Ruskin for permission to use his photographs, AMD for permission to use the CubeMapGen image, and Paul Edelstein, Yoshiharu Gotanda and Dimitar Lazarov for many thought-provoking discussions on physically-based shading models.

Further Reading

Chapter 7 of the 3rd edition of "Real-Time Rendering" [22] surveys various shading models appropriate for real-time use. More detail can be found in the book *Digital Modeling of Material Appearance* [7] by Dorsey, Rushmeier, and Sillion.

Bibliography

[1] Adams, Ansel, *The Negative: Exposure and Development*, Morgan and Lester, 1948 (we recommend reading the latest 1995 edition). Cited on p. 13

[2] Blinn, James F., "Models of Light Reflection for Computer Synthesized Pictures," *ACM Computer Graphics (SIGGRAPH '77 Proceedings)*, pp. 192–198, July 1977. http://research.microsoft.com/apps/pubs/default.aspx?id=73852 Cited on p. 5

[3] Chen, Hao, "Lighting and Material of Halo 3," *Game Developers Conference*, March 2008. http://www.bungie.net/images/Inside/publications/presentations/lighting_material.zip Cited on p. 12

[4] Cook, Robert L., and Kenneth E. Torrance, "A Reflectance Model for Computer Graphics," *Computer Graphics (SIGGRAPH '81 Proceedings)*, pp. 307–316, July 1981. Cited on p. 12, 13

[5] Cook, Robert L., and Kenneth E. Torrance, "A Reflectance Model for Computer Graphics," *ACM Transactions on Graphics*, vol. 1, no. 1, pp. 7–24, January 1982. http://graphics.pixar.com/library/ReflectanceModel/ Cited on p. 12, 13

[6] "CubeMapGen," *AMD GPU Tools website*, http://developer.amd.com/gpu/cubemapgen/Pages/default.aspx Cited on p. 11

[7] Dorsey, Julie, Holly Rushmeier, and François Sillion, *Digital Modeling of Material Appearance*, Morgan Kaufmann, 2007. Cited on p. 14

[8] Ericson, Christer, "Converting RGB to LogLuv in a fragment shader," "Real-Time Collision Detection" blog, July 9, 2007. http://realtimecollisiondetection.net/blog/?p=15 Cited on p. 3

[9] Green, Paul, Jan Kautz, and Frédo Durand, "Efficient Reflectance and Visibility Approximations for Environment Map Rendering," *Computer Graphics Forum*, vol. 26, no. 3, pp. 495–502, 2007. http://people.csail.mit.edu/green/ Cited on p. 12

[10] Gritz, Larry, and Eugene d'Eon, "The Importance of Being Linear," in Hubert Nguyen, ed., *GPU Gems 3*, Addison-Wesley, pp. 529–542, 2007. http://http.developer.nvidia.com/GPUGems3/gpugems3_ch24.html Cited on p. 3

[11] Hable, John, "Uncharted 2: HDR Lighting," *Game Developers Conference*, March 2010. http://filmicgames.com/archives/6 Cited on p. 3

[12] Hable, John, "Filmic Tonemapping Operators," "Filmic Games" blog, May 5, 2010. http://filmicgames.com/archives/75 Cited on p. 3

[13] Hable, John, "Why Reinhard Desaturates Your Blacks," "Filmic Games" blog, May 17, 2010. http://filmicgames.com/archives/183 Cited on p. 3

[14] Hable, John, "Why a Filmic Curve Saturates Your Blacks," "Filmic Games" blog, May 24, 2010. http://filmicgames.com/archives/190 Cited on p. 3

[15] Han, Charles, Bo Sun, Ravi Ramamoorthi, and Eitan Grinspun, "Frequency Domain Normal Map Filtering," *ACM Transactions on Graphics (SIGGRAPH 2007)*, vol. 26, no. 3, 28:1–28:11, July, 2007. http://www.cs.columbia.edu/cg/normalmap/ Cited on p. 10

[16] Hoffman, Naty, "Adventures with Gamma-Correct Rendering," "RenderWonk" blog, August 3, 2007. http://renderwonk.com/blog/index.php/archive/adventures-with-gamma-correct-rendering/ Cited on p. 3

[17] Hoffman, Naty, Haarm-Pieter Duiker, Joseph Goldstone, Yoshiharu Gotanda, Dominic Glynn, Lou Levinson, Josh Pines, and Jeremy Selan, "Color Enhancement and Rendering in Film and Game Production," *SIGGRAPH 2010 Course Notes*, 2010. http://renderwonk.com/publications/s2010-color-course/ Cited on p. 3

[18] International Electrotechnical Commission, "IEC/4WD 61966-2-1: Colour Measurement and Management in Multimedia Systems and Equipment - Part 2-1: Default RGB Colour Space - sRGB," CIE Division 8 website, May 28, 1998. http://www.colour.org/tc8-05/Docs/colorspace/61966-2-1.pdf Cited on p. 2

[19] Kaplanyan, Anton, "CryENGINE 3: Reaching the Speed of Light," *SIGGRAPH 2010 Advanced Real-Time Rendering in 3D Graphics and Games course notes*, 2010. http://advances.realtimerendering.com/s2010/index.html Cited on p. 3

[20] Karis, Brian, "RGBM color Encoding," "Graphic Rants" blog, April 28, 2009. http://graphicrants.blogspot.com/2009/04/rgbm-color-encoding.html Cited on p. 3

[21] Kelemen, Csaba, and Lázló 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 Cited on p. 13

[22] Akenine-Möller, Tomas, Eric Haines, and Naty Hoffman, *Real-Time Rendering*, third edition, A K Peters Ltd., 2008. http://realtimerendering.com/ Cited on p. 14

[23] Olano, Marc, and Dan Baker. "LEAN Mapping," *ACM Symposium on Interactive 3D Graphics and Games (I3D 2010)*, pp. 181–188, February 2010. http://www.cs.umbc.edu/~olano/papers/ Cited on p. 10

[24] "OpenColorIO," *Sony Pictures Imageworks open source website*, http://opensource.imageworks.com/?p=opencolorio Cited on p. 13

[25] Phong, Bui Tuong, "Illumination for Computer Generated Pictures," *Communications of the ACM*, vol. 18, no. 6, pp. 311–317, June 1975. http://jesper.kalliope.org/blog/library/p311-phong.pdf Cited on p. 4

[26] Poynton, Charles, *Digital Video and HDTV: Algorithms and Interfaces*, Morgan Kaufmann, 2003. Cited on p. 2

[27] Reinhard, Erik, Mike Stark, Peter Shirley, and James Ferwerda, "Photographic Tone Reproduction for Digital Images," *ACM Transactions on Graphics (SIGGRAPH 2002)*, vol. 21, no. 3, pp. 267–276, July 2002. http://www.cs.utah.edu/~reinhard/cdrom/ Cited on p. 3

[28] Robertson, Barbara, "Rampant Risk-Taking," *Computer Graphics World*, vol. 31, no. 7, pp. 10–17, July 2008. http://www.cgw.com/Publications/CGW/2008/Volume-31-Issue-7-July-2008-/Rampart-Risk-Taking.aspx Cited on p. 1

[29] Robertson, Barbara, "Shades of the Future," *Computer Graphics World*, vol. 32, no. 5, pp. 18–24, May 2009. http://www.cgw.com/Publications/CGW/2009/Volume-32-Issue-5-May-2009-/Shades-of-the-Future.aspx Cited on p. 1

[30] Robertson, Barbara, "Rare Mettle," *Computer Graphics World*, vol. 33, no. 5, pp. 16–22, May 2010. http://www.cgw.com/Publications/CGW/2010/Volume-33-Issue-5-May-2010-/Rare-Mettle.aspx Cited on p. 1

[31] Schilling, Andreas, "Towards Real-Time Photorealistic Rendering: Challenges and Solutions," *ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* Los Angeles, CA, pp. 7–16, August 1997. http://graphics.stanford.edu/courses/cs448-05-winter/Schilling-1997-Towards.pdf Cited on p. 10

[32] Schilling, Andreas, "Antialiasing of Environment Maps," *Computer Graphics Forum*, vol. 20, no. 1, pp. 5–11, March 2001. http://www.gris.uni-tuebingen.de/fileadmin/user_upload/Paper/Schilling-2001-Antialiasing.pdf Cited on p. 10

[33] Schüler, Christian, "An Efficient and Physically Plausible Real Time Shading Model," in Wolfgang Engel, ed., *ShaderX*^{7}, Charles River Media, pp. 175–187, 2009. Cited on p. 12

[34] Smith, Bruce G., "Geometrical Shadowing of a Random Rough Surface," *IEEE Transactions on Antennas and Propagation*, vol. 15, no. 5, pp. 668–671, September 1967. Cited on p. 13

[35] Tchou, Chris, "HDR The Bungie Way," *Gamefest*, August 2006. http://www.microsoft.com/downloads/details.aspx?FamilyId=995B221D-6BBD-4731-AC82-D9524237D486&displaylang=en Cited on p. 3

[36] Toksvig, Michael, "Mipmapping Normal Maps," *journal of graphics tools*, vol. 10, no. 3, pp. 65–71, 2005. http://developer.nvidia.com/object/mipmapping_normal_maps.html Cited on p. 10

[37] Walter, Bruce, Stephen R. Marschner, Hongsong Li, Kenneth E. Torrance, "Microfacet Models for Refraction through Rough Surfaces," *Eurographics Symposium on Rendering (2007)*, 195–206, June 2007. http://www.cs.cornell.edu/~srm/publications/EGSR07-btdf.html Cited on p. 6, 13