Moving Frostbite to Physically Based Rendering 3.0

Sébastien Lagarde

Electronic Arts Frostbite

Charles de Rousiers

Electronic Arts Frostbite

SIGGRAPH 2014

Contents

1 Introduction

Over the course of the past few months, we have been re-evaluating our entire approach to image quality in *Frostbite*[1]. Our overall goal has been to achieve a 'cinematic look', thus moving to physically based rendering (PBR) was the natural way to achieve this. The transition work we have done for *Frostbite* has been based upon the current state of the art in the game industry, such as *Killzone Shadow Fall* [Dro13], *Unreal Engine 4* [Kar13], *Remember Me* [LH13] and *Metal Gear Solid V: Ground Zeroes* [Koj+13]. From this foundation, we have tried to further refine existing techniques and chip away at open problems in the field.

Throughout the R&D process, we have used 'ground truth' reference—measured or rendered, as appropriate—to evaluate the accuracy of our solutions. However, being truly physically correct is a huge task and it is unlikely that game engines can achieve it today, given current real-time performance contraints. Therefore, where decent approximations are still possible, we favor *believability* over absolute correctness if it brings us closer to our image quality target.

PBR has become a common industry term, but its meaning differs a lot between game engines. For us, one of the core PBR principles is the decoupling of material and lighting information, which is key to ensuring visual consistency between all objects in a scene. With this approach, the same lighting is applied on all objects and their material layers, without any hacks such as negative lights, or artifacts such as the 'double counting' of light contributions. From a production perspective, this facilitates the reuse of assets and lighting rigs across different environments, in a transparent fashion. At the same time, it reduces the number of parameters exposed to artists and makes the authoring more intuitive. However, as we will see later in this document, this separation is only true from an authoring point of view, as lighting and materials are tightly coupled in the code, for performances reasons.

When embracing PBR, one quickly understands that the entire graphics pipeline (renderer and tools) needs to be updated. With that in mind, our aim with these course notes has been to cover all of the different upgrades required for a large-scale production engine, including many small details typically omitted in the literature. First off, Section 2 explains in more detail how ground-truth references are important in the context of PBR. Next, Section 3 presents materials, and reviews how light interacts with matter. Following this, Section 4 describes how light is defined and emitted. Section 5 focuses on the camera and the output image, covering how luminance is transformed to final pixel values. Finally, we conclude with Section 6, reviewing how our transition to PBR was scheduled and what we considered during this period.

Before continuing, we would like to mention this work is the result of a collaboration between many people across the industry. This document gathers a lot of information that has been shared by our fantastic graphics community and many people should be credited for this work (see the Acknowledgments section).

Remarks: throughout this document, we will use the notation described in Table 1.

---

[1]Frostbite is a game development platform that powers an increasing number of titles at EA. See `http://www.frostbite.com` for more details.

$\mathbf{v}$	view vector
$\mathbf{l}$	incident light vector
$\mathbf{n}$	normal
$\mathbf{h}$	half vector
L	lighting function
f	BRDF
f_d	diffuse component of a BRDF
f_r	specular component of a BRDF
α	material roughness
α_{lin}	perceptually linear material roughness
$\cdot$	dot product
$\langle \, \cdot \, \rangle$	clamped dot product
$\lvert \, \cdot \, \rvert$	absolute value of the dot product
ρ	diffuse reflectance
$\chi^+(a)$	Heaviside function: 1 if $a > 0$ and 0 if $a <= 0$

Table 1: Mathematical notation.

2 Reference

2.1 Validating models and hypothesis

The video game industry have try during decades to get more photorealistic image. But *Photorealism* says nothing about the data and the methods used to create such imagery. Results are judged qualitatively. Unlike *physically based rendering* which try to simulate real-world behavior and properties, where the result are judge quantitatively. This place an additional requirement on ground-truth data. It is important to carefully choose the right models and the correct hypotheses, i.e good references. Observing and comparing with the real world is the best approach to make the right choices and to judge how relevant a technique or a method is. At a coarse level observing the real world allows us to quickly grasp the shape of highlights, the behavior of a wet surface, the difference in light intensities, as well as many other visual features, see Figure 1. When taking real world reference for a material, it is important to take pictures at multiple scales to capture the differing lighting behavior exhibited at each scale.

Figure 1: Comparison of real world lighting (left) and in-engine lighting (right).

However it is often quite complex or too time consuming to measure real data accurately. Certain databases like MERL [MER] offer access to such data which allows us to quickly assess models. In our approach we tried to measure and verify actual data, such as light intensity and falloff, sky brightness and camera effects. But all these steps are time consuming and not always easy to set up.

2.2 Validating in-engine approximations

Modern PBR path tracers like Mitsuba [Jak10] (courtesy of Wenzel Jakob) implement state of the art rendering techniques and can create incredible realistic images. Using such software[2] is an easier alternative for assessing the accuracy of a model. In *Frostbite* we have written a simple exporter for Mitsuba which allows us to quickly assess the validity of our approximations. The exporter is capable of exporting geometry and constant material information (i.e. no textures) and all light sources. With this setup it is easy to check material models, light integration and light intensities. In addition this exporter allows us to verify the accuracy of more complex phenomena like global illumination, ambient occlusion and reflections. Figure 2 shows a widget automatically triggered after export, allowing one to quickly swipe and compare pixels values between the in-engine result and offline reference, with full control over exposure. This last point is important, due to the wide range of intensity output by the renderers. To conserve this range, both renderers export their final image into a linear HDR format, OpenEXR [Opea].

---

[2]We tested several rendering packages, but found that most were not accurate enough for our needs. Mitsuba and Maxwell are our preferred options.

Figure 2: Our tool for comparing in-engine results with an offline path-traced result, in order to assess the validity of our implementation.

2.3 Validating in-engine reference mode

An exporter, as described above, is useful but it requires anywhere between a few seconds and a few minutes to export and render a scene. In order to quickly iterate on various approximations and choose the right ones, we have added an in-engine reference mode for our lighting integration by brute-force sampling (image based lights and area lights) on the GPU, Figure 3. Render times are not fast but iteration times are an order of magnitude faster than with our simple exporter. Appendix A contains listings for evaluating several types of light in reference mode.

(a) In engine.   (b) In engine reference.   (c) Path tracer reference.
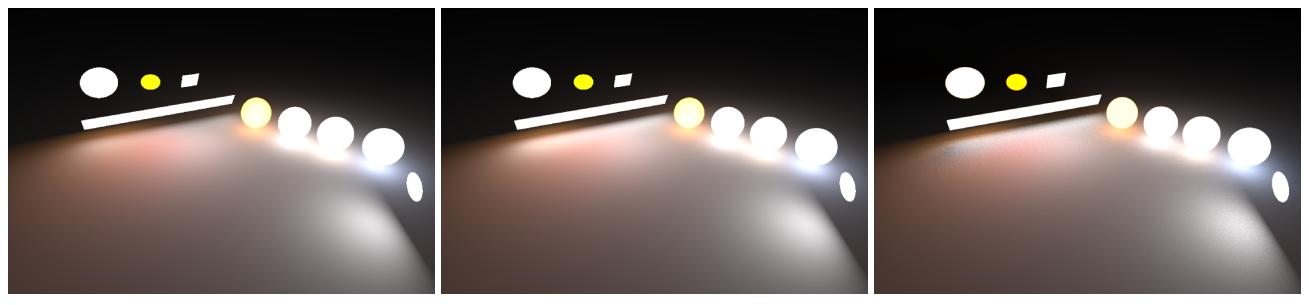
Figure 3: Left: Rendering of a scene composed of several types of area lights, in-engine. Middle: Same scene rendered in-engine by importance sampling the area lights on GPU. Right: Scene rendered inside a path tracer for validation.

Note: It is important to use correct references. This may seem obvious but if the reference is not good, the approximation will not be too. If approximating a hair shading model, use the one closest to the real world as reference. When approximating formulas, be sure to use the original equation and not an already approximated one, such as Oren-Nayar or Schlick's approximation to the Fresnel equation, as this can lead to errors. The only reference that can be fully trusted will always be the real world.

5

3 Material

3.1 Material models

3.1.1 Appearance

Surface appearance results from the interaction between incoming light and the material properties of a surface. The variety of appearances observable in the real world is quite wide, ranging from simple uniform materials to complex layered and heterogeneous materials, see Figure 4.
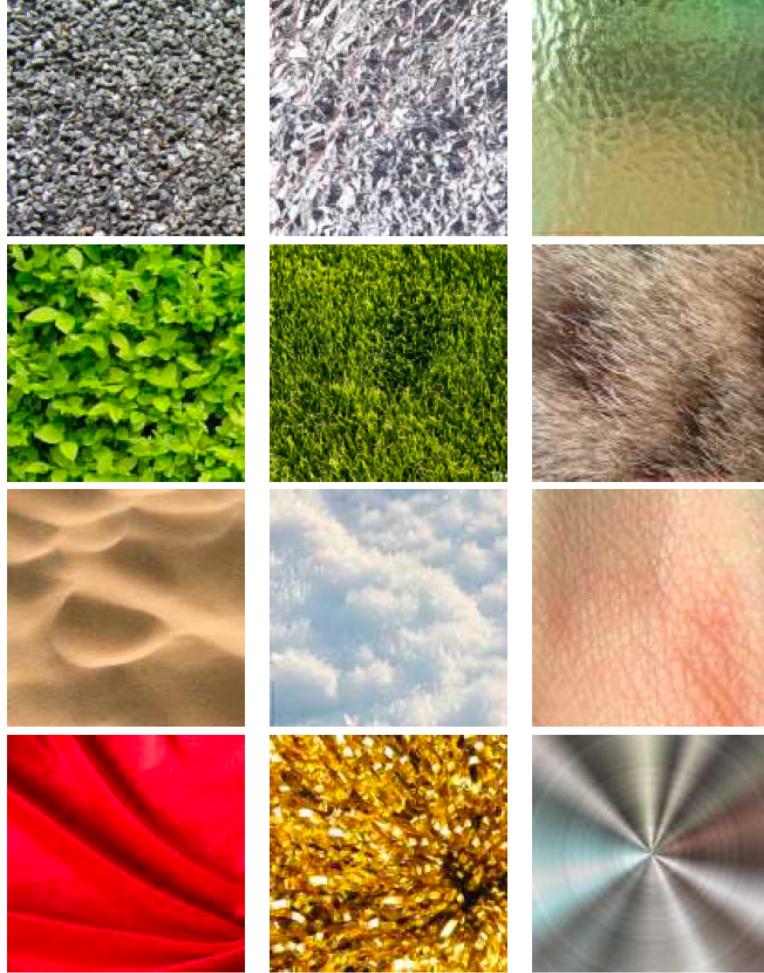

Figure 4: Various surface appearances showing the diversity of interaction between light and matter.

These different appearances can be categorized by certain intrinsic physical properties such as conductivity, mean-free-path, and absorption. Based on these material properties, the literature has exposed various material models capable of representing a certain range of appearances amongst the full spectrum. Material model literature is extensive, and a lot of different models exist with various trade-offs and accuracies. A material model, referred to as a BSDF (Bidirectional Scattering Distribution Function), can be decomposed into two parts: a Reflectance part (BRDF) and a Transmittance part (BTDF). In this document we will focus on the reflective part and in particular on a material model capable of representing "standard" appearances i.e. the vast majority of surfaces we encounter in our every day life. Hence, we will limit ourselves to reflective, isotropic, dielectric/conductor surfaces with short mean-free-paths.

(a) Light interactions.   (b) BSDF modelization.
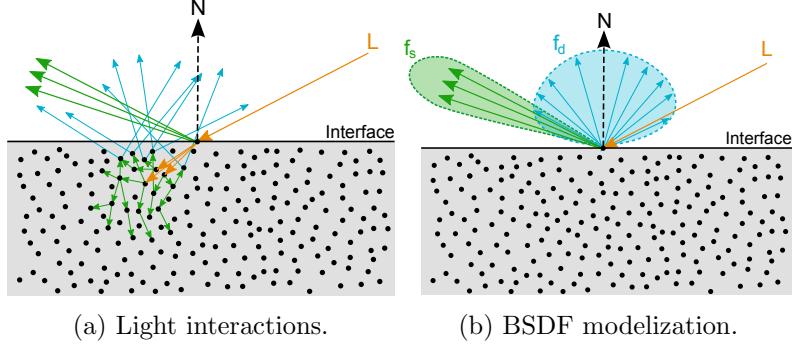
Figure 5: Light interaction with a slab of "standard" matter. Left: Light interactions. Right: a BSDF model of the interaction with a diffuse term f_d and a specular term f_r.

3.1.2 Material models

In the context of this standard material model, a surface response f is often decomposed into two different terms: a low angular frequency signal called "diffuse" (f_d) and a low to high angular frequency part called "specular" (f_r), see Figure 5. An interface separates two media: the air and the matter. Surfaces made of a flat interface can easily be represented by the Fresnel law [Wikd] for both dielectric and conductor surfaces. When the interface is irregular, see Figure 6, the literature shows that microfacet based models [CT82] are well adapted to characterize the light interaction for these types of surfaces. A microfacet model is described by Equation 1, for more details about the derivations see [Hei14]:

$$f_{d/r}(\mathbf{v}) = \frac{1}{|\mathbf{n} \cdot \mathbf{v}||\mathbf{n} \cdot \mathbf{l}|} \int_\Omega f_m(\mathbf{v}, \mathbf{l}, \mathbf{m}) \; G(\mathbf{v}, \mathbf{l}, \mathbf{m}) \; D(\mathbf{m}, \alpha) \; \langle \mathbf{v} \cdot \mathbf{m} \rangle \langle \mathbf{l} \cdot \mathbf{m} \rangle \; \mathrm{d}\mathbf{m} \tag{1}$$

The term D models the microfacet distribution (i.e. the NDF, Normal Distribution Function). The G term models the occlusion (shadow-masking) of the microfacets. This formulation is valid for both the diffuse term f_d and the specular term f_r. The difference between these two terms lies in the microfacet BRDF f_m. For the specular term, f_m is a perfect mirror and thus is modeled with the Fresnel F law, which leads to the well-known following formulation:

$$f_r(\mathbf{v}) = \frac{F(\mathbf{v}, \mathbf{h}, f_0, f_{90}) \; G(\mathbf{v}, \mathbf{l}, \mathbf{h}) \; D(\mathbf{h}, \alpha)}{4 \langle \mathbf{n} \cdot \mathbf{v} \rangle \langle \mathbf{n} \cdot \mathbf{l} \rangle} \tag{2}$$

The term D plays an important role in the appearance of surfaces, as shown by Figure 6. The literature, [Wal+; Bur12] has recently pointed out that "long-tailed" NDFs, like the GGX distribution, are good at capturing real world surfaces. The G term also plays an important role for high roughness values. Heitz [Hei14] has recently shown that the Smith visibility function is the correct and exact G term to use. He also points out that the literature often tends to use an approximated version of the Smith visibility function, while a more accurate form of the masking-shadowing function models the correlation between the masking and shadowing due to the height of the microsurface, see Equation 3. Figure 7 shows the difference between the simple Smith function and the height-correlated Smith function.

$$G(\mathbf{v}, \mathbf{l}, \mathbf{h}, \alpha) = \frac{\chi^+(\mathbf{v}.\mathbf{h})\chi^+(\mathbf{l}.\mathbf{h})}{1 + \Lambda(\mathbf{v}) + \Lambda(\mathbf{l})} \; \text{with} \; \Lambda(\mathbf{m}) = \frac{-1 + \sqrt{1 + \alpha^2 \tan^2(\theta_m)}}{2} = \frac{-1 + \sqrt{1 + \frac{\alpha^2(1 - \cos^2(\theta_m))}{\cos^2(\theta_m)}}}{2}$$

$$\tag{3}$$
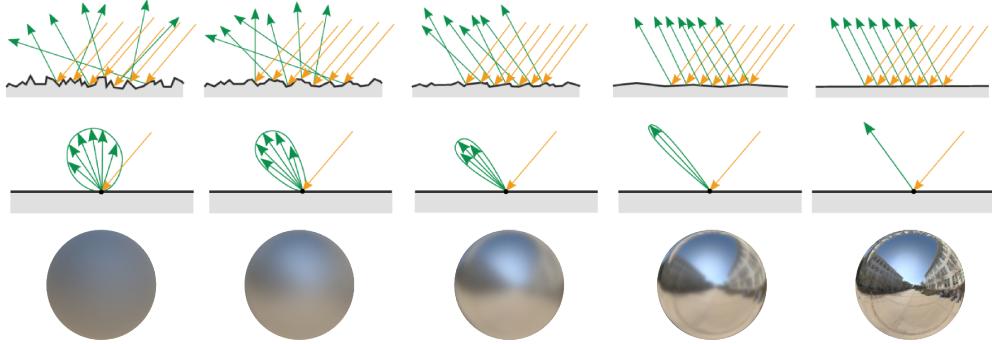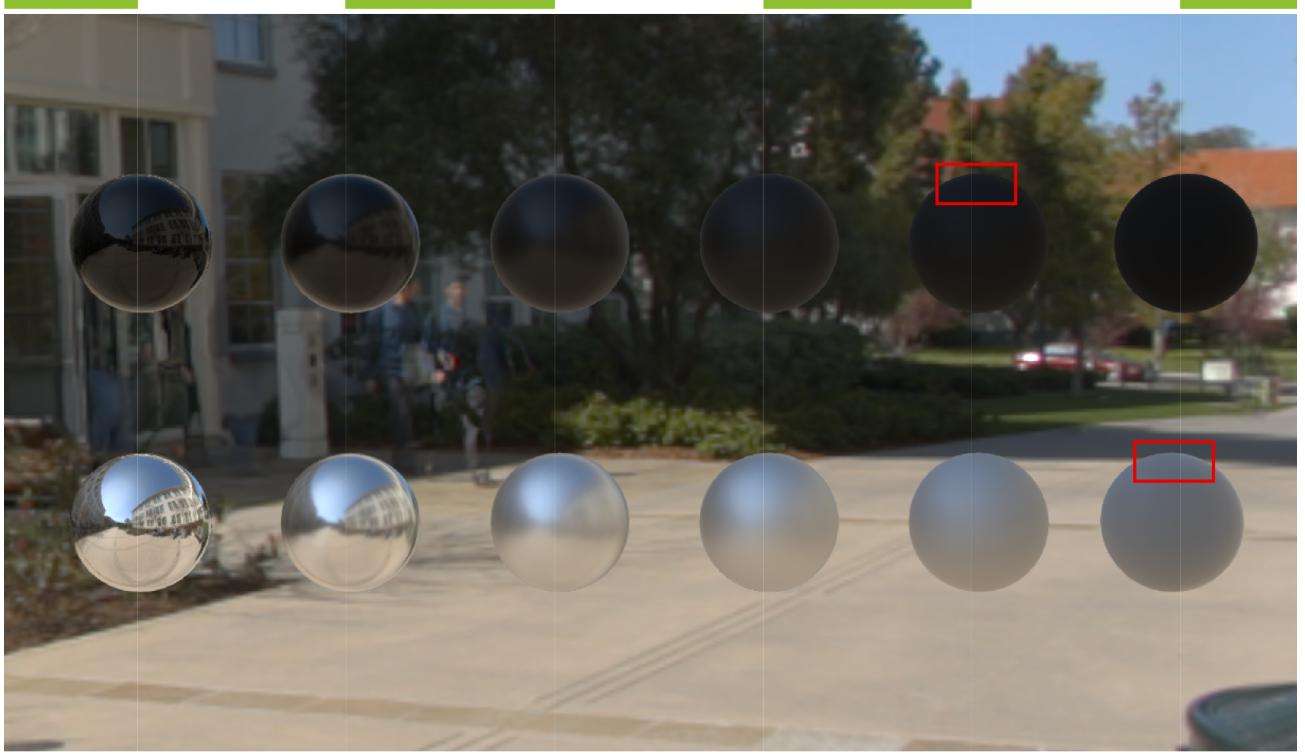
Figure 6: Shows surfaces of different roughness, modeled by the D term. Top: Light interactions with the microfacets. Middle: Resulting BRDF f_r lobe. Bottom: Resulting appearance on spheres.

Smith G term - Height correlated

Smith G term - Decorrelated

Figure 7: Comparison between uncorrelated and correlated Smith visibility function on a set of black dielectric (Top) and chrome metallic (Bottom) spheres with increasing roughness. Note how the height-correlated version brings slightly more energy back for high roughness values.

For the diffuse term, f_m follows a Lambertian model and Equation 1 can be simplified into:

$$f_d(\mathbf{v}) = \frac{\rho}{\pi} \frac{1}{|\mathbf{n} \cdot \mathbf{v}||\mathbf{n} \cdot \mathbf{l}|} \int_\Omega G(\mathbf{v}, \mathbf{l}, \mathbf{m}) \ D(\mathbf{m}, \alpha) \ \langle \mathbf{v} \cdot \mathbf{m} \rangle \langle \mathbf{l} \cdot \mathbf{m} \rangle \ \mathrm{d}\mathbf{m} \tag{4}$$

Until recently the diffuse term f_d was assumed to be a simple Lambertian model. However, except for layered materials, the diffuse part needs to be coherent with the specular term and must take into account the roughness of the surface [Bur12] (i.e specular and diffuse term should use same roughness

term[3]), see Figure 8. The Equation 4 does not have an analytic solution. Oren et al. [ON94] found an empirical approximation of this equation using a Gaussian NDF distribution and a V-cavity G term know as the Oren-Nayar model. To correctly support our model we should make an equivalent approximation for Equation 4 with a GGX NDF as describe in Gotanda [Got14]. Appendix B details some of our analysis about such a diffuse model, but further research is required.
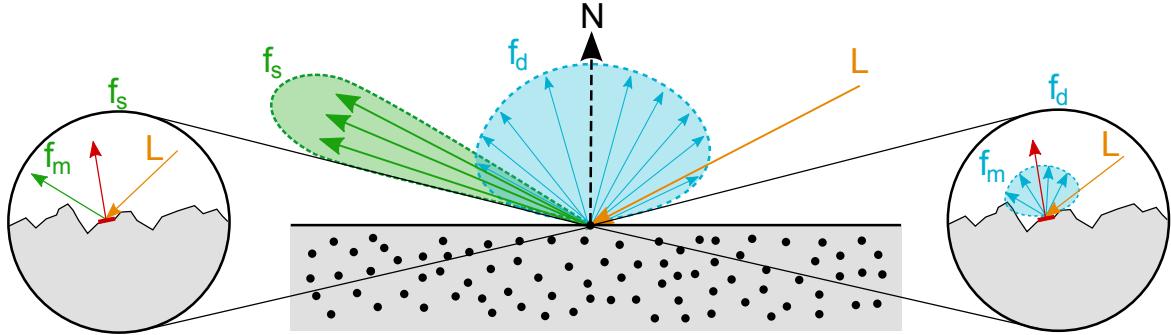

Figure 8: Shows a view of the interaction at the micro scale for both the specular f_r term, which has a mirror BRDF f_m (left), and the diffuse term f_d, which has a diffuse BRDF f_m (right).

Burley [Bur12] has presented another diffuse model built on real world surface observations, see Equation 5. While this model is empirical, it allows us to reproduce the main features of the MERL database's materials. For this reason and because of its simplicity, we have chosen to use this model in *Frostbite*. This diffuse term takes into account the roughness of the material and creates some retro-reflection at grazing angles.

$$f_d = \frac{\rho}{\pi}(1 + F_{D90}(1 - \langle \mathbf{n} \cdot \mathbf{l} \rangle))^5)(1 + F_{D90}(1 - \langle \mathbf{n} \cdot \mathbf{v} \rangle))^5) \quad \text{where} \quad F_{D90} = 0.5 + \cos(\theta_d)^2\alpha \qquad (5)$$

3.1.3 Energy conservation

Energy conservation is important to consider in order to not add more energy than received. In addition it allows us to correctly handle the behavior at grazing angles, for which the light tends to be scattered more by the specular term than the diffuse term. In *Frostbite* we have chosen to keep the computation simple and only ensure the preservation of energy, by ensuring that the hemispherical-directional reflectance, which gives the total reflectance in a given direction due to constant illumination over the hemisphere, is below one for our whole BRDF (diffuse + specular term):

$$\rho_{hd}(\mathbf{v}) = \int_\Omega f(\mathbf{v}, \mathbf{l}) \langle \mathbf{n} \cdot \mathbf{l} \rangle \, d\mathbf{l} = \int_\Omega (f_r(\mathbf{v}, \mathbf{l}) + f_d(\mathbf{v}, \mathbf{l})) \langle \mathbf{n} \cdot \mathbf{l} \rangle \, d\mathbf{l} \leq 1 \qquad (6)$$

Making the proper derivations would be not easy due to the non-direct relation between our specular and diffuse models (see Appendix C for the case where specular and diffuse terms are both based on microfacet models). One important caveat of the Disney diffuse model is its lack of energy conservation[4]. Figure 9a shows the hemispherical-directional reflectance of the Disney diffuse model. We can clearly see this BRDF is not energy conserving, since the resulting reflectance value is above 1.

---

[3]There is two ways to model the relation between the diffuse and the specular term. Either consider the specular and the diffuse terms as part of the same layer or consider the specular term as a separate layer on top of the diffuse term. In *Frostbite* we use the former model and thus use same roughness term for diffuse and specular terms.

[4]This is by design, as explained by Burley [Bur12]. This choice was motivated by allowing artist to get the same diffuse color across all roughness values.

(a) Original f_d.   (b) Renormalized f_d.   (c) Renormalized $f_d + f_r$.
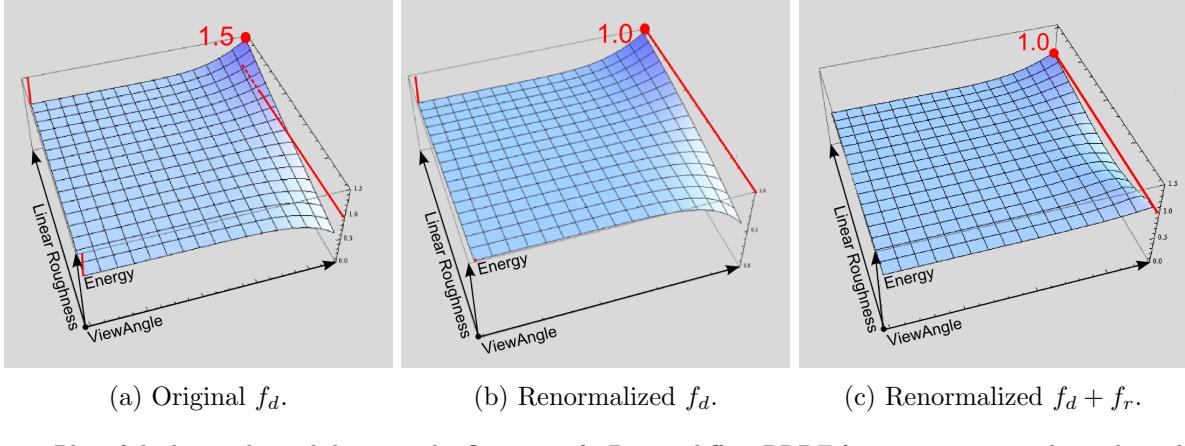
Figure 9: Plot of the hemispherical-directional reflectance of a Disney diffuse BRDF for various view angles and roughness. Left: original reflectance value, which can go above one. Middle: new renormalized reflectance values. Right: the combination of a specular and diffuse terms.

We have slightly modified it to compensate for the gain of energy while preserving its retro-reflective properties. Listing 1 shows the Disney evaluation function with the renormalization factors. Figure 9c shows the hemispherical-directional reflectance of the complete f, composed of a specular microfacet model for f_r and the Disney diffuse model for f_d. While not perfectly equal to one, it is close enough. Figure 10 compares the original Disney diffuse term with its renormalized version.
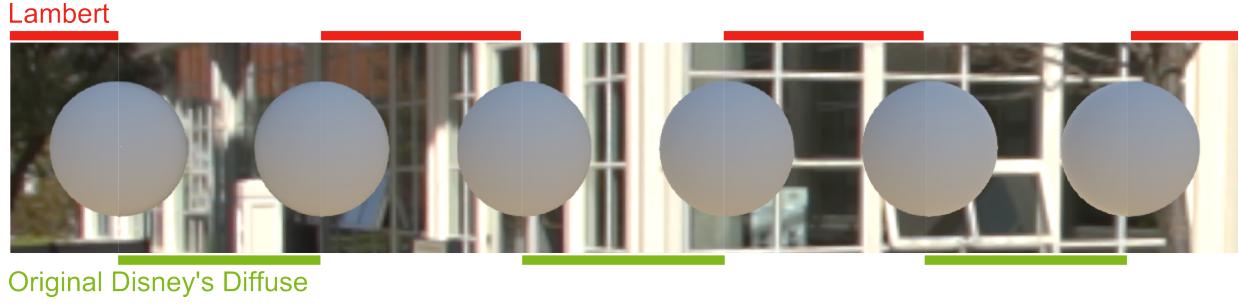
```
1  float Fr_DisneyDiffuse(float NdotV, float NdotL, float LdotH,
2                          float linearRoughness)
3  {
4      float energyBias    = lerp(0, 0.5, linearRoughness);
5      float energyFactor  = lerp(1.0, 1.0 / 1.51, linearRoughness);
6      float fd90          = energyBias + 2.0 * LdotH*LdotH * linearRoughness;
7      float3 f0           = float3(1.0f, 1.0f, 1.0f);
8      float lightScatter  = F_Schlick(f0, fd90, NdotL).r;
9      float viewScatter   = F_Schlick(f0, fd90, NdotV).r;
10
11     return lightScatter * viewScatter * energyFactor;
12 }
```

Listing 1: Disney's diffuse BRDF code with renormalization of its energy. linearRoughness is the perceptually linear roughness (See section 3.2.1)
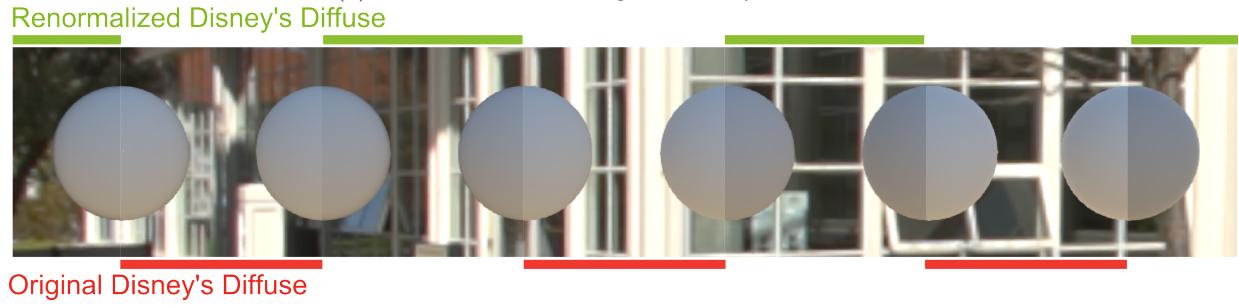
3.1.4 Shape characteristics

Specular microfacet-based BRDFs have certain properties which are often bypassed but have a strong impact on the final appearance. In particular two phenomena are important:

- **Half-angle parametrization**: this parametrization implies a non-linear transformation of the BRDF shape which goes from isotropic at normal incident angles to anisotropic towards grazing angles. See the Section 4.9 for more insight on this part.

- **Off-specular**: it is often assumed that a BRDF lobe is centred around the reflected view direction (also known as mirror direction). However due to $\langle \mathbf{n} \cdot \mathbf{l} \rangle$ and the shadow-masking term G, the BRDF lobe gets shifted towards the normal direction when roughness increases, see Figure 11. This shifting is called "*Off-specular peak*" and it plays an important role in the rough appearance of a surface.

10

(a) Lambert and the original Disney diffuse term.

(b) Original Disney diffuse term and its renormalized version.

Figure 10: Top: Comparison of the Disney diffuse term with a Lambertian diffuse term. Bottom: Comparison of the original Disney diffuse term and the renormalized version that we have introduced.
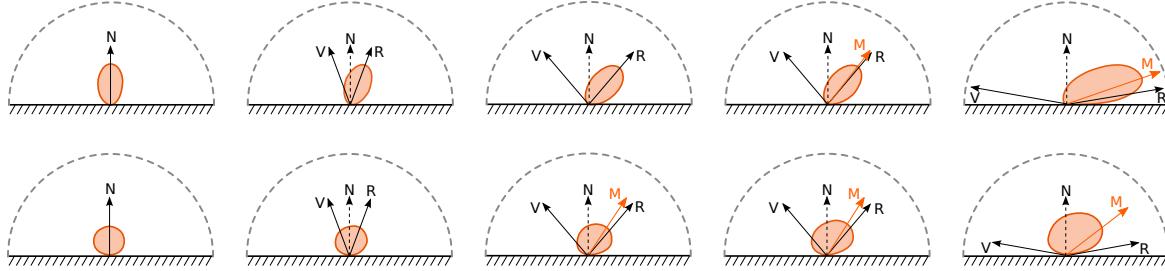
Figure 11: Example of BRDF lobe shapes for various view angles, where the dominant lobe direction is not aligned with the mirror direction R for grazing view angles, but instead with the direction M. Top row: $\alpha = 0.4$. Bottom row: $\alpha = 0.8$.

The off-specular peak can lead to large differences for high roughness values. In order to take into account this important feature, we have tried to model this "dominant direction" which is used during area light and image-based light evaluation, see area light Section 4.7 and image-based lighting Section 4.9.

3.1.5 *Frostbite* standard model

To summarize, *Frostbite*'s "standard" material model is close to the one used by other game engines [Kar13; NP13; Bur12]. It is composed of:

- **Specular term** f_r: a specular microfacet model (see Equation 2) with a Smith correlated visibility function and a GGX NDF.
- **Diffuse term** f_d: the Disney diffuse term with energy renormalization.

For both parts, we apply our dominant direction correction (off-specular peak handling) when integrating lighting. The parameters for manipulating these models will be described in the next section.

Frostbite also supports other types of material such as clearcoat and materials with subsurface scattering, but we will focus only on the standard material model in this document.

```
1  float3 F_Schlick(in float3 f0, in float f90, in float u)
2  {
3      return f0 + (f90 - f0) * pow(1.f - u, 5.f);
4  }
5
6  float V_SmithGGXCorrelated(float NdotL, float NdotV, float alphaG)
7  {
8      // Original formulation of G_SmithGGX Correlated
9      // lambda_v            = (-1 + sqrt(alphaG2 * (1 - NdotL2) / NdotL2 + 1)) * 0.5f;
10     // lambda_l            = (-1 + sqrt(alphaG2 * (1 - NdotV2) / NdotV2 + 1)) * 0.5f;
11     // G_SmithGGXCorrelated = 1 / (1 + lambda_v + lambda_l);
12     // V_SmithGGXCorrelated = G_SmithGGXCorrelated / (4.0f * NdotL * NdotV);
13
14     // This is the optimize version
15     float alphaG2 = alphaG * alphaG;
16     // Caution: the "NdotL *" and "NdotV *" are explicitely inversed, this is not a mistake.
17     float Lambda_GGXV = NdotL * sqrt((-NdotV * alphaG2 + NdotV) * NdotV + alphaG2);
18     float Lambda_GGXL = NdotV * sqrt((-NdotL * alphaG2 + NdotL) * NdotL + alphaG2);
19
20     return 0.5f / (Lambda_GGXV + Lambda_GGXL);
21 }
22
23 float D_GGX(float NdotH, float m)
24 {
25     // Divide by PI is apply later
26     float m2 = m * m;
27     float f = (NdotH * m2 - NdotH) * NdotH + 1;
28     return m2 / (f * f);
29 }
30
31 // This code is an example of call of previous functions
32 float    NdotV          = abs(dot(N, V)) + 1e-5f; // avoid artifact
33 float3   H              = normalize(V + L);
34 float    LdotH          = saturate(dot(L, H));
35 float    NdotH          = saturate(dot(N, H));
36 float    NdotL          = saturate(dot(N, L));
37
38 // Specular BRDF
39 float3   F              = F_Schlick(f0, f90, LdotH);
40 float    Vis            = V_SmithGGXCorrelated(NdotV, NdotL, roughness);
41 float    D              = D_GGX(NdotH, roughness);
42 float    Fr             = D * F * Vis / PI;
43
44 // Diffuse BRDF
45 float    Fd             = Fr_DisneyDiffuse(NdotV, NdotL, LdotH, linearRoughness) / PI;
```

Listing 2: BSDF evaluation code.

3.2 Material system

3.2.1 Material

Frostbite is used in a wide variety of games, from sports to racing, from first person shooters to open world games. In order to satisfy the different requirements that these games have, the engine needs to offer flexible controls regarding lighting and material support. In addition, one of the constraints during the move to PBR was to ensure compatibility with our old non-PBR lighting model in order to ease the transition. The lighting path is controllable, supporting: deferred, forward, or hybrid. This path will be detailed in Section 4.11.

In *Frostbite* a "material" is defined by:

- A **lighting path**: deferred, forward or both.
- A set of **input parameters**: diffuse, smoothness, thickness, etc.
- A **material model**: rough surface, translucency, skin, hair, etc., as well as non-PBR rough surface. This is the shader code.
- A **GBuffer layout** in case of deferred path support. The number of buffers is variable.

A game team can choose a set of materials from those available for a given light path. Each material is identified with a *materialID* attribute for the game. A *base* material covering the most common cases (which we call the "standard" material) is always present and defines parameters shared with other material (e.g. roughness). For deferred shading, the *base* material is commonly set to the "Disney" model, referring to Burley's model [Bur12]. However, we also support two other base materials: a "two-color" material and an "old" material.

Disney base material: Our Disney material uses the following parameters:

Normal	Standard normals
BaseColor	Defines the diffuse albedo for non-metallic objects and Fresnel reflectance at normal incidence (f_0) for metallic ones, as suggested by Burley's presentation. For metallic objects the lower part of this attribute defines a micro-specular occlusion.
Smoothness	Defines the roughness of an object. We chose to use *smoothness* instead of *roughness* because mapping white to smooth values is more intuitive for artists, and they were already used to it with *Frostbite*'s non-PBR material model. Similar to Burley's presentation, smoothness is remapped into perceptually linear smoothness ($1\text{-}\alpha_{\text{lin}}$).
MetalMask	Defines the "metalness" or conductivity of a surface (i.e. dielectric/conductor), as in Burley's presentation. We named it metal mask to suggest the binary nature of this variable to artists.
Reflectance	Defines the Fresnel reflectance value at normal incidence (f_0) into an artist-friendly range for non-metallic materials (i.e. MetalMask < 1). The lower part of this attribute defines a micro-specular occlusion term for non-metallic materials.

For the **smoothness** remapping, we analyzed different remapping functions and tested them with artists. Figure 12 shows the plot of different remapping functions and results. Similarly to Burley's presentation, we have chosen the "squaring" remapping which seems the most pleasing one for our artists. For the **reflectance**, we chose the following remapping function:

$$f_0 = 0.16 \; reflectance^2$$

The goal was to map f_0 onto a range which could include the high Fresnel values of gemstones, with the constraint of remapping RGB 128 onto the common dielectric 4% reflectance. For gemstones, f_0 goes approximately from 8% for ruby to 17% for diamond. We chose to limit the function to 16% as an approximation. Comparisons with common values are shown in Figure 13. In practice, with our real-time limitations, a variation in f_0 of 1% or 2% is barely noticeable[5]. The fast growing values above 4% fit well with this.

For the Fresnel reflectance at normal incidence (f_0) in both case of non metallic (**Reflectance**) and metallic (**BaseColor**) objects we also use the lower part $< 2\%$ (water reflectance) to provide micro-specular control (See Section 4.10 for more details). Notice that this imply a different range of micro-specular occlusion values to use for non metallic objects due to our particular encoding.

---

[5]Small f_0 changes for dielectrics matter mostly for refracted ray directions which are rarely supported by real time engines.
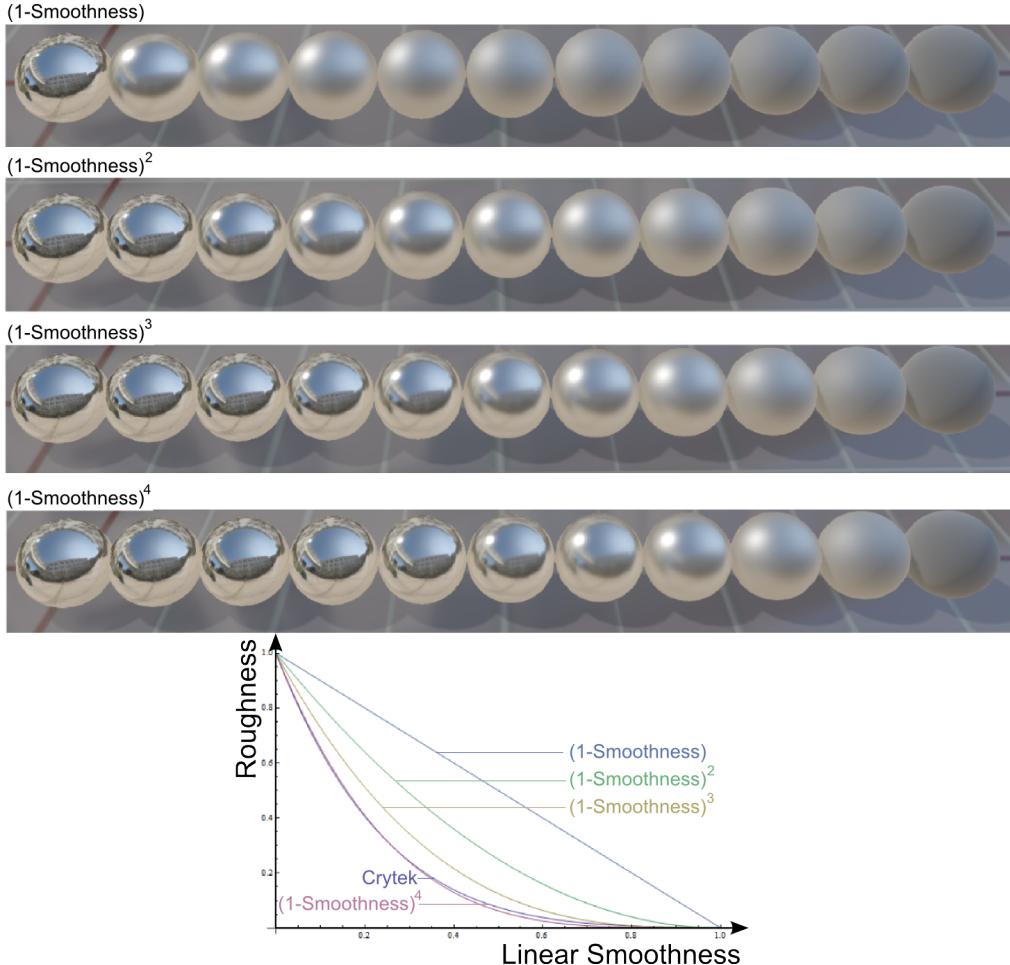
Figure 12: Smoothness remapping comparison. The $(1 - \mathrm{Smoothness})^2$ curve matches Burley's remapping [Bur12]. The dark blue curve matches Crytek's Ryse remapping [Sch14] of $(1 - 0.7\,\mathrm{Smoothness})^6$, but is not shown in the visual results. However, the $(1 - \mathrm{Smoothness})^4$ curve is a very close match.
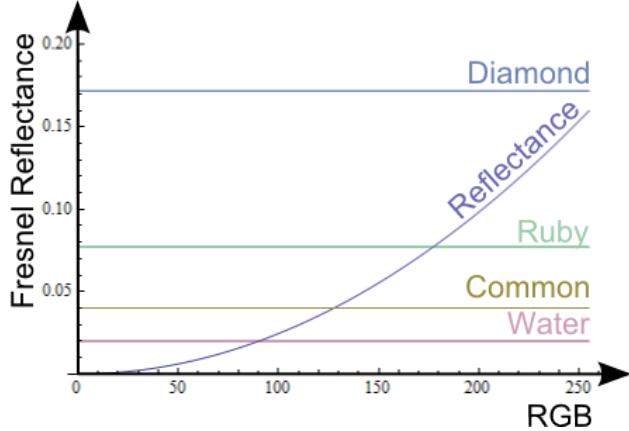
Figure 13: Plot of the remapping function for reflectance, and several reference values.

The Disney material uses the "standard" material model described in Section 3.1. The associated GBuffer layout is shown in Table 2. This layout follows these constraints:

- All basic attributes (`Normal`, `BaseColor`, `Smoothness`, `MetalMask`, `Reflectance`) need to be blendable to support deferred decals. Unblendable attributes, like `MaterialId` are stored into the alpha channel. We have also avoided compression and encoding mechanisms which could affect blending quality[6].
- MSAA support prevents the usage of chroma sub-sampling of the `BaseColor` attribute [MP12].
- Commonly used parameters need to be gathered into the same buffer.
- `MaterialId` needs to be stored at the same place for all materials.
- Performance requirements mean that only four buffers are used for our base/standard material (not including depth).

	R	G	B	A
GB0	Normal (10:10)		Smoothness	MaterialId (2)
GB1	BaseColor			MatData(5)/Normal(3)
GB2	---------	MetalMask	Reflectance	AO
GB3	Radiosity/Emissive			

Table 2: GBuffer layout for Disney deferred base material.

The `MaterialId` allows us to interpret the `MaterialData` (MatData) parameter. For example, for a deferred skin material, it stores a diffusion profile index. For an anisotropic material, it stores the anisotropic strength. The `AO` parameter is an ambient occlusion term which is always present (independent of material type), see Section 4.10 for more details. `Radiosity` is a lighting buffer storing indirect diffuse lighting which is evaluated during the GBuffer creation. The `normal`, split into two parts, uses a lossy encoding method that still allows normal blending (This algorithm may be explain in a future talk).

Two-color base material: The Disney parameterization (using a single base color, a metal mask and a scalar reflectance value) falls short when expressing specular highlights of mixed materials, e.g.

---

[6]For example, encoding just the X and Y components of the normal will fail to blend correctly because of the reconstruction of Z.

at the interface of a metal and a dielectric, like metallic oxides. For this reason, we support a "two-color" deferred base material with a colored f_0 and a diffuse color. The associated GBuffer layout is shown in Table 3. The $f0$ term supports micro-occlusion in its lower range similar to `Reflectance`. Conversion between the two parameterizations are needed depending on the context. The conversion from the Disney's parameterization to this "two color" parameterization is trivial by definition and already happens during our GBuffer unpacking for use in lighting computations. The opposite conversion (i.e. from "two color" parameterization to Disney's parameterization) is more involved as it requires a non-linear optimization. This is only performed when assets need to be converted from one engine mode to another. We provide the details of this conversion in Appendix D.

	R	G	B	A
GB0	Normal (10:10)		Smoothness	MaterialId (2)
GB1	DiffuseColor			MatData(5)/Normal(3)
GB2	f0 Color			AO
GB3	Radiosity/Emissive			

Table 3: GBuffer layout for a two-color deferred base material.

Old base material: is a base material for our old non-PBR engine which has been detailed in [Cof10]. To support legacy content and to ease the transition to PBR, we have added an automatic conversion between non-PBR materials and PBR materials based on basic artistic rules. The conversion happens in shaders before storing input parameters. This automatic conversion gives low quality results compared to properly authored assets. We have not yet found an automatic way to convert these assets whilst retaining quality.

Figure 14 highlights the different transformation of material parameters and shows the dependencies between the material model and the lighting functions. For performance, area lights (Section 4.7) and image based lights (Section 4.9) rely on pre-integration which depend on material models. This implies that **lighting and material models are coupled inside the engine**. The separation between lighting and material claimed by the PBR approach is only valid for asset creation: artists have no access to any lighting information in shaders. But under the hood this separation does not hold and adding a new material often implies adding new lighting code.

3.2.2 Render loop

The previous section provides the definition of a material. This section will describe how materials are rendered. For surfaces using a forward lighting path: we setup the material model shader code, transmit the parameters list, and render the surface. For surfaces using a deferred shading pass, it is more involved. To be able to manage a set of materials efficiently, there are certain considerations:

- Material models try to share the same lighting code as much as possible, relying on dynamic branching for small adjustments. Lighting code that differs too much will require different lighting passes using the stencil buffer.
- Materials try to stay consistent with the base material layout, and rely on dynamic branching for small adjustments for storing parameters. Materials differing too much will require different GBuffer passes, again using the stencil buffer to identify them during the lighting pass.
- We try to share lighting passes with materials having different GBuffer layouts by performing a "fix-up pass".
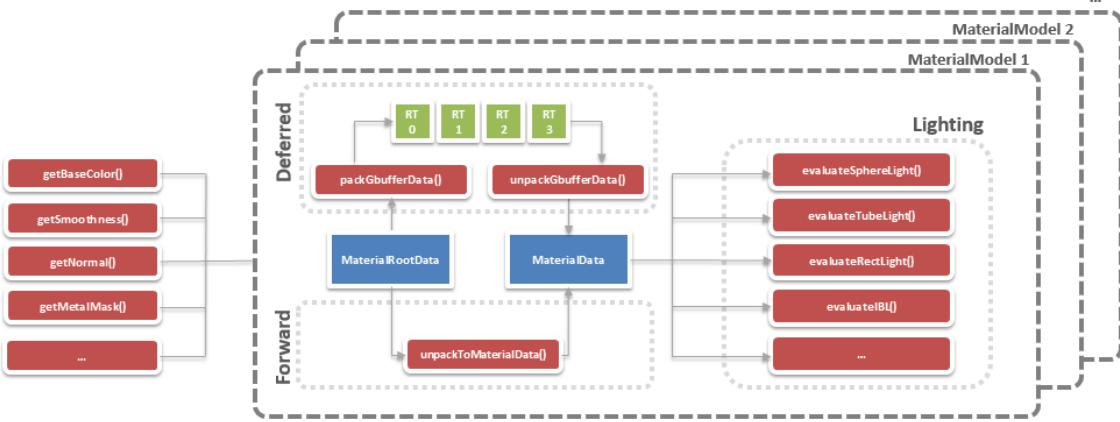
Figure 14: A schematic representation of a Material in *Frostbite*. The list of **input parameters** (MaterialRootData) will be converted to a lighting evaluation structure (MaterialData) which is used to evaluate the **material model**'s lighting function. The lighting function includes all light types: punctual lights, area lights, and IBLs. If a material uses a deferred **lighting path**, it will be packed into a **GBuffer layout** then unpacked and converted. If a material uses a forward **lighting path**, it will be directly converted. Different materials can share the same lighting code if they can be unpacked into the same lighting evaluation structure (MaterialData).

For geometry-heavy passes one wants to reduce the GBuffer creation cost, but still use the same lighting pass. A typical use case is vegetation, for which we fill only the two first buffers of our base GBuffer layout then patch the missing GBuffer parameters later with a "fix-up" pass shown in Listing 3. The fix-up pass may need to be done in multiple passes depending on the hardware capacity for reading and writing into the same buffer.

```
void psMain(
    in float4 pos        : SV_Position ,
    in float2 texCoord   : TEXCOORD0 ,
    out float4 outGBuffer2 : SV_Target0 ,
    out float4 outGBuffer3 : SV_Target1)
{
    float4 gbuffer0 = FETCH_TEXTURE(g_gbufferTexture0 , (texCoord.xy, 0);
    float4 gbuffer1 = FETCH_TEXTURE(g_gbufferTexture1 , (texCoord.xy, 0);
    float3 worldNormal = normalize(unpackNormal(gbuffer0.rg, gbuffer1.a));

    // Read value for radiosity from a global diffuse probe
    float3 indirectLight = calcShL2Lighting(worldNormal , ...);

    // Fix smoothness , metalMask and reflectance with a global common value
    // ambient occlusion set to 1
    outGBuffer2 = float4(g_smoothness , 0, g_reflectance , 1);
    outGBuffer3 = packLightingRGBA(indirectLight);
}
```

Listing 3: Sample GBuffer fixup.

The rendering loop is then:

// GBuffer creation
ForEach different stencil bit **do**
    Render GBuffer pass for deferred material with n number of buffers
        Do branching in shaders if needed
    Render GBuffer pass for deferred material with (n + 1) number of buffers

17

Do branching in shaders if needed

// Fix-up pass
ForEach deferred material requiring sharing the deferred shading pass
Render GBuffer fix-up pass

// Decals
Render deferred decals affecting common parameters

// Deferred shading
ForEach individual deferred shading pass required **do**
Render deferred shading
Do branching in shaders if needed

// Forward rendering and lighting
ForEach ForwardMaterialModel **do**
Render forward lighting

Decals will be discussed in the next section. In practice, it is rather difficult for a game team to customize **everything** due to the coupling between material parameters and lighting functions. The `MaterialId` offers a simple way to customize a material, but with the constraints of having to reuse lighting code and rely on dynamic branching. As an example, a clear coat model which requires sampling the lighting twice has been added by a game team. This system allowed us to perform the transition from a non-PBR engine and fit the different needs of our users.

3.3 PBR and decals

Decals can be seen as a dynamic system for layering material properties, allowing us to create rich appearances and variations. Within this context, the importance of having "correct" decals increases with PBR. By "correct", we mean the ability to properly combine the material parameters of decals and surfaces before any lighting happens, even in the case of multiple overlapping decals. In *Frostbite*, we primarily use deferred shading for rendering decals (deferred decals[7]) for this purpose, but handling them correctly under hardware and performance constraints is almost impossible. We have not found any good solution for blending decals correctly whilst taking into account the variety of physically based materials, but for the sake of completeness we will list some of the major pitfalls and our choice for *Frostbite*.

- **Correctness:** The correctness of blending operations is important, i.e the recovery of an attribute after a blending operation must be correct. For example, for the `Normal` with standard encoding like $normal*0.5+0.5$ or $normal.x, normal.y$ only components, only a replacement (i.e. linear interpolation) will allows to recover correctly the normal. A valid operation like additive detail [BH12] will produce wrong result. This is a good case for programmable blending, but few platforms support it. We chose to avoid compression on few of our attributes and rely on simple, linearly-interpolating alpha blending in this case.

- **Interaction:** To blend correctly, decals and target surfaces should use the same material model (i.e. same list of parameters). From both a production and a performance point of view, this is not manageable, as it would require authoring decals by `MaterialID` and doing a stenciled pass for each case. We chose to restrict decal parameters to the common set of parameters among the

---

[7]In this context, "deferred decal" means both volume decals [Per11] and classic geometric decals.

materials selected by a game. Commonly `Normal`, `BaseColor`, `Smoothness`. Other parameters like `MetalMask`, `Reflectance` (or `f0 color`) are only consider if the Disney (or two-color) base material is chose[8]. However this is not sufficient. Decals affecting other `MaterialID`s than the default will produce artifacts. The `MaterialData` and `MaterialID` are not blendable data and can only be replaced. One option is using the alpha channel's separate blend factor to force a neutral value like 0. Another option is to forbid decals to affect other materials than `MaterialID` = 0.

- **Indirect lighting:** Surfaces need to be affected by decals before any lighting evaluation. However in *Frostbite*, the indirect diffuse is evaluated during the GBuffer creation pass, preventing any further modifications of material properties except diffuse albedo[9]. One solution would be to evaluate all the indirect diffuse lighting in a deferred pass, for instance with light probe volume textures covering the level [Val14]. Another solution would be to use a decal pre-pass, like proposed by UE4 [UE4], to store decal attributes in a buffer before the GBuffer creation. This requires rendering objects that receive decals twice. In *Frostbite*, we chose to leave artifacts due to the overhead implied by such solutions.

- **Emissive:** Due to limited GBuffer storage in *Frostbite*, emissive information is sometime combined with the radiosity buffer for performance reasons, see Section 4.8. The radiosity buffer is combined later with the albedo. Decals modifying the albedo will modify the emissive color in this case. In *Frostbite* we have accepted artifacts dealing with emissive and decals.

- **Highlight shape preservation and specular aliasing:** Any normal parameter modification implies a change of the NDF beneath the pixel footprint. This means that it will bias any shape preservation (handled by techniques like Toskvig or LEAN mapping) which has been performed before the decal application, see Section 5.3. A solution is to perform a normal filtering pass[10] as a post-process after the decal application [Sch14].

- **Forward rendered surfaces:** Deferred decals are not compatible with transparent and any forward rendered objects. The common solution in this case is to rely on forward decals that blend their lit results with surface lighting. Supporting decal parameter blending for forward rendered objects would be possible but would require heavy constraints on the art side (Limited blending mode, restrict texture size for texture array, etc.). This problem shares some similarities with lighting application. Applying solutions already found for lighting could open the way to tiled-deferred and tiled-forward decals.

---

[8]We also support emissive decals as a forward case, i.e. not manipulating the GBuffer parameters.

[9]In *Frostbite*, the radiosity buffer is composed with the albedo after the lighting pass.

[10]A normal filtering pass implies a modification of the roughness.

4 Lighting

4.1 General

A lighting pipeline must respect important foundations which have already been extensively discussed in the literature. The lighting pipeline should support high dynamic range (HDR) and lighting processing must be done in linear space [Gd08]. All inputs and outputs of the pipeline should be gamma corrected (mip-mapping, blending, filtering, etc.). For *Frostbite*, like all game engines, we chose to rely on the sRGB convention due to its hardware support[11].

A believable scene is based on the coherence and correctness of the lighting: every object should receive lighting from its surrounding environment, reflect light with the right amount of intensity and have shadows. Game engines often provide a lot of lighting tools without link between them. Artists have difficulty manipulating one lighting tool without having to re-tweak other lighting components to get the correct result, thus breaking credibility and spatial reference. The main guideline we have with *Frostbite* is to have everything **correct by default** then give the possibility for artists to tweak what they want to. It should be harder for an artist to deviate from correct result than to achieve correct results. But artistic controls must not be forgotten to work around engine limitations and for artistic reasons.

Frostbite supports several types of lights: punctual lights, photometric lights, area lights, emissive surfaces, and image based lights (IBLs). In the term "IBLs" we include distant light probes (representing the sky), as well as localized light probes, and screen space reflections (SSR). Coherence is achieved when all lighting components are coupled and interact correctly with the material properties. Here are some examples:

- **Coherent material lighting**: All BSDFs should interact correctly with all light types. This includes direct lighting, such as punctual lights and area lights, as well as indirect lighting, such as IBLs. For example, our rough diffuse material is supported by all light types and our radiosity system, in order to reveal its specific appearance.
- **Coherent indirect-diffuse lighting**: All light types must be taken into account by the radiosity system. The sun and sky lighting are a very important part, but every other light types should be included as well.
- **Coherent indirect-specular lighting (i.e. reflections)**: SSR, localized light probes, and distant light probes must be correctly combined together.
- **Coherent light units**: All light should be expressed in the same units in order to achieve realistic ratios. For instance, mistakes are easily made when dealing with HDR captured light probes, preventing achieving a good ratio with analytical lights.
- **Coherent decals**: Decals should be correctly affected by all light types, including the indirect lighting.

The following sections will describe all the light types supported in *Frostbite* and how we try to maintain lighting coherency within the engine. We have not solved all the cases and thus pitfalls will be detailed. For example, see Section 3.3 for problems related to decals.

4.2 Analytical light parameters

For artists' convenience, in *Frostbite*, punctual and area lights share the same interface and settings. A part of these settings are shown in Figure 15. We choose to separate light hue (referred to as

---

[11]Our artists all have their monitors calibrated for sRGB.

color) from intensity. We will use the word *intensity* to refer to the amount of energy emitted by a light, not as its strict definition. In order to differentiate the various hues of white, artificial light sources are labeled with either a color temperature (incandescent and tungsten halogen) or a correlated color temperature (CCT) (nearly everything else). Color temperature is the temperature of an ideal black-body radiator[12] that radiates light of comparable hue. CCT is the color temperature of a black-body radiator which to human color perception most closely matches the light from the lamp. Color temperature and CCT are usually measured in Kelvin (K). For simplification in this document we use the term color temperature to refer to either color temperature or CCT. We also call our front facing parameter **color temperature**. Table 4 shows different light types with their associated temperature and perceived color.
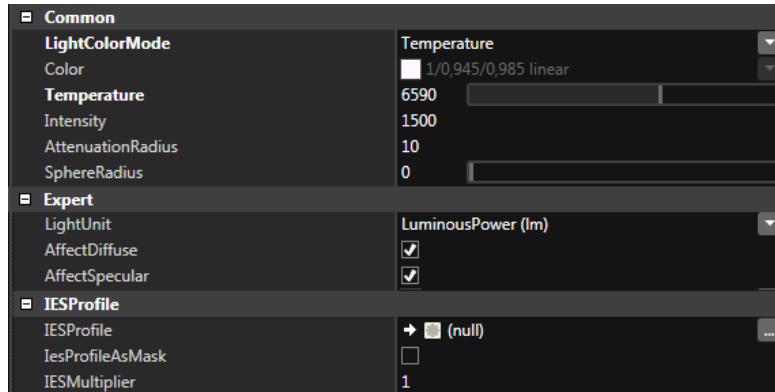

Figure 15: A sample of the *Frostbite* light settings.

A spectral renderer (i.e. a renderer which uses spectral values instead of a RGB triplet to represent color) could directly use the radiation spectrum defined by a black body [13]. In *Frostbite*, we chose to only retrieve the hue from the color temperature [14] and let artists control the light intensity independently. Retrieving the hue from a color temperature is a relatively complex operation and is explained by Charity in [**Charity**]. Artists can also specify an RGB color.
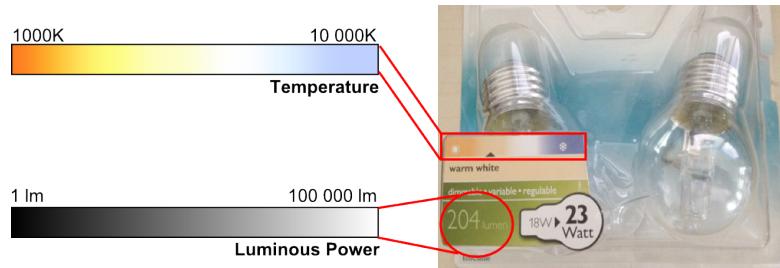

Figure 16: Information found on light bulb packages can be directly put into *Frostbite*: color temperature and luminous power.

The "light intensity" parameter will be described for each light type. With both color temperature and intensity, an artist is able to pick reference values from a manufacturers' website and drop them directly

---

[12]A black body is an idealized physical body that absorbs all incident electromagnetic radiation, regardless of frequency or angle of incidence.

[13]The spectrum represents both the hue and the intensity at the same time.

[14]The color temperature from black body radiators does not cover the entire range of a real-life lights' colors. For instance, fluorescent lamps with bright phosphors do not follow this theory and generate green or blue colors. However, most of the time simple tinted bulbs are used.

Degrees Kelvin	Type of light source
1700-1800K	Match flame
1850-1930K	Candle flame
2000-3000K	Sun at sunrise or sunset
2500-2900K	Household tungsten bulb
3000K	Tungsten lamp 500W-1k
3200-3500K	Quartz lights
3200-7500K	Fluorescent lights
3275K	Tungsten lamp 2K
3380K	Tungsten lamp 5K, 10K
5000-5400K	Sun direct at noon
5500-6500K	Daylight (Sun + Sky)
5500-6500K	Sun through clouds/haze
6000-7500K	Sky overcast
6500K	RGB Monitor (white point)
7000-8000K	Outdoor shade areas
8000-10000K	Sky partly cloudy

Table 4: Color temperature and perceived color of several lights. Temperatures ranging from $2700°K$ - $3000°K$ are called warm colors, $3500°K - 4100°K$ neutral colors and $> 5000°K$ cool colors.

into *Frostbite*. See Figure 16. Light settings include also the usual attenuation range and control over the physical light size. The light's physical size (e.g. sphere radius, disc radius, tube length etc.) allows an artist to define if the light will be an **area light** or a **punctual light**. Figure 17 shows the lights supported in *Frostbite*: point and spot lights are the only punctual lights, as all the other lights are considered as area lights. Separating punctual lights and area lights matters for performance and *Frostbite* supports a smooth transition between these two types.

4.3 Light unit

To have coherent lighting, it is necessary to respect the ratio of light intensities, and thus to have a common unit system. Light intensity can span a large range, as shown in Figure 18, and it is important to conserve it. The perceived richness of a scene comes from a right balance of lighting. Figure 19 shows a mix of indoor and outdoor lighting. The exposition process will transform this wide range of intensities into a normalized pixel value, at the end of the pipeline. See Section 5.1.

Usually lighting artists set up light ratios by arbitrarily defining a reference, like the sun. This reference is set to emit a certain intensity (usually a small number, around 5 to 10), and the values of all other light sources will be based upon it. However most of the time these settings are biased by the current scene context (underground, indoor, outdoor, etc.) making the lighting rig unusable for other scenes. To introduce correct lighting ratios, we have adopted physical units for our lights in *Frostbite*. This allows us to:
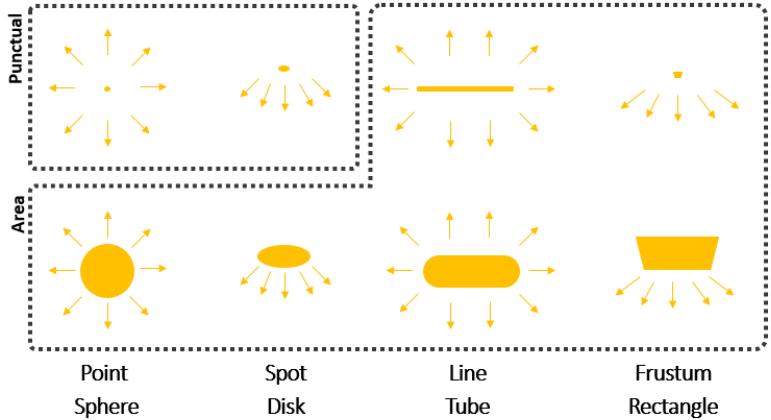
Figure 17: Four different light types supported by *Frostbite*. Categorization depends on the light's physical size. Point and spot lights are punctual while all other lights are considered as area lights.
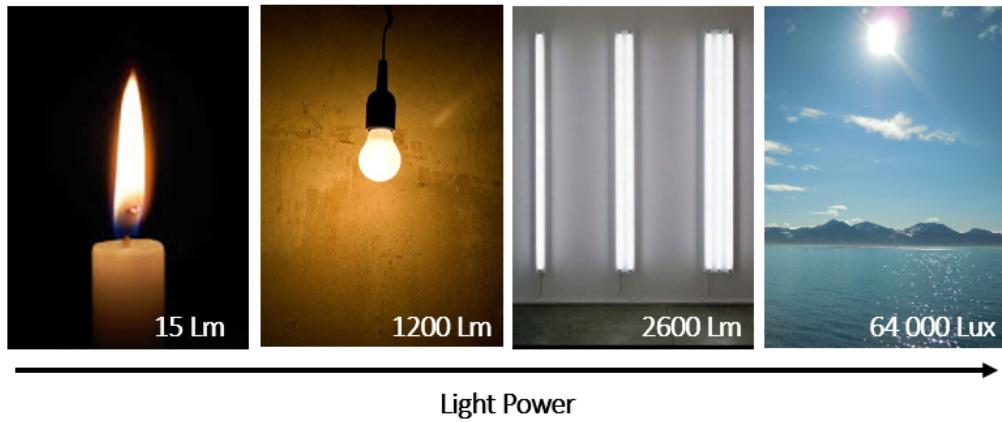

Figure 18: Miscellaneous light sources which have an increasing luminous power.

- Deal coherently with different light types.
- Reuse lighting rigs in several places.
- Get better responses from physically based materials: high contrast lighting helps to reveal material richness.
- Be able to use a physically based camera and rely on a photographer's knowledge. See Section 5.1

Light units are related to light measurements, which are split into two categories:

- **Radiometric**: Deals with "pure" physical quantities and used in the context of optical radiation measurement and spectral rendering[15].

- **Photometric**: Concerned only with radiation falling within the visible spectrum.

Quantities derived in radiometry and photometry are closely related: photometry is essentially radiometry weighted by the sensitivity of the human eye. These two forms have been widely covered in the literature [Rei+08]. The most commonly used radiometric and photometric quantities are listed in Table 5. The energy subscript e is used for radiometric quantities and the visual subscript v is used

---

[15]Optical radiation is radiation that obeys the principles of optics whose wavelength region approximately includes ultraviolet, visible and infrared radiation.

Figure 19: Indoor and outdoor lighting can be coherently combined with the right units.

for photometric quantities.

The sensitivity of the human eye is represented by the CIE photometric curve $V(\lambda)$. It follows a bell-shaped curve which represents how efficiently our eyes pick up certain light wavelengths, see Figure 20. The sensitivity of human eyes peaks at 555nm which appears to us as green. At this wavelength the sensitivity function value is 1 unit, meaning 100% efficiency. Photometric quantities are related to radiometric quantity with the following integration over the visible spectrum (380nm to 780nm):

$$X_v = K_m \cdot \int_{380}^{780} X_e(\lambda) V(\lambda) d\lambda \tag{7}$$

The constant K_m is called *the maximum spectral luminous efficacy of radiation for photoscopic vision* and its value is based on the definition of candela[Wikb] which is the SI unit for luminous intensity measurement. Its definition is: *one candela is the luminous intensity, in a given direction, of a source that emits monochromatic radiation at a frequency of 540THz (i.e a wavelength of 555nm) and whose radiant intensity in that direction is 1/683 watts per steradian.*, meaning that $K_m = 683$. When dealing with light bulb intensity, usually expressed in radiant power, the above formulation can be reworded into a simple phrase: *"1 watt of a green 555nm light is 683 lumens"*. The photometric curve also allows us to deduce the luminous efficacy[16] for a light. This can be interpreted as *how much visible light is produced by a light source*. The formula for calculating luminous efficacy is as follows:

$$\eta = 683 \cdot \frac{\int_{380}^{780} X_e(\lambda) V(\lambda) d\lambda}{\int_{380}^{780} X_e(\lambda) d\lambda} \tag{8}$$

---

[16]Remark: Luminous efficacy is in lumens per watt while luminous efficiency is a percentage.

Quantity	Radiometric term	Units	Photometric term	Units
Energy	Radiant energy Q_e	J (Joule)	Luminous energy Q_v	$lm.s$
Power	Radiant flux Φ_e or Radiant power	$\frac{J}{s}$ or Watt(W)	Luminous flux Φ_v or Luminous power	Lumen (lm)
Power per solid angle	Radiant intensity I_e	$\frac{W}{sr}$	Luminous intensity I_v	$\frac{lm}{sr}$ or Candela (cd)
Power per area	Radiant exitance M_e or Irradiance E_e	$\frac{W}{m^2}$	Luminous exitance M_v or Illuminance E_v	$\frac{lm}{m^2}$ or Lux (lx)
Power per area per solid angle	Radiance L_e	$\frac{W}{m^2.sr}$	Luminance L_v	$\frac{lm}{m^2.sr} = \frac{cd}{m^2}$ or Nit (nt)

Table 5: Radiometric and photometric quantities.

Incandescent lights	Between 2% and 5%
LED lights	Between 0.66% and 8.8%
Fluorescent lights	Between 9% and 15%
The sun	Between 15% and 19%

Table 6: Luminous efficiency (in percent) of a few lights.

where η is in lumens (lm) per watt (W). This formula means that a green (555 nm) light has an efficacy of 683 lm/W. This is equivalent to a luminous efficiency of 100%. Luminous efficiency can differ between light types. The consequence is that two lights with the same wattage can produce different perceived intensity. Table 6 show luminous efficiency of few different lights.

In the context of non-spectral rendering, like games, some simplifications can be made to ease conversion between radiometric and photometric units. Providing the radiant power and luminous efficacy of a light allow us to convert its power from watts to lumens.

$$\Phi_v = \eta \cdot \Phi_e \tag{9}$$

When this information is not available, it is common to assume that a light is 100% efficient with $\eta = 683$. This formula is often extended to other quantities:

$$X_v = 683 \cdot X_e \tag{10}$$

In a spectral renderer, lights need to be described in radiometric units. Successive operations that use spectral weighted radiance are only correct within the radiometric domain. Subsequent stages to convert this spectral radiance into a pixel value often requires a photometrically weighted process
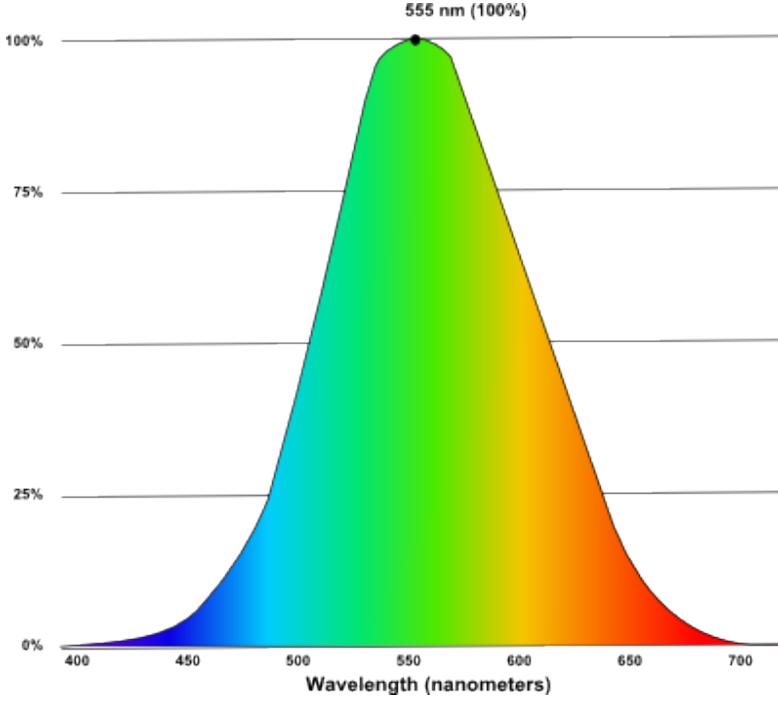
Figure 20: The sensitivity curve of the human eye.

(including auto-exposure or tone mapping).

Radiometry or Photometry? With non-spectral rendering, like most game engines, the luminous efficacy information is required for going back and forth between radiometric and photometric units. As it increases complexity for artists to provide two values (light intensity and luminous efficacy), it is preferable to use the hypothesis $\eta = 683$. With such an approximation, each quantity is a linear transform of the other and thus their processing will be identical. However it is desirable to be able to use real world light intensity in the engine to perceptually match what one observe in the real world. The assumption of 100% luminous efficacy prevents the usage of radiometric reference values, because this would produce overly bright lighting. Fortunately, commercial lights provide their characteristic in photometric quantity. In addition, as discussed in Section 4.9.1, HDRI are often provided in photometric units (luminance).

In *Frostbite*, all lights use photometric units, implying that our rendering pipeline stores luminance values in render targets. Lighting intensity can also be expressed in a relative way (the ratio of different light types) or in absolute way (true light units). For *Frostbite*, we found it easier to deal with absolute values in order to be able to match values provided on a light manufacturer's website and any reference data.

Remark: Some devices can measure emitted light. For instance, with an incident light meter, one can measure the actual illuminance reaching a surface, Figure 21. Luminance can also be measured with a spot/luminance meter[17]. This is essentially an incident light meter with an opaque shield that limits the incident light to a single direction. We can not measure luminous intensity directly. Instead, we must measure illuminance at a known distance from the source and calculate its equivalent luminous

---

[17]Luminance meters are often a more expensive device because they are more accurate. Their precision is based on the size of the probed angle.

intensity from the inverse square law $luminousIntensity = illuminance\ distance^2$.

Figure 21: Left: an incident meter, notice the white ball. Right: a reflective (spot) meter used as a telescope to get a measurement. Some meters combine both measurement into one device.

Exposure Value: Exposure value (EV) is often used by artists with photographic knowledge to describe light intensity [Ree14]. Originally, EV defines a set of camera settings (a combination of aperture, shutter speed, and sensitivity) for a given shot. With the help of a spot meter[18] photographers are able to determine automatically adapted camera settings for maximizing the camera film usage, see Section 5.1 for more details. But EVs are nowadays abused from their primary usage and used as a light unit. Expressed in a base-2 logarithmic scale, called **f-stop**, one positive step corresponds to a factor of two in luminance (twice the brightness), one negative step corresponds to a factor of a half (half the brightness). This scale is almost perceptually linear, meaning that the difference between +0 and +1 EV looks almost the same as in between +1 and +2 EV. EVs were not designed as a light unit making their definition dependent on a per-device calibration constant K:

$$EV = \log_2(\frac{L_{\text{avg}}\ S}{K}) \tag{11}$$

where L_{avg} is the average scene luminance, S is the ISO arithmetic, and K is the reflected-light meter calibration constant. ISO 2720:1974 recommends a range for K between 10.6 and 13.4. Two values for K are in common use: **12.5** (Canon, Nikon, and Sekonic) and **14** (Minolta, Kenko, and Pentax)[Wikg]. $K = 12.5$[19] seems to be the most common adopted value among renderers [Wikc]. Table 7 shows the correspondence between EV and luminance[20]. Having a base-2 logarithmic light unit for area/emissive lights is something desirable to deal with high values. As artists are used to EV units, we decided to adopt EV with $K = 12.5$ as an optional unit. Thus to convert from EV to luminance:

$$L = \frac{2^{EV}\ 12.5}{100} = 2^{EV-3} \tag{12}$$

Table 8 shows the light units used for the different light types supported by *Frostbite*. The respective advantage of each unit for a light type will be explained in following sections.

---

[18]Spot meters are devices measuring the average luminance of a scene and converting it to an EV.

[19]The value 12.5 is roughly half a stop below middle grey to provide some headroom: $18\%/\sqrt{2} = 12.7\%$.

[20]The definition of EV is dependent on ISO sensitivity. When using EV as a light unit, one refers to EV at ISO 100, noted EV_{100}. For more details see Section 5.1.

EV_{100}	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Luminance	0.008	0.016	0.031	0.063	0.125	0.25	0.5	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192

Table 7: EV_{100} value and corresponding luminance($\frac{cd}{m^2}$) for K = 12.5.

Type	Units
Area	Luminous power (lm), Luminance $(cd.m^{-2})$, or EV
Punctual	Luminous power (lm)
Photometric	Luminous intensity (cd)
Emissive surfaces	Luminance $(cd.m^{-2})$ or EV
Sun	Illuminance (lx)
Sky and Image Based	Luminance $(cd.m^{-2})$

Table 8: Light units associated with various *Frostbite* light types.

4.4 Punctual lights

Frostbite supports only two types of punctual lights: point and spot. For a punctual light to be physically correct, it must follow the so-called *"inverse square law"* [Wikf], see Figure 22. The observed light intensity from a source of constant luminosity decreases proportional to the square of the distance from the object. The inverse square law is only valid for point lights and does not apply to:
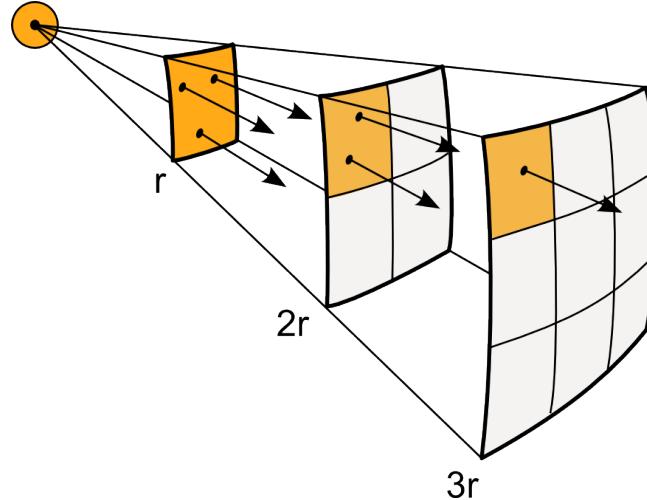
Figure 22: Inverse square law: the arrows represent the flux emanating from the source. The density of arrow per unit area (orange square), i.e. the density flux, is decreasing with squared distance.

1. Floodlights and searchlights with narrow distributions because the light beam is highly collimated.
2. Area lights or specific lights like the Fresnel lens.

Luminous power: Translating the inverse square law into an equation gives:

$$E = \frac{I}{distance^2} \tag{13}$$

with luminous intensity I and illuminance E[21]. The equation requires that distance units are homogeneous (m, cm, mm,...) across the lighting calculation. The values can go to infinity when the distance tends to 0 but this will not happen in real life because a light will always have a size. However in real-time graphics it is common to add a small bias to avoid numerical issues. A good rule of thumb is to consider punctual lights as having a minimum size and never let objects penetrate into them. In *Frostbite* 1 unit = 1 meter and we define punctual lights to have a size of 1 cm (to simplify the notation, we will omit the "max" operator in the reminding parts of this document):

$$E = \frac{I}{\max(distance^2, 0.01^2)} \tag{14}$$

In *Frostbite*, artists are allowed to control the punctual light intensity either with luminous power units or luminous intensity units for the photometric profile, see Section 4.5. The luminous power is always converted into luminous intensity for lighting calculations. The luminous power can be calculated from luminous intensity by integrating the luminous intensity over the light solid angle.

- **Point light**:

$$\phi = \int_S I \, d\mathbf{l} = \int_0^{2\pi} \int_0^\pi I \, d\theta d\phi = 4\pi \, I \tag{15}$$

- **Spot light** (with an opening angle $0 \leq \theta_{\text{outer}} \leq \pi$):

$$\phi = \int_\Omega I \, d\mathbf{l} = \int_0^{2\pi} \int_0^{\theta_{\text{outer}}} I \, d\theta d\phi = 2\pi(1 - \cos\frac{\theta_{\text{outer}}}{2}) \, I \tag{16}$$

Equation 16 is the exact solid angle of a cone. If θ_{outer} is the half angle, the integral becomes $I \, 2\pi(1 - \cos\theta_{\text{halfouter}})$. The consequence of this formulation[22] is that for equal values, the illumination level will look brighter with smaller cone angles since the lighting will be more focused, see Figure 23. Such a coupling makes spotlight manipulation harder for artists and causes trouble when optimizing[23]. We chose to drop focused light coupling and act as if the spot reflector would absorb light, acting like a simple masking. With these considerations, and to smooth the transition with the disk area light, we have defined the luminous power of a spotlight in *Frostbite* as:

$$\phi = \pi I \tag{17}$$

The conversion from luminous power to luminous intensity for punctual lights is summed up in Table 9. For completeness, we also add the conversion for a frustum light [Wiki] (i.e., a rectangular pyramid). Frustum lights and line lights are considered as area lights in *Frostbite*, so they do not use these formulae[24]. They are used only occasionally and are not able to share the efficient lighting path of point and spot lights.

Light calculation: The lighting calculations for these light sources are given as follows:

- **Point light**: light evaluation can be expressed as

$$L_{out} = f(\mathbf{v}, \mathbf{l}) \, E = f(\mathbf{v}, \mathbf{l}) \, L_{in} \, \langle \mathbf{n} \cdot \mathbf{l} \rangle = f(\mathbf{v}, \mathbf{l}) \, \frac{I}{distance^2} \, \langle \mathbf{n} \cdot \mathbf{l} \rangle = f(\mathbf{v}, \mathbf{l}) \, \frac{\phi}{4\pi \, distance^2} \, \langle \mathbf{n} \cdot \mathbf{l} \rangle \tag{18}$$

---

[21] In this document all light transfer equations use photometric terms but this is also applicable to radiometric terms.

[22] Krivanek [Kři06] proposes another derivation based on the common inner and outer angles often used in game development for spotlights.

[23] Artists cannot reduce the cone attenuation to affect fewer pixels without changing its intensity.

[24] We do not provide a formulation for punctual line lights because we have not found one.
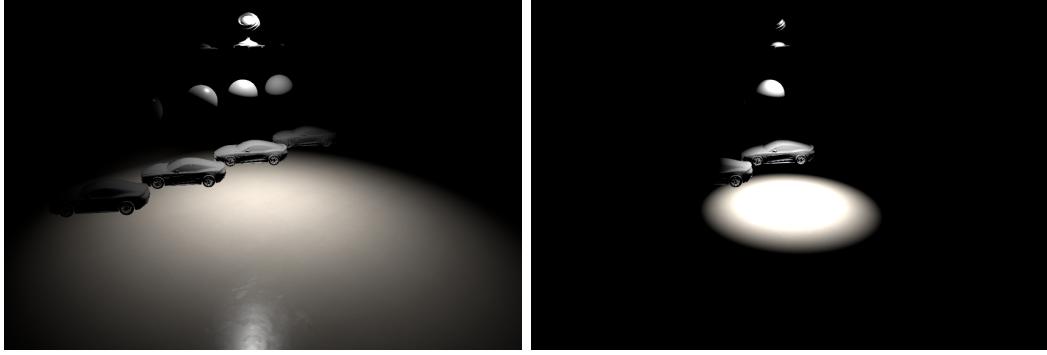
Figure 23: With luminous power, decreasing the aperture of the cone causes an increase of illumination level inside the cone.

Point	$\frac{\phi}{4\pi}$
Spot	$\frac{\phi}{2\pi(1-\cos\frac{\theta_{\mathrm{outer}}}{2})}$ or $\frac{\phi}{\pi}$
Frustum	$\frac{\phi}{4\arcsin\left[\sin(\frac{\theta_a}{2})\sin(\frac{\theta_b}{2})\right]}$

Table 9: Conversion from luminous power (lm) to luminous intensity (cd) for punctual and frustum lights. For spot lights, θ_{outer} refers to the opening angle. For frustum lights, θ_a and θ_b refer to the frustum opening angles.

Considering a surface normal pointing directly to the light, the resulting illuminance following the inverse square law is:

$$E_{\perp} = \frac{\phi}{4\pi \; distance^2} \tag{19}$$

- **Spot light**: light evaluation can be expressed as

$$L_{out} = f(\mathbf{v},\mathbf{l}) \; \frac{I}{distance^2} \; \langle \mathbf{n} \cdot \mathbf{l} \rangle = f(\mathbf{v},\mathbf{l}) \; \frac{\phi}{\pi \; distance^2} \; \langle \mathbf{n} \cdot \mathbf{l} \rangle \; \mathrm{getAngleAttenuation}() \tag{20}$$

The resulting illuminance follows the inverse square law, and considering a surface normal pointing directly to the light, is thus (with *Frostbite* convention):

$$E_{\perp} = \frac{\phi}{\pi \; distance^2} \tag{21}$$

Measurement: To validate both relationships we have performed a set of measurements with an incident light meter. The set-up is explained in Figure 24. Results are provided in Table 10, which validated our handling of luminous power units and our assumptions about point light sources being able to represent a light bulb. Note how the light having double the amount of lumens has twice the illuminance. The nearest possible measurement is a measurement against the light bulb to highlight the highest value we can get, and thus depends on the size of the light bulb. We have not performed any measurements for spot lights, as the set-up is more complex and does not fit with our definition of a spot light. Point lights in *Frostbite* are our reference lights for all intensity calibration. They match real world measurements and thus are trusted.

Attenuation: One issue with the inverse square law is that it never reaches zero. For performance reasons, the renderer must implement a finite light range to support light culling algorithms. At a certain limit, the illuminance should smoothly reach zero. One approach to solve this is to window the falloff in such a way that most of the function is unaffected [Kar13]. For this we will use a basic linear interpolation to zero based on distance criteria:

30

Figure 24: Lighting measurement setup: located in a sound room with no windows; walls are covered by a sound-reducing material with low albedo that reduces light bounce. The bottom right image shows that the light bulb was self-shadowing at the bottom, thus the intensity is not fully isotropic. Consequently, the measurement was taken horizontal to the filament. For each measurement, we waited a certain amount of time to let the light bulb warm up. Note that the incident meter is very sensitive to any obstructions (like a human body) or reflections (like a white surface). All measurements of distance are from the sensor of the incident light meter to the filament inside the light bulb.

Light bulb brand	Watts	Lumens	1 m	50 cm	25 cm	10 cm	5.5 cm	Nearest
Dial		625	57	213	940	5 800	17 000	60 000
Frostbite		625	50	199	796	4 973	16 441	?
Philips	70 to 92	1 200	122	440	1 800	12 000	33 000	130 000
Frostbite		1 200	95	382	1 528	9 549	31 568	?

Table 10: Real world measurements of light bulbs with an incident meter compared to in-game values..

$$E = \text{lerp}(\frac{I}{distance^2}, 0, \frac{distance}{lightRadius}) = (\frac{I}{distance^2})(1 - \frac{distance}{lightRadius}) \tag{22}$$

and to keep the function unaltered, we tweak the distance criteria as follows:

$$E_{window2} = (\frac{I}{distance^2})(1 - \frac{distance^{20}}{lightRadius^{20}}) \tag{23}$$

This simple approach works but causes a hard cutoff which looks unnatural. A second approach is to bias the function by a threshold and remap it to its initial range (0-1) [Mad11].

$$threshold = \frac{1}{lightRadius^2} \tag{24}$$

$$E_{scaleBias} = (\frac{1}{1 - threshold})(\frac{I}{distance^2} - threshold) \tag{25}$$

Results are better but this approach suffers from having a non-zero gradient at 0 which causes a visible discontinuity. A better approach is to window the function and ensuring zero gradient at *lightRadius*. This can be achieved by raising the power of the windowing function. See Figure 25.
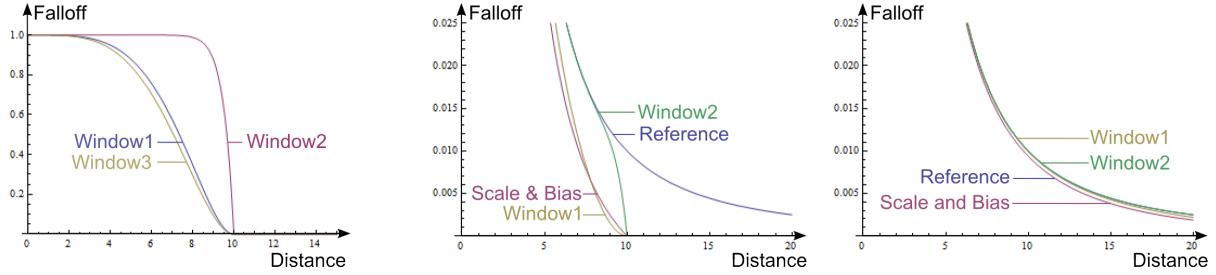
Figure 25: Illuminance: The first graph shows different windowing functions to highlight the discontinuity when crossing the distance axis. Window1 is the one used in [Kar13], Window2 is with the discontinuity problem and Window3 is a smoothstep. The second and third graphs show the windowing functions applied to a falloff of light radius respectively of 10 and 40. This highlights how well the curves fit the reference as the light radius increases.

$$E_{window1} = (\frac{I}{distance^2}) \text{ saturate}(1 - \frac{x^n}{lightRadius^n})^2 \qquad (26)$$

n allows us to tweak the transition smoothness. We have adopted Window1 with value $n = 4$ for *Frostbite*, as presented by Karis [Kar13]. This smoothing function has been adopted for all punctual lights as well as all area lights. However, as the lerp criteria is radial ($\frac{x^2}{d^2}$) it does not work well with non-spherical shapes like tube or rectangular lights. We accept this trade-off for performance reasons. Artists still have the possibility of increasing the light radius if accuracy is needed.

Listing 4 shows the resulting code of the different subjects discussed in this section.

```
1  float smoothDistanceAtt(float squaredDistance, float invSqrAttRadius)
2  {
3      float factor = squaredDistance * invSqrAttRadius;
4      float smoothFactor = saturate(1.0f - factor * factor);
5      return smoothFactor * smoothFactor;
6  }
7
8  float getDistanceAtt(float3 unormalizedLightVector, float invSqrAttRadius)
9  {
10     float sqrDist = dot(unormalizedLightVector, unormalizedLightVector);
11     float attenuation = 1.0 / (max(sqrDist, 0.01*0.01));
12     attenuation *= smoothDistanceAtt(sqrDist, invSqrAttRadius);
13
14     return attenuation;
15 }
16
17 float getAngleAtt(float3 normalizedLightVector, float3 lightDir,
18                   float lightAngleScale, float lightAngleOffset)
19 {
20     // On the CPU
21     // float lightAngleScale = 1.0f / max(0.001f, (cosInner - cosOuter));
22     // float lightAngleOffset = -cosOuter * angleScale;
23
24     float cd = dot(lightDir, normalizedLightVector);
25     float attenuation = saturate(cd * lightAngleScale + lightAngleOffset);
26     // smooth the transition
27     attenuation *= attenuation;
28
29     return attenuation;
30 }
31
32 // Process punctual light
33 float3 unormalizedLightVector = lightPos - worldPos;
34 float3 L = normalize(unormalizedLightVector);
35 float att = 1;
```

```
36
37  att *= getDistanceAtt(unormalizedLightVector, lightInvSqrAttRadius);
38  att *= getAngleAtt(L, lightForward, lightAngleScale, lightAngleOffset);
39
40  // lightColor is the outgoing luminance of the light time the user light color
41  // i.e with point light and luminous power unit: lightColor = color * phi / (4 * PI)
42  float3 luminance = BSDF(...) * saturate(dot(N, L)) * lightColor * att;
```

Listing 4: Attenuation.

4.5 Photometric lights

Photometric lights use a photometric profile for describing their intensity distribution. These distributions are stored in a photometric file. Two common formats exist: *IES* (.ies) and *EULUMDAT* (.ldt) which are simple ASCII files.

- **IES**: stands for Illuminating Engineering Society and was created for the electronic transfer of photometric data over the web. IES is the most popular photometric data file format in North America and is also widely used in Europe.

- **EULUMDAT**: is the European standard and is the de-facto industry standard photometric data file in Europe but there is no associated recognized standards organization to define and maintain it.

Many lighting manufacturers provide photometric files which are freely available on their websites. For instance the Lithonia Lighting (www.lithonia.com) manufacturer has an extensive library of IES and EULUMDAT files.
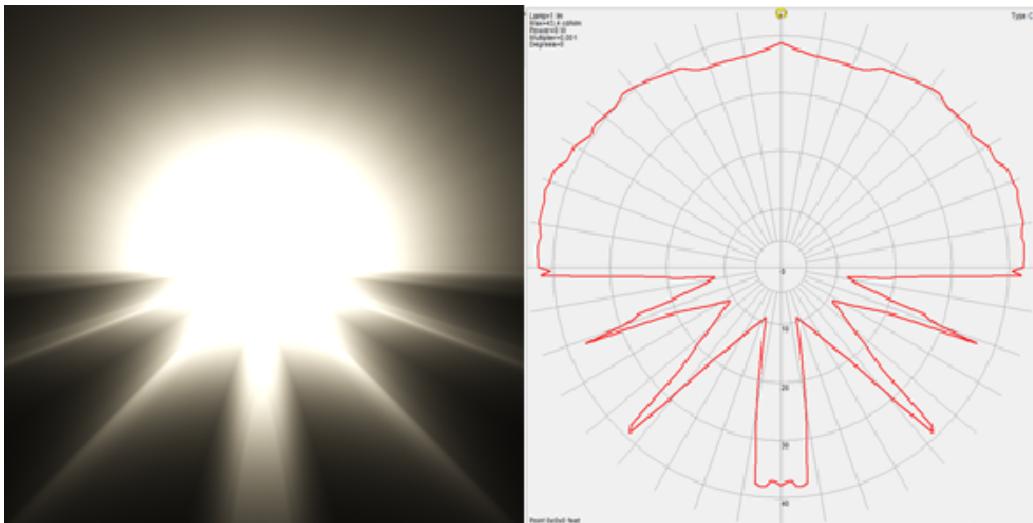
Figure 26: An example of an IES render and drawing of lighting distribution.

Both formats store luminous intensity for various angles. They are measured in laboratories using light sensors, spread spherically around a light source and pointing toward the center of the sphere. Due to this capturing method, all lamps are considered as point sources, see Figure 26. For getting physically correct results, IES/EULUMDAT files should only be applied on small spheres. Applying an IES/EULUMDAT file onto geometry other than a small sphere can cause inaccuracies inherent to the method and format definition. The spherical coordinate system used to describe light distribution

is referred to as *"the photometric web"*. There are three different types of photometric webs, called Types A, B, and C.

- **Type A**: for automotive headlamp and signal lights,
- **Type B**: for adjustable outdoor area and sports lighting luminaires,
- **Type C**: for architectural and road lights.

In practice types A and B are rarely used, and type C is the most common type used in computer graphics.

The **IES** format stores luminous intensity values in candela in a format whose structure can be hard to understand. This applies for both the absolute and relative intensities supported by this format. Absolute intensity is required for LED fixtures and relative intensity is the most common case in computer graphics and for traditional lights (fluorescent, incandescent, metal halide, etc). There are five different specifications to this format, from IES LM-63-86 to IES-LM-63-2002. Writing an IES parser supporting every format is not an easy task and several files used in the computer graphics world seem to be incomplete or ill formatted. The specifications are not freely accessible and have a few obscure areas. In particular, there is some confusion in the IESNA literature regarding photometric types A and B with the direction of horizontal rotation. The best resource today regarding IES seems to be the Helios32 website [Con14] and his associated documents. A good resource for writing an IES parser is written by Ian Ashdown [Ash98] who maintains the Helios32 website. To visualize IES files, the most popular viewer is IESviewer [Leg]. In contrast, the **EULUMDAT** format is a well-structured format that stores luminous intensity values in candela per total kilo-lumens emitted by the light.

$$\text{candela per total kilolumens } \left(\tfrac{cd}{klm}\right) = \tfrac{1000 \text{ luminous intensity } (cd)}{\text{total luminous flux } (lm)}$$

This distinction is important and requires some clarification. An IES file stores candela values for both relative and absolute intensities[25]. To retrieve candela values from an EULUMDAT file for both absolute and relative intensities, one needs to do the following conversion:

$$\text{luminous intensity } (cd) = \tfrac{\text{candela per total kilolumens } \left(\tfrac{cd}{klm}\right) \text{ total luminous flux } (lm)}{1000}$$

The EULUMDAT format is able to store more complex fixtures and additional information such as color temperature. Without an organization to maintain the EULUMDAT format, the specification has remained largely unchanged since 1990. There is no official file format publication, and one of the only specifications is available through the Helios32 website [Con14]. A good set of tools, including visualizer and documentation, is available on the Eulumdat Tools website [Sys]

In *Frostbite*, we chose to support only the IES format despite its drawbacks due to a lack of development time. It also seems widespread in computer graphics. A photometric profile, created from IES files, can be directly applied on a point or a spot light. The IES profile can be used for describing the light intensity and can be adjusted with a multiplier. This is the only way to control lights with luminous intensity. A second option is to use an IES profile as a mask, normalized by the maximum intensity of the profile. To handle both cases with the luminous intensity point light equation, we normalize the profile by its maximum intensity and then perform the point light evaluation as follow:

$$L = f(\mathbf{v}, \mathbf{l}) \, \frac{I}{d^2} \, \langle \mathbf{n} \cdot \mathbf{l} \rangle \, \text{getIESProfileAttenuation}() \tag{27}$$

---

[25]If you use PhotometricViewer to visualize your IES, it converts candela values to candela per total kilo-lumens.

I represents either the maximum intensity of the profile, or the user defined light intensity for the mask case. The getIESProfileAttenuation function returns the normalized profile value. For spot lights, the angle attenuation is applied on top of this equation. The tighter shape allows more optimization and is convenient for simple profiles, see Figure 27[26].
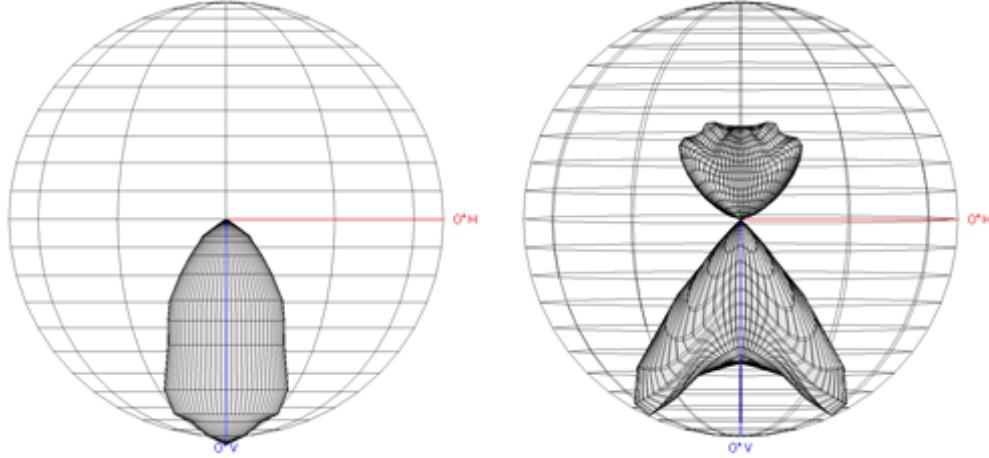

Figure 27: 3D views of IES light profiles. Left: a simple profile suitable for a spot light. Right: a complex light profile requiring a point light.

When creating a new light profile, the spherical photometric function is reconstructed and sampled[27] to fill a 2D texture with a spherical parametrization $(\theta, \cos(\phi))$. We store normalized values scaled by the inverse of the maximum intensity in order to handle both masked and unmasked usage. In shaders, the 2D texture is evaluated and applied as an attenuation Listing 5.

```
float getIESProfileAttenuation(float3 L, ShadowLightInfo light)
{
    // Sample direction into light space
    float3 iesSampleDirection = mul(light.worldToLight, -L);

    // Cartesian to spherical
    // Texture encoded with cos(phi), scale from -1->1 to 0->1
    float phiCoord = (iesSampleDirection.z * 0.5f) + 0.5f;
    float theta = atan2(iesSampleDirection.y, iesSampleDirection.x);
    float thetaCoord = theta * FB_INV_TWO_PI;
    float3 texCoord = float3(thetaCoord, phiCoord, light.lightIndex);
    float iesProfileScale = iesTexture.SampleLevel(sampler, texCoord, 0).r;

    return iesProfileScale;
}

...

att *= getAngleAtt(L, lightForward, lightAngleScale, lightAngleOffset);
att *= getIESProfileAttenuation(L, light);
// lightColor depends on option.
// Non masked: lightColor = color * MaxCandelas
// Masked (for point light with luminous power): lightColor = color * phi / (4 * PI)
float3 luminance = BSDF(...) * saturate(dot(N, L)) * lightColor * att;
```

Listing 5: Sample IES profile.

---

[26]The 3D view is obtained with IES_View available on the Helios32 website.

[27]It is recommended to interpolate the horizontal angles using a cubic spline curve (open or closed depending on whether or not the full 360° range of horizontal angles is specified) [Ash14].

Why use an IES profile? IES profiles are more useful for architectural design than in games. However, being able to use light profiles as a mask brings interesting use cases. For instance, IES profiles can be created by artists with certain tools [Gen] and can be used for simulating complex shadows Figure 28. This is similar to cookie textures but with a different parametrization.

Figure 28: An example of an artist created IES profile to fake shadows. The IES profile was created with IESGen3.

4.6 Sun

The sun is an important light source, especially for outdoor environment. It has very high values, see Table 11, whilst covering a very small solid angle[28]. This configuration makes it very sensitive to normal orientation and variation. Considering such a source as a punctual light for the diffuse part of a material is an acceptable approximation, but doing so for the specular part leads to issues for mirror-like surfaces. In *Frostbite*, in order to partially alleviate these issues, the sun is handled as a disc area light always perpendicular to the outer hemisphere.

Artists specify the sun illuminance (in lux) for a surface perpendicular to the sun direction. This is convenient as they can directly check values against the real world with a light meter (like in Table 11)[29]. The calculation is also simplified:

$$L_{out} = f(\mathbf{v}, \mathbf{l}) \ E = f(\mathbf{v}, \mathbf{l}) \ E_\perp \ \langle \mathbf{n} \cdot \mathbf{l} \rangle \tag{28}$$

where $E_\perp$ is the illuminance value provided by artists. Listing 6 shows a function for evaluating the sun. The diffuse part is lit as if the sun was punctual, whilst the specular part takes the shape of the sun into account.

```
1    float3 D     = sunDirection;
2    float  r     = sin(sunAngularRadius);    // Disk radius
3    float  d     = cos(sunAngularRadius);    // Distance to disk
4
5    // Closest point to a disk (since the radius is small, this is
6    // a good approximation
7    float DdotR = dot(D, R);
8    float3 S     = R - DdotR * D;
9    float3 L     = DdotR < d ? normalize(d * D + normalize(S) * r) : R;
10
```

---

[28]The sun has an angular diameter between $0.526°$ and $0.545°$ (As the orbit of the earth vary with time) [Wika], this mean a solid angle between 0.000066 and 0.000071 as seen from Earth. To calculate sun solid angle use cone solid angle formulae $2\pi(1 - \cos(0.5 * AD * \frac{\pi}{180}))$.

[29]Theorical luminance of the sun is around $1.6 * 10^9 cd/m2$. So lux due to the Sun at Earth surface in direction perpendicular to the ground without considering participating media should be in the range 105 000 and 114 000 lux. Formulae is lux = luminance * solid angle.

Time of day	9 am	10 am	11 am	12 am	1 pm	2:30 pm	5 pm	6:30 pm
Sky + Sun	85 500	88 000	101 600	110 000	113 600	109 300	77 000	39 800
Sky	25 700	19 700	20 100	19 300	19 600	25 500	29 000	14 000
Sun	59 800	68 300	81 500	90 700	94 000	83 800	48 000	25 800
Sky + Sun ⊥					150 400	145 400	142 500	99 000
Sky ⊥					27 300	29 000	36 000	20 000
Sun ⊥					123 100	116 400	106 500	79 000
Sensor's view								

Table 11: Examples of sun and sky illuminance values (in lux) measured with a light meter. Measurements were taken at ground level with the light meter's sensor angled horizontally or perpendicular to the sun (⊥ index). Measurements were performed at various hours of a sunny day with few clouds, in Stockholm in July. The sky values were obtained by occluding the sun with a small object (i.e. casting a shadow on the sensor). The sun value is deduced by subtracting the sky measurement from the combined sun and sky value. The variation of the sky value is due to moving sun and clouds; for the perpendicular case it includes surrounding buildings. The sun value varies because of the orientation of the sensor (cosine law) for the non-oriented measurement and because of sunset for the perpendicular measurement. The perpendicular measurement of the sun is the $E_\perp$ sun value, as described in this section.

```
11      // Diffuse and specular evaluation
12      float illuminance = sunIlluminanceInLux * saturate(dot(N, D));
13
14      // D: Diffuse direction use for diffuse lighting
15      // L: Specular direction use with specular lighting
16      luminance = BSDF(V, D, L, data) * illuminance;
```

Listing 6: Sun evaluation as a disk area light.

Remark: We have no energy conserving factor for the disk area light, as we have not found a correct one currently. The lighting from the sky is handled with a distant light probe, see Section 4.9.

4.7 Area lights

Area lights are important in physically based rendering, for matching real world lighting since all real world sources have a physical shape. This shape allows artists to easily grasp the material properties of a surface, in particular the roughness. Previously, with punctual light, artists tended to hide the infinitesimal specular highlight by modifying the surface roughness and thus coupling lighting and material. Punctual sources can be a good approximation of light sources, but it depends on the context (distance, participating media, etc.). As we will see later, area lights ease the interaction with IBLs as they are conceptually closer. More importantly, area lights provide softer lighting, reducing specular aliasing.

In *Frostbite*, we support different shapes of area lights: sphere, disk, tube, and rectangle. The outgoing luminance of a point illuminated by an area light is given by:
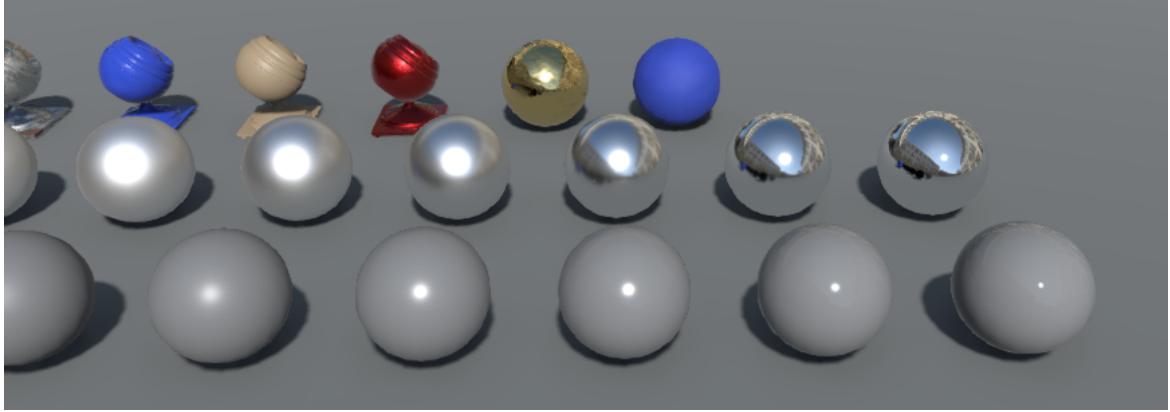
Figure 29: Example of materials with different roughnesses reacting to the sun represented as an oriented disk area light.

$$L_{out} = \int_{\Omega^+} f(\mathbf{v}, \mathbf{l})\; V(\mathbf{l})\; L_{in} \langle \mathbf{n} \cdot \mathbf{l} \rangle\; \mathrm{d}\mathbf{l} = \int_{\Omega_{\text{light}}} f(\mathbf{v}, \mathbf{l})\; L_{in} \langle \mathbf{n} \cdot \mathbf{l} \rangle\; \mathrm{d}\mathbf{l} \tag{29}$$

The function V is 1 if the area light is reachable from the shaded point, 0 otherwise. The function V allows us to take into account both the area light's shape and its shadow. In this section we will not consider area shadow, this will be discussed in Section 4.10.4. Thus V represents only the visibility of the light's shape and Ω_{light} is the solid angle subtended by the area light. This integral over the solid angle can be rewritten as an integral over the area [PH10], see Figure 30:
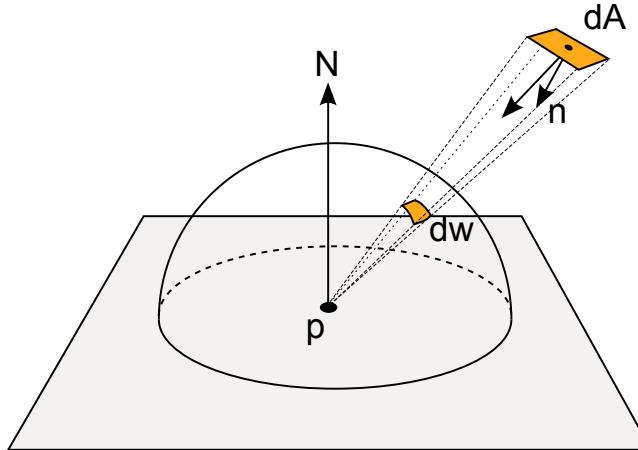

Figure 30: The differential solid angle subtended by a differential area $\mathrm{d}A$ is equal to $\mathrm{d}A\frac{\cos\theta}{r^2}$, where θ is the angle between $\mathrm{d}A$'s surface normal and the vector to the point p, and r is the distance from p to dA.

$$L_{out} = \int_A f(\mathbf{v}, \mathbf{l})\; L_{in}\; \frac{\langle \mathbf{n} \cdot \mathbf{l} \rangle \langle n_a \cdot -l \rangle}{distance^2}\; \mathrm{d}A \tag{30}$$

This equation does not always have an analytical solution, but can be numerically integrated with Monte Carlo and importance sampling. We have implemented an in-engine reference which importance samples all our area lights as discussed in Section 2. The code is provided in Appendix A. Naturally, this kind of computation is too expensive for production and we have developed approximations for them. The in-engine reference mode has been an invaluable tool for checking the accuracy of our approximations.

The diffuse lighting and specular lighting integrals will be discussed separately for each of the area light types:

$$L_{out} = \int_{\Omega_{\text{light}}} f_d(\mathbf{v}, \mathbf{l}) \ L_{in} \ \langle \mathbf{n} \cdot \mathbf{l} \rangle \ d\mathbf{l} = \int_A f_d(\mathbf{v}, \mathbf{l}) \ L_{in} \ \frac{\langle \mathbf{n} \cdot \mathbf{l} \rangle \langle \mathbf{n}_a \cdot -\mathbf{l} \rangle}{distance^2} \ dA \tag{31}$$

$$L_{out} = \int_{\Omega_{\text{light}}} f_r(\mathbf{v}, \mathbf{l}) \ L_{in} \ \langle \mathbf{n} \cdot \mathbf{l} \rangle \ d\mathbf{l} = \int_A f_r(\mathbf{v}, \mathbf{l}) \ L_{in} \ \frac{\langle \mathbf{n} \cdot \mathbf{l} \rangle \langle \mathbf{n}_a \cdot -\mathbf{l} \rangle}{distance^2} \ dA \tag{32}$$

Remark: The area lights presented in this document have not been used in production at the time of writing. They aim to target games at 30fps as well as cinematics. Thus we will only provide non-optimized versions.

4.7.1 Area light unit

With physical light units, the size of area lights and their intensity are connected to each other. Artists can choose between two intensity units:

- **Luminous power**: Luminous power, specified in lumens (lm), describes the total amount of visible light emitted by a light in all directions. This amount is independent of the light size. Increasing the size of a light will not change the illumination level of a scene. However, highlights produced by such a light will get dimmer as its area increases since the same power will be distributed over a larger area. See the top row of Figure 31.
- **Luminance**: Luminance, specified in nits (cd/m^2) or in exposure value (EV), describes the surface power of visible light. When this unit is used, the total energy emitted by the light will depend on its size. The illumination level in the scene will increase as the area increases, but the highlights produced by such a light will conserve their intensity. See the bottom of Figure 31.

In practice artists rarely use luminance as a setting for area lights. It is mostly appropriate for flat diffuse surfaces which emit light evenly over an entire surface, such as a display.

For convenience, we systematically convert intensity into luminance for lighting calculations. The luminance due to a point on a Lambertian emitter[30], emitted in any direction, is equal to its total luminous power ϕ divided by the emitter area A and the projected solid angle:

$$L = \frac{\phi}{A \ \int_{\Omega^+} \langle \mathbf{l} \cdot \mathbf{n} \rangle d\mathbf{l}} = \frac{\phi}{A \ \pi} \tag{33}$$

The formulae for converting from luminous power to luminance are provided in Table 12. Conversion from EV to luminance has been detailed in Section 4.3.

It is now more clear why punctual lights do not have a luminance unit. When calculating the lighting for an area light, we effectively perform an integration over the light area and deal with luminance. But punctual lights do not have an area and thus rely on luminous intensity. However in *Frostbite*, punctual and area lights share the same interface. In order to support this common interface, we chose to automatically promote punctual lights to small area lights of 1 cm if the artists specify a luminance unit to control the intensity. This small hack is transparent for the artists and is convenient for smoothing the transition between punctual and area lights. The luminance unit is rarely used and thus this hack does not hurt performance. The line and frustum lights which should have been considered as punctual lights are also considered as area lights due to their complexity, so they are no different from tube or rectangular lights respectively.

---

[30]A *Lambertian emitter* is a light source whose radiance distribution follows a cosine distribution.
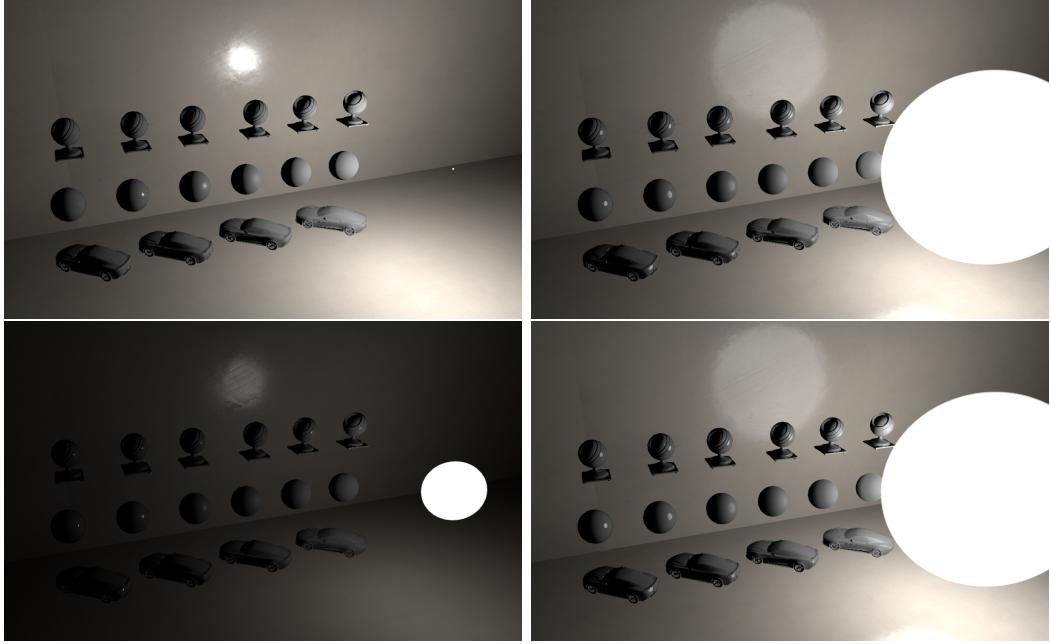
Figure 31: Top: from left to right, the sphere radii increase, but the illumination level of the scene stays constant whilst the specular highlights get bigger and dimmer. Bottom: from left to right, the sphere radii increase, as does the overall diffuse intensity, but the specular highlights conserve their intensity whilst still getting bigger.

Sphere	$\frac{\phi}{4\ radius^2\ \pi^2}$
Disk	$\frac{\phi}{radius^2\ \pi^2}$
Tube	$\frac{\phi}{(2\pi\ radius\ width\ +\ 4\pi\ radius^2)\ \pi}$
Rectangle	$\frac{\phi}{width\ height\ \pi}$

Table 12: Luminous power (lm) to Luminance ($cd.m^{-2}$) conversion formulae for *Frostbite* lights. Example: a sphere area light of radius 15 cm and emitting 1500 lm will have a luminance of 1688 $cd.m^{-2}$.

4.7.2   Diffuse area lights

4.7.2.1   General

Initially, we will solve area lights integration for a Lambertian diffuse BRDF $\frac{\rho}{\pi}$, before looking at Disney's diffuse BRDF case. We will also assume that area lights have a uniform and constant intensity L_{in}. Thus, the diffuse lighting integral from Equation 31 can be re-written as:

$$L_{out} = \frac{\rho}{\pi} \int_{\Omega_{light}} L_{in}\ \langle \mathbf{n} \cdot \mathbf{l} \rangle = \frac{\rho}{\pi}\ E(n) \qquad (34)$$

Where the illuminance E is defined as:

$$E(n) = \int_{\Omega_{light}} L_{in}\ \langle \mathbf{n} \cdot \mathbf{l} \rangle\ \mathrm{d}\mathbf{l} = \int_{A} L_{in}\ \frac{\langle \mathbf{n} \cdot \mathbf{l} \rangle \langle \mathbf{n}_a \cdot -\mathbf{l} \rangle}{distance^2}\ \mathrm{d}A \qquad (35)$$

The inverse square law for computing the illuminance, as seen in Section 4.4 with Equation 13, is not valid for an area light unless this area light is sufficiently far away from the receiver surface, see

Section 4.7.3. Thus to calculate the illuminance of an area light, we need to solve Equation 35[31].
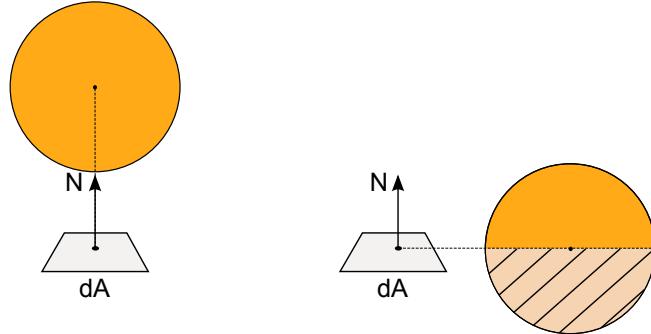

Figure 32: Two light configuration. Left: the light is entirely visible from the patch dA. Right: the light is partially visible from the patch dA, where half of the light is below the horizon of the surface.

Area lights can span a large solid angle and even go partially below the horizon of a shaded point. In this case, area light lighting wraps around objects, resulting in a softer look. Without correctly handling this "horizon case", the wrapped lighting intensity will be incorrect and it will no longer look like it is lit by an area light, see Figure 32 and 33. There are a few different methods to solve Equation 35. Some of them are only correct for area lights subtending a small solid angle from the shaded point, which do not cross the horizon. Some others handle the horizon case but with an incorrect intensity. In the following sections, we will refer to a method as "*correctly handling the horizon*" if it solves the case of a large area light with correct intensity.

Analytic integration: The integral is complex but in some constrained cases, there exists an analytic solution. Fortunately solutions for miscellaneous configurations and light types are available in two other scientific fields: *light transport* and *heat transfer*. In light transport, the well known *form factor* between two differential areas, P_i and P_j, used in radiosity systems, is defined as:

$$\text{FormFactor} = \int_{x \epsilon P_i} \int_{y \epsilon P_j} \frac{\cos \theta \cos \theta'}{\pi r^2} \ V(x, y) \ \mathrm{d}x\mathrm{d}y \tag{36}$$

This equation is a general form of Equation 35. It handles a surface shaded by another surface while Equation 35 handles a point shaded by a surface. Various solutions are provided for radiosity systems for points shaded by surfaces. Note the presence of the divide by π inside the equation which comes from the Lambert BRDF. This is important because it means we will need to multiply the analytic solution based on the form factor formulation by π to cancel it, as in our case the definition of the illuminance does not contain the $\frac{1}{\pi}$ term:

$$E(n) = \int_{\Omega_{\text{light}}} L_{in} \ \langle \mathbf{n} \cdot \mathbf{l} \rangle \ \mathrm{d}\mathbf{l} = L_{in} \ \pi \ \text{FormFactor} \tag{37}$$

In radiative transfer, there is an equivalent formulation called *view factor*. A view factor, noted $F_{a \rightarrow b}$, is the proportion of radiation which leaves a surface A and strikes a surface B. The heat transfer field provides a large set of analytic solutions for points receiving radiation from various surface shapes [HSM10a]. For simplicity, we will refer to both *form factor* and *view factor* as **form factor** even if they do not have the same semantic since they share the same formulation. Form factor can have formulation with or without correct horizon handling. We have chosen to adopt the form factor

---

[31]The calculation of illuminance is in many ways similar to ambient occlusion computation. See Section 4.10.1. Thus the results of this section can be reused for the ambient occlusion volume algorithm [Hil10; Iwa13].
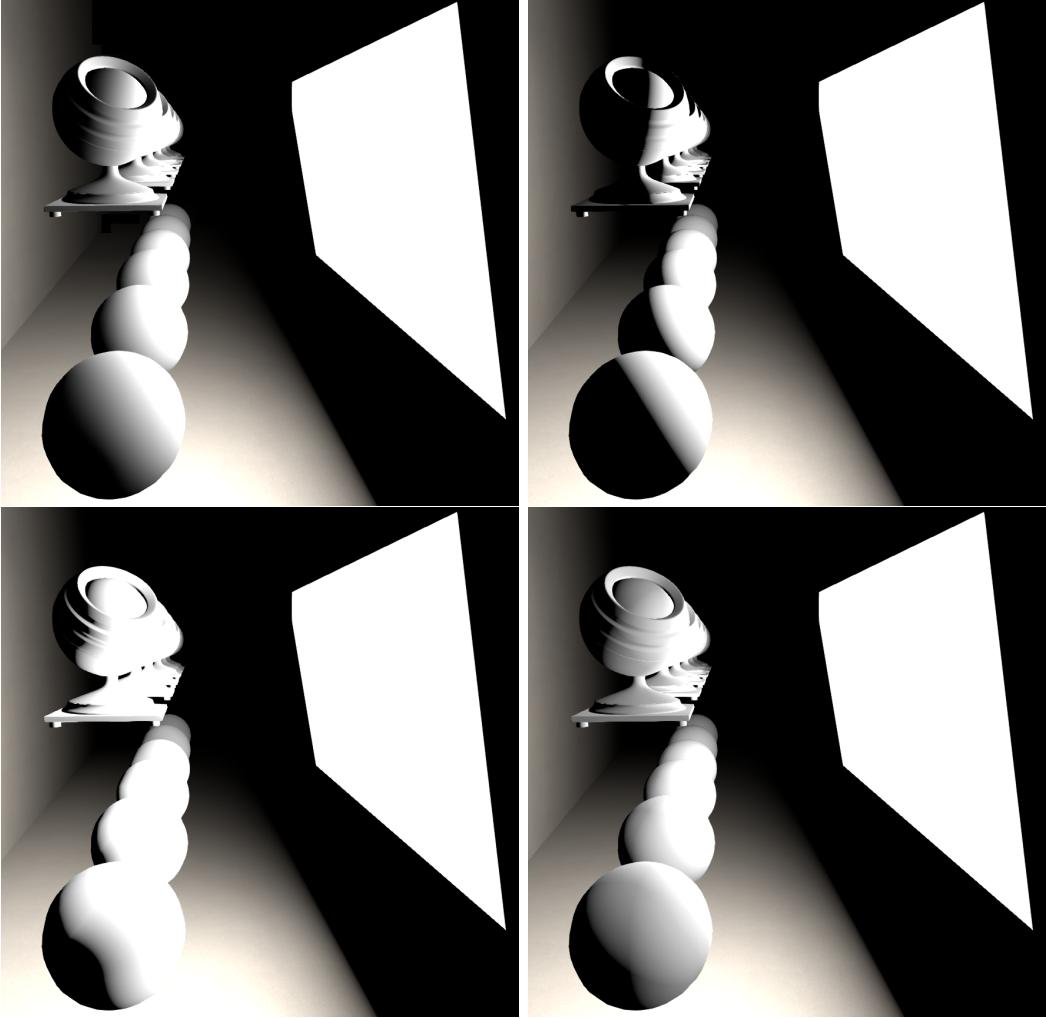
Figure 33: Large rectangular light lighting. Top left: Reference. Top right: Method without horizon handling. The lighting is correct in a small cone oriented toward the area light's center. Lighting is constrained inside an hemisphere defined by the light position and the shaded point. Bottom left: Method with horizon handling but wrong intensity of the wrap lighting. Bottom right: Method with correct horizon handling. The look is softer and there is a wrap lighting with a smooth transition.

method for sphere and disk area lights. For completeness we also provide one for rectangular light in Appendix E, but we have not adopted it. Most form factor expressions are complex. We chose not to reproduce the equation here and only provide the corresponding code listing. We refer the reader to the provided references for more details.

4.7.2.2 Sphere area lights

- Quilez [Quí06] provides a solution which does not handle the horizon case. It is similar to the "Patch to a sphere frontal" configuration, provided in [Mar14]. The Quilez version has an extra $\langle \mathbf{n} \cdot \mathbf{l} \rangle$ to allow handling of the case where the area sphere light center is not aligned with the surface normal. However being above horizon is not enough, the formulae only works for small subtended solid angles above the horizon. See Figure 36.
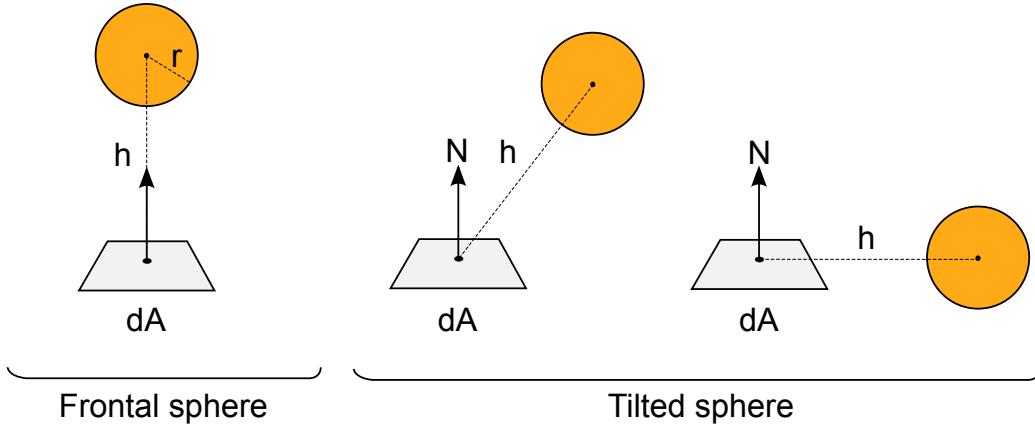
Figure 34: Sphere area light. Left: "Patch to a frontal sphere", i.e. case where the light source is above the horizon and is aligned with the surface normal of the patch dA. Right: "Patch to a tilted sphere ", i.e. general case where the light source can be below the horizon of the patch dA.

- The configuration "Patch to a sphere tilted" provided in [Mar14] correctly handles the horizon case. Another formulation is provide by Snyder in [Sny96][32].

The "patch to a sphere frontal" and the "patch to a sphere tilted" are represented in Figure 34. Equations of Quilez and "patch to a sphere tilted" are reported in Listing 7 and results are shown in Figure 35.
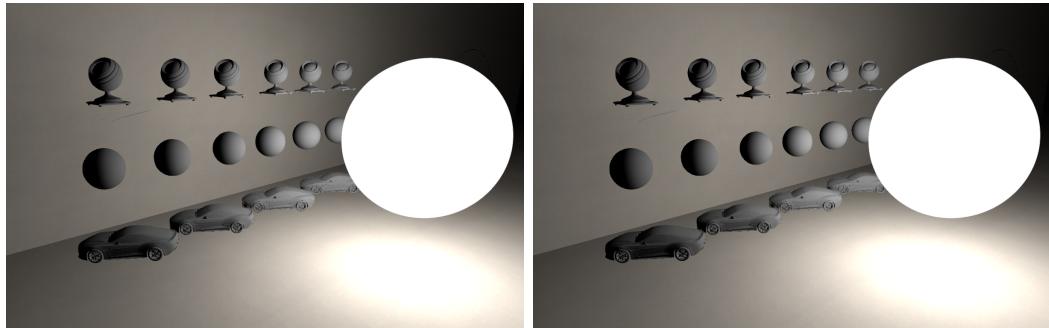

Figure 35: Left: Result of a sphere area light with correct horizon handling. Right: Reference. It matches perfectly.

```
1    float3 Lunormalized = lightPos - worldPos;
2    float3 L = normalize(Lunormalized);
3    float sqrDist = dot(Lunormalized, Lunormalized);
4
5  #if WITHOUT_CORRECT_HORIZON // Analytical solution above horizon
6
7    // Patch to Sphere frontal equation (Quilez version)
8    float sqrLightRadius = light.radius * light.radius;
9    // Do not allow object to penetrate the light (max)
10   // Form factor equation include a (1 / FB_PI) that need to be cancel
11   // thus the "FB_PI *"
12   float illuminance = FB_PI *
13       (sqrLightRadius / (max(sqrLightRadius, sqrDist))) * saturate(dot(worldNormal, L));
```

---

[32]It can be shown by using trigonometric identities that Snyder's formulae is equivalent to the "Patch to a sphere tilted" formulae. Thus the cubic hermite approximation provide by Snyder applies to both formulae. We have chosen to only present unoptimized formulae in this document, and the "Patch to a sphere tilted" formulae is simpler and is reused for the Disk area light.

```
14
15  #else // Analytical solution with horizon
16
17      // Tilted patch to sphere equation
18      float Beta = acos(dot(worldNormal, L));
19      float H = sqrt(sqrDist);
20      float h = H / radius;
21      float x = sqrt(h * h - 1);
22      float y = -x * (1 / tan(Beta));
23
24      float illuminance = 0;
25      if (h * cos(Beta) > 1)
26          illuminance = cos(Beta) / (h * h);
27      else
28      {
29          illuminance = (1 / (FB_PI * h * h)) *
30              (cos(Beta) * acos(y) - x * sin(Beta) * sqrt(1 - y * y)) +
31              (1 / FB_PI) * atan(sin(Beta) * sqrt(1 - y * y) / x);
32      }
33
34      illuminance *= FB_PI;
35
36  #endif
```

Listing 7: Sphere area light illuminance.
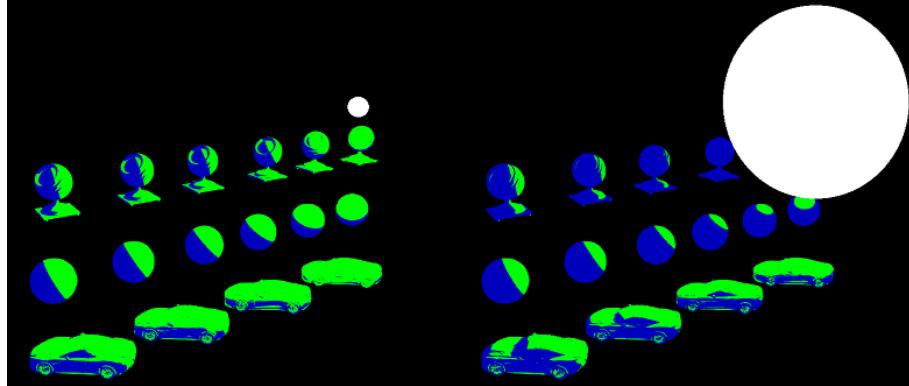
Figure 36: Sphere area lighting. The green color highlights where the commonly used formula without horizon handling $\frac{radius^2}{distance^2} \langle \mathbf{n} \cdot \mathbf{l} \rangle$ is correct. Left: With small area light, most of the lighting is correct. Right:With large area light the area of accurate result decreases. Thus we can see that correct values are restricted to shaded points where light does not cross the horizon and the subtended solid angle is small.

Remark: Section 4.7.1 claims that using luminous power units makes the illumination level independent of the light area. By using the formulation without horizon handling (without loss of generality), the illuminance for a sphere area light with luminous power ϕ is given by:

$$E = L_{in} \ \pi \ \text{FormFactor} = \frac{\phi}{4\pi^2 r^2} \ \pi \ \frac{r^2}{d^2} \langle \mathbf{n} \cdot \mathbf{l} \rangle = \frac{\phi}{4\pi d^2} \ \langle \mathbf{n} \cdot \mathbf{l} \rangle \tag{38}$$

This equation matches a point light illuminance Equation 19 (which consider $\langle \mathbf{n} \cdot \mathbf{l} \rangle = 1$) and is independent of the sphere's surface area. It also matches the real world measurements of Section 4.4.

4.7.2.3 Disk area lights

44
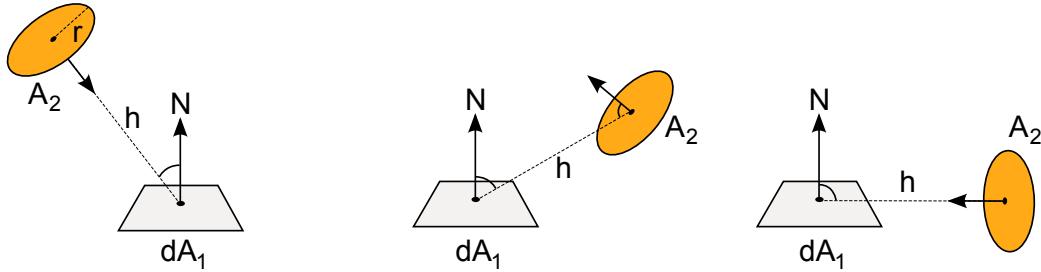
Figure 37: Disk area light. Left: simple case, tilted patch dA to a non oriented disk (facing disk). Middle and right: general case where the disk and the patch dA are randomly oriented and can be below the horizon.

- Coombe [CH05] proposes a solution without correct horizon handling for an oriented disk. Like for the sphere, being above horizon is not enough, the formulae only works for small subtended solid angles above the horizon and with constraint orientation.

- The configuration [HSM10b] of the radiative transfer catalogue [HSM10a] correctly handles the horizon case but it is limited to a tilted plane and a non-oriented disk, see Figure 37. To take account of the disk orientation, we multiply the formula by an extra $\langle \mathbf{n}_{\text{light}} \cdot -\mathbf{l} \rangle$, with $\mathbf{n}_{\text{light}}$ the light plane normal. This addition allows matching of the reference above the horizon within a constrained orientation and with small subtended solid angle, but has a small discrepancy in other case. Results are good enough for our purposes.

Equations of these two cases are reported in Listing 8 and results are shown in Figure 38
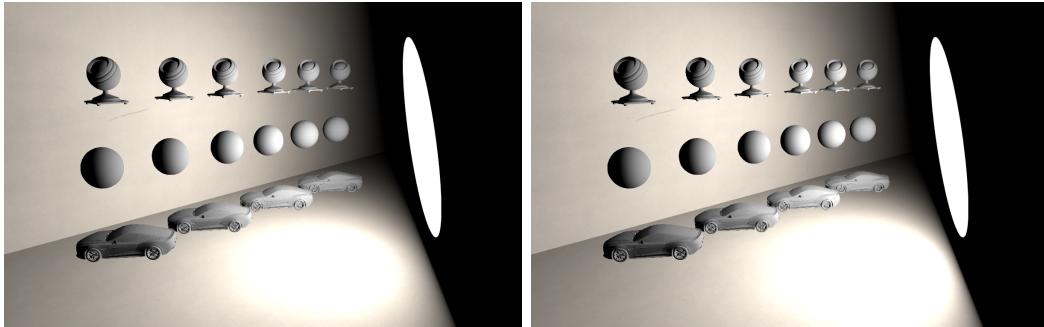
Figure 38: Left: Result of a disk area light with correct horizon handling. Right: Reference.

```
1  float cot(float x)  { return cos(x) / sin(x); }
2  float acot(float x) { return atan(1 / x); }
3
4  #if WITHOUT_CORRECT_HORIZON // Analytical solution above horizon
5
6      // Form factor equation include a (1 / FB_PI) that need to be cancel
7      // thus the "FB_PI *"
8      float illuminance = FB_PI * saturate(dot(planeNormal, -L)) *
9                          saturate(dot(worldNormal, L)) /
10                         (sqrDist / (radius * radius) + 1);
11
12 #else // Analytical solution with horizon
13
14     // Nearly exact solution with horizon
15     float h = length(lightPos - worldPos);
16     float r = lightRadius;
17     float theta = acos(dot(worldNormal, L));
```

45

```
18      float H = h / r;
19      float H2 = H * H;
20      float X = pow((1 - H2 * cot(theta) * cot(theta)), 0.5);
21
22      float illuminance = 0;
23      if (theta < acot(1 / H))
24      {
25          illuminance = (1 / (1 + H2)) * cos(theta);
26      }
27      else
28      {
29          illuminance = -H * X * sin(theta) / (FB_PI * (1 + H2)) +
30              (1 / FB_PI) * atan(X * sin(theta) / H) +
31              cos(theta) * (FB_PI - acos(H * cot(theta))) / (FB_PI * (1 + H2));
32      }
33
34      // Multiply by saturate(dot(planeNormal, -L)) to better match ground
35      // truth. Matches well with the first part of the equation but there
36      // is a discrepancy with the second part. Still an improvement and it is
37      // good enough.
38      illuminance *= FB_PI * saturate(dot(planeNormal, -L));
39
40  #endif
```

Listing 8: Disk area light illuminance.

Remark: As with the sphere and point lights illuminance comparison, we compare the disk to our *Frostbite* spot light illuminance equations with a constant luminous power. By using the formulation without horizon handling (without loss of generality), the illuminance for a disk area light with luminous power ϕ is given by

$$E = L\,\pi\,\text{FormFactor} = \frac{\phi}{\pi^2 radius^2}\,\pi\,\frac{radius^2}{distance^2 + radius^2}\,\langle \mathbf{n}\cdot\mathbf{l}\rangle = \frac{\phi}{\pi\,distance^2 + \pi\,radius^2}\,\langle \mathbf{n}\cdot\mathbf{l}\rangle \quad (39)$$

The conversion chosen for luminous power for the spotlight in *Frostbite* aims to mimic the property of the luminous power unit. This equation and Equation 21 are close (with $\langle \mathbf{n} \cdot \mathbf{l}\rangle = 1$) and allows us to smoothly fade from one to the other (as the radius decreases, the extra term disappears) see Figure 39. So the statement that the illumination level is independent of the area with luminous power is not totally correct with our luminous power approximation, but we found it to work well enough.
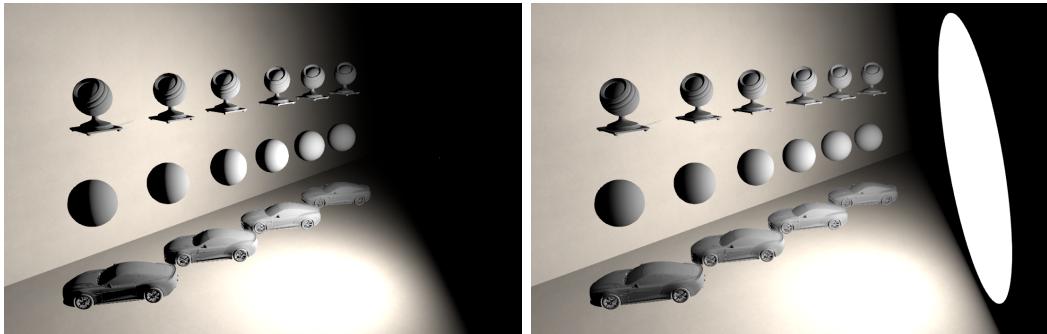
Figure 39: Comparison of the illuminance of a spot light (left) and a disk light (right) with a large radius. Note that the overall illumination levels stay constant and that a small region on the objects close to the disk (like the spheres) are dimmer with a large disk.

A disk area light is similar to a spot light in *Frostbite* and supports an angular attenuation. This angular attenuation is simply applied on the illuminance without any realistic consideration to make

the transition with the spot light smoother. The angular attenuation is obtained with the same equation as for the spot light by faking the position of the light similar to the way we calculate the shadow for area lights, see Listing 9 and Section 4.10.4.

```
1
2  // On the CPU
3  float3 virtualPos = lightPos + lightForward * (discRadius / tan(halfOuterAngle))
4
5  // On the GPU
6  // Attenuate with a virtual position which is the light position shifted
7  // in opposite light direction by an amount based on the outer angle.
8  illuminance *= getAngleAtt(normalize(virtualPos - worldPos), lightForward,
9                             lightAngleScale, lightAngleOffset);
```

Listing 9: Disk area light illuminance with angle attenuation.

4.7.2.4 Sphere and disk area lights merging

After playing with trigonometric identities it can be shown that both sphere and disk area lights with correct horizon handling formulae presented in previous sections are similar. They only differ by their subtended angle and thus can share their evaluation code. The Listing 10 contains some details. The expensive inverse trigonometric functions can be approximated efficiently on the AMD GCN architecture [Dro14c].

```
1
2  // A right disk is a disk oriented to always face the lit surface.
3  // Solid angle of a sphere or a right disk is 2 PI (1 - cos(subtended angle)).
4  // Subtended angle sigma = arcsin(r / d) for a sphere
5  // and sigma = atan(r / d) for a right disk
6  // sinSigmaSqr = sin(subtended angle)^2, it is (r^2 / d^2) for a sphere
7  // and (r^2 / ( r^2 + d^2)) for a disk
8  // cosTheta is not clamped
9  float illuminanceSphereOrDisk(float cosTheta, float sinSigmaSqr)
10 {
11     float sinTheta = sqrt(1.0f - cosTheta * cosTheta);
12
13     float illuminance = 0.0f;
14     // Note: Following test is equivalent to the original formula.
15     // There is 3 phase in the curve: cosTheta > sqrt(sinSigmaSqr),
16     // cosTheta > -sqrt(sinSigmaSqr) and else it is 0
17     // The two outer case can be merge into a cosTheta * cosTheta > sinSigmaSqr
18     // and using saturate(cosTheta) instead.
19     if (cosTheta * cosTheta > sinSigmaSqr)
20     {
21         illuminance = FB_PI * sinSigmaSqr * saturate(cosTheta);
22     }
23     else
24     {
25         float x = sqrt(1.0f / sinSigmaSqr - 1.0f); // For a disk this simplify to x = d / r
26         float y = -x * (cosTheta / sinTheta);
27         float sinThetaSqrtY = sinTheta * sqrt(1.0f - y * y);
28         illuminance = (cosTheta * acos(y) - x * sinThetaSqrtY) * sinSigmaSqr + atan(
29             sinThetaSqrtY / x);
30     }
31
32     return  max(illuminance, 0.0f);
33 }
34
35 // Sphere evaluation
36 float cosTheta = clamp(dot(worldNormal, L), -0.999, 0.999); // Clamp to avoid edge case
37 // We need to prevent the object penetrating into the surface
38 // and we must avoid divide by 0, thus the 0.9999f
```

```
38  float sqrLightRadius = lightRadius * lightRadius;
39  float sinSigmaSqr = min(sqrLightRadius / sqrDist, 0.9999f);
40  float illuminance = illuminanceSphereOrDisk(cosTheta, sinSigmaSqr);
41
42
43  // Disk evaluation
44  float cosTheta = dot(worldNormal, L);
45  float sqrLightRadius = lightRadius * lightRadius;
46  // Do not let the surface penetrate the light
47  float sinSigmaSqr = sqrLightRadius / (sqrLightRadius + max(sqrLightRadius, sqrDist));
48  // Multiply by saturate(dot(planeNormal, -L)) to better match ground truth.
49  float illuminance = illuminanceSphereOrDisk(cosTheta, sinSigmaSqr)
50                    * saturate(dot(planeNormal, -L));
```

Listing 10: Sphere and disk area light illuminance.

4.7.2.5 Rectangular area lights
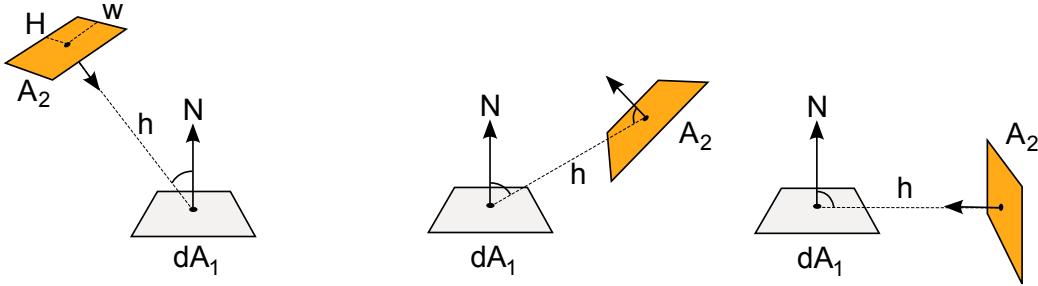
Figure 40: Rectangular area light. Left: simple case, where the rectangle normal is pointing toward the patch dA. Middle and right: general case where the rectangle normal is randomly oriented and can be below the horizon.

We have not found any affordable form factor solution with correct horizon handling for rectangular light. Figure 40 highlights the different configurations. Thus we have looked for an alternate solution. For completeness we provide form factor for rectangle light without horizon handling in Appendix E.

Most representative point: Drobot [Dro14b] proposes to approximate the illuminance integral with *a single representative diffuse point light weighted by the light's solid angle*[33]:

$$E(n) = \int_{\Omega_{\text{light}}} L_{in} \langle \mathbf{n} \cdot \mathbf{l} \rangle \, \mathrm{d}\mathbf{l} \approx \ \Omega_{\text{light}} \ L_{in} \ \langle \mathbf{n} \cdot \mathbf{l} \rangle \tag{40}$$

This approximation is valid for small solid angles. It can be improved with a good choice of $\mathbf{l}$ and thus extended to a larger solid angle. $\mathbf{l}$ refers to the direction going from the shaded point to a point on the area light called the *Most Representative Point* (MRP). The MRP is the point with the highest value when performing the diffuse lighting integral. The method correctly handles the horizon case as the light position will be moved to the MRP. This implies correctly calculating the solid angle by taking into account the horizon and thus having the MRP inside this solid angle. Unfortunately, this is a costly operation. The solid angle can be calculated with the following integral:

$$\Omega_{\text{light}} = \int_{\Omega_+} V(\mathbf{l}) \, \mathrm{d}\mathbf{l} = \int_{\Omega_{\text{light}}} \mathrm{d}\mathbf{l} \tag{41}$$

This formulation looks similar to the illuminance integral without L and $\langle \mathbf{l} \cdot \mathbf{n} \rangle$. It can be solved either numerically or analytically in a similar fashion than form factors. Analytic formulations for solid angle

---

[33]This formulation is not explicit in Drobot's GPU Pro 5 article [Dro14b] but it is the author's intention [Dro14a].

of an oriented rectangle are provided by [UFK13], but without horizon handling. For performance reason, Drobot [Dro14a] proposes to use a right pyramid solid angle [Pla] without horizon handling in place of an oriented rectangle. These solid angle formulae are gathered in Listing 11. Mathar [Mat14] provides the solid angle formulation for an always facing rectangle with horizon handling, but it is too complex and does not handle oriented rectangles.

```
float rightPyramidSolidAngle(float dist, float halfWidth, float halfHeight)
{
    float a = halfWidth;
    float b = halfHeight;
    float h = dist;

    return 4 * asin(a * b / sqrt((a * a + h * h) * (b * b + h * h)));
}


float rectangleSolidAngle(float3 worldPos,
                          float3 p0, float3 p1,
                          float3 p2, float3 p3)
{
    float3 v0 = p0 - worldPos;
    float3 v1 = p1 - worldPos;
    float3 v2 = p2 - worldPos;
    float3 v3 = p3 - worldPos;

    float3 n0 = normalize(cross(v0, v1));
    float3 n1 = normalize(cross(v1, v2));
    float3 n2 = normalize(cross(v2, v3));
    float3 n3 = normalize(cross(v3, v0));

    float g0 = acos(dot(-n0, n1));
    float g1 = acos(dot(-n1, n2));
    float g2 = acos(dot(-n2, n3));
    float g3 = acos(dot(-n3, n0));

    return g0 + g1 + g2 + g3 - 2 * FB_PI;
}
```

Listing 11: Solid angle without horizon handling of a right pyramid and an oriented rectangle.

The quality of the MRP approach depends on the accuracy of the solid angle calculation. Unfortunately, the cost of handling the solid angle horizon is not affordable for our real time constraints. Thus we have looked for an alternative solution. Drobot's MRP approach with improvements and a screenshot comparison are provided in Appendix E.

Structured sampling of the light shape: We know that for any function we can write[34]:

$$\int_\Omega f(x) \ dl = \ \Omega \ \text{Average}[f(x)] \tag{42}$$

Thus with the assumption of a constant L_{in} we can write:

$$E(n) = \int_{\Omega_{\text{light}}} L_{in}\langle \mathbf{n} \cdot \mathbf{l}\rangle dl = \Omega_{\text{light}} \ L_{in}\text{Average}[\langle \mathbf{n} \cdot \mathbf{l}\rangle] \tag{43}$$

Finding the average of $\langle \mathbf{n} \cdot \mathbf{l}\rangle$ over the solid angle is not easy, but we could approximate it with a well chosen set of light points. With real time constraint in mind, we chose to average the aforementioned

---

[34]For instance, the solid angle of a hemisphere is 2π, the average height of the upper hemisphere of a unit sphere is $\frac{1}{2}$. The average height is the average cosine between the normal of the hemisphere and the directions. Thus $\int_{\Omega+} \langle \mathbf{n} \cdot \mathbf{l}\rangle \ dl = 2\pi \ \frac{1}{2} = \pi$.

cosine by selecting a set of N light points for maximizing the light surface coverage. In the case of a rectangular light, we take the four corners and the center:

$$E(n) = \int_{\Omega_{\text{light}}} L_{in}\langle \mathbf{n} \cdot \mathbf{l}\rangle \mathrm{d}\mathbf{l} \approx \ \Omega_{\text{light}} \ L_{in} \ \frac{1}{N} \sum_{i=1}^{N} \max(\langle \mathbf{n} \cdot \mathbf{l}_i\rangle, 0) \tag{44}$$

The benefit of this approach is that by clamping the cosine, we restrict the point to be located in the positive hemisphere define by the surface normal, allowing implicit handling of the solid angle horizon. The MRP approach is based on an importance sampling integration with one sample, the most important one. This one is based on a structured sampling integration relying on knowledge of the light's shape. This has been proven to be more accurate when the integration is restricted to a low number of samples [Dup+13].

We have adopted this method for rectangular area lights, allowing us to have the right intensity for the wrapped lighting without having to calculate the solid angle with the horizon. However it exhibits a barely noticeable banding artifact due to the under sampling of using only 5 samples, shown on Figure 41. The code is provided in Listing 12.

```
1  if (dot(worldPos - lightPos, lightPlaneNormal) > 0)
2  {
3      float halfWidth  = lightWidth * 0.5;
4      float halfHeight = lightHeight * 0.5;
5      float3 p0 = lightPos + lightLeft * -halfWidth + lightUp *  halfHeight;
6      float3 p1 = lightPos + lightLeft * -halfWidth + lightUp * -halfHeight;
7      float3 p2 = lightPos + lightLeft *  halfWidth + lightUp * -halfHeight;
8      float3 p3 = lightPos + lightLeft *  halfWidth + lightUp *  halfHeight;
9      float  solidAngle = rectangleSolidAngle(worldPos, p0, p1, p2, p3);
10
11     float illuminance = solidAngle * 0.2 * (
12                         saturate(dot(normalize(p0 - worldPos), worldNormal) +
13                         saturate(dot(normalize(p1 - worldPos), worldNormal))+
14                         saturate(dot(normalize(p2 - worldPos), worldNormal))+
15                         saturate(dot(normalize(p3 - worldPos), worldNormal))+
16                         saturate(dot(normalize(lightPos - worldPos), worldNormal)));
17 }
```

Listing 12: Illuminance of a rectangular light with average clamped cosine method.

Remark: To develop area light approximations, we have created a framework in Mathematica (provided as a companion file to this document). This framework includes various error estimations for several methods described here and in the Appendix E. From our experience, the MRP approach is less accurate than our structured sampling approach for diffuse area lighting, even by using the true MRP (determined by brute force) and the true solid angle.

4.7.2.6 Tube area lights

In *Frostbite*, our tube area lights are represented as a capsule, i.e a cylinder with two hemispheres at each end, see Figure 42. Finding the form factor or the solid angle for a capsule is a difficult problem which will not fit with our real-time constraints. We chose to approximate the capsule by splitting it into a cylinder and two hemispheres and relying on previous results:

- The illuminance from a cylinder is approximated with a facing rectangular light generated from the cylinder's size
- The illuminance from the two hemispheres are approximated by the form factor of a sphere located at the closest point from the shaded point to the cylinder axis
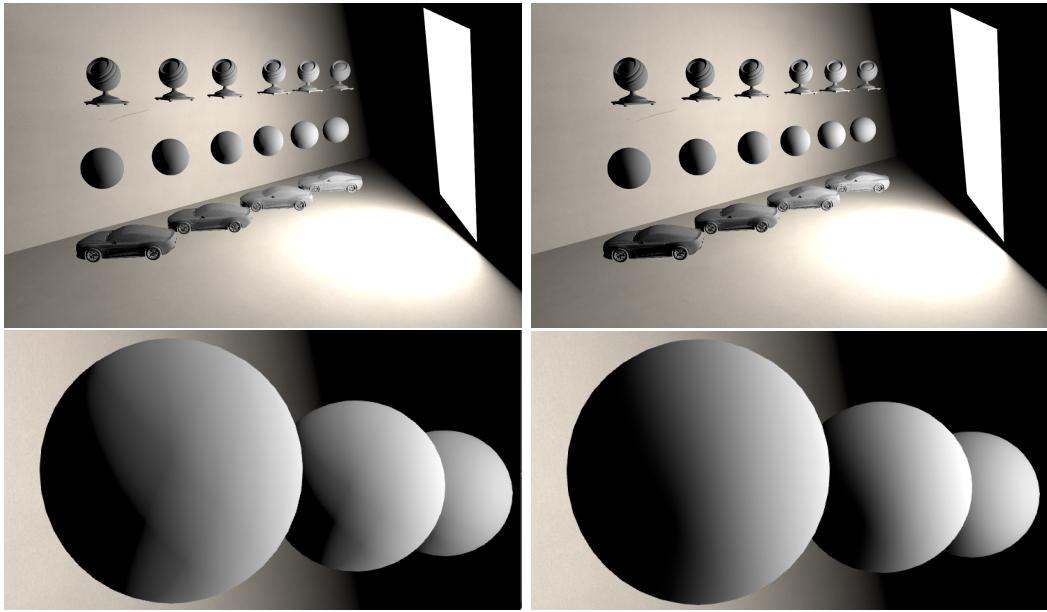
Figure 41: Top left: Result of a rectangular area light with correct horizon handling. Top right: Reference. Bottom left: Closer shot of top left. See how the wrapped lighting has the correct intensity but exhibits barely visible banding. Bottom right: Closer shot of top right.
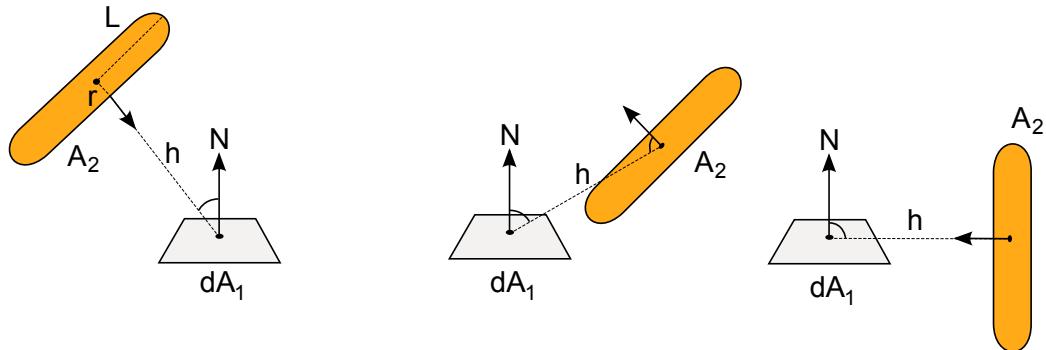
Figure 42: Tube area light. Left: simple case, where the tube normal is pointing toward the patch dA. Middle and right: general cases where the tube normal is randomly oriented and can be below the horizon.

- Both illuminance are added to get the final illuminance, see Listing 13 and Figure 43.

Despite this splitting and approximations, the results are close enough to the ground truth.

```
1  // Return the closest point on the line (without limit)
2  float3 closestPointOnLine(float3 a, float3 b, float3 c)
3  {
4      float3 ab = b - a;
5      float t = dot(c - a, ab) / dot(ab, ab);
6      return a + t * ab;
7  }
8
9  // Return the closest point on the segment (with limit)
10 float3 closestPointOnSegment(float3 a, float3 b, float3 c)
11 {
12     float3 ab = b - a;
13     float t = dot(c - a, ab) / dot(ab, ab);
```
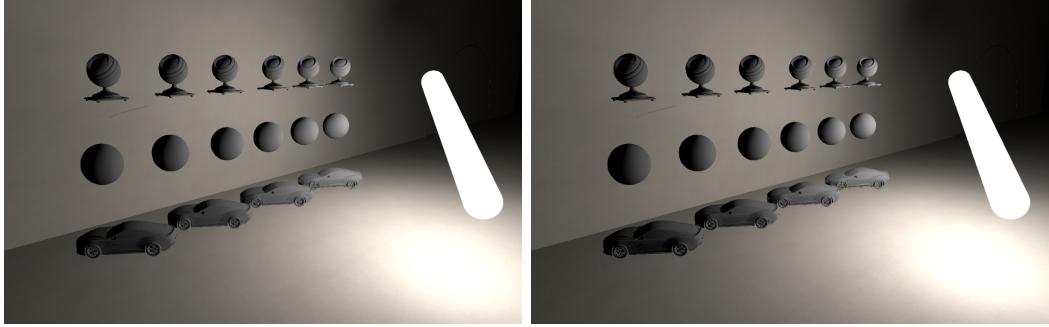
Figure 43: Left: Result of a tube area light with correct horizon handling. Right: Reference.

```
14        return a + saturate(t) * ab;
15 }
16
17 // The sphere is placed at the nearest point on the segment.
18 // The rectangular plane is define by the following orthonormal frame:
19 float3 forward  = normalize(closestPointOnLine(P0, P1, worldPos) - worldPos);
20 float3 left     = lightLeft;
21 float3 up       = cross(lightLeft, forward);
22
23 float3 p0 = lightPos - left * (0.5 * lightWidth) + lightRadius * up;
24 float3 p1 = lightPos - left * (0.5 * lightWidth) - lightRadius * up;
25 float3 p2 = lightPos + left * (0.5 * lightWidth) - lightRadius * up;
26 float3 p3 = lightPos + left * (0.5 * lightWidth) + lightRadius * up;
27
28 float solidAngle = rectangleSolidAngle(worldPos, p0, p1, p2, p3);
29
30 float illuminance = solidAngle * 0.2 * (
31         saturate(dot(normalize(p0 - worldPos), worldNormal)) +
32         saturate(dot(normalize(p1 - worldPos), worldNormal)) +
33         saturate(dot(normalize(p2 - worldPos), worldNormal)) +
34         saturate(dot(normalize(p3 - worldPos), worldNormal)) +
35         saturate(dot(normalize(lightPos - worldPos), worldNormal)));
36
37 // We then add the contribution of the sphere
38 float3 spherePosition    = closestPointOnSegment(P0, P1, worldPos);
39 float3 sphereUnormL      = spherePosition - worldPos;
40 float3 sphereL           = normalize(sphereUnormL);
41 float  sqrSphereDistance = dot(sphereUnormL, sphereUnormL);
42
43 float illuminanceSphere = FB_PI * saturate(dot(sphereL, data.worldNormal)) *
44                           ((lightRadius * lightRadius) / sqrSphereDistance);
45
46 illuminance += illuminanceSphere;
```

Listing 13: Illuminance of a tube.

4.7.3 Five times rule

When an area light is far away from the receiver point, it is possible to approximate its illuminance by the inverse square law. A general rule of thumb for illuminance is *the five times rule* [Rye]: *"the distance to a light source should be greater than five times the largest dimension of the sources"* for the inverse square law to be applicable, see Figure 44. We can use this rule to optimize complex illuminance calculations to the simpler inverse square law based on the properties of the diffuse area light and its distance to the receiver. Listing 14 provide example code for sphere and disk area light in *Frostbite*. This code is not use in practice because these cases are simple and already handled by the subtended angle test of original code. Handling rectangular and tube lights is more tricky and need further research.
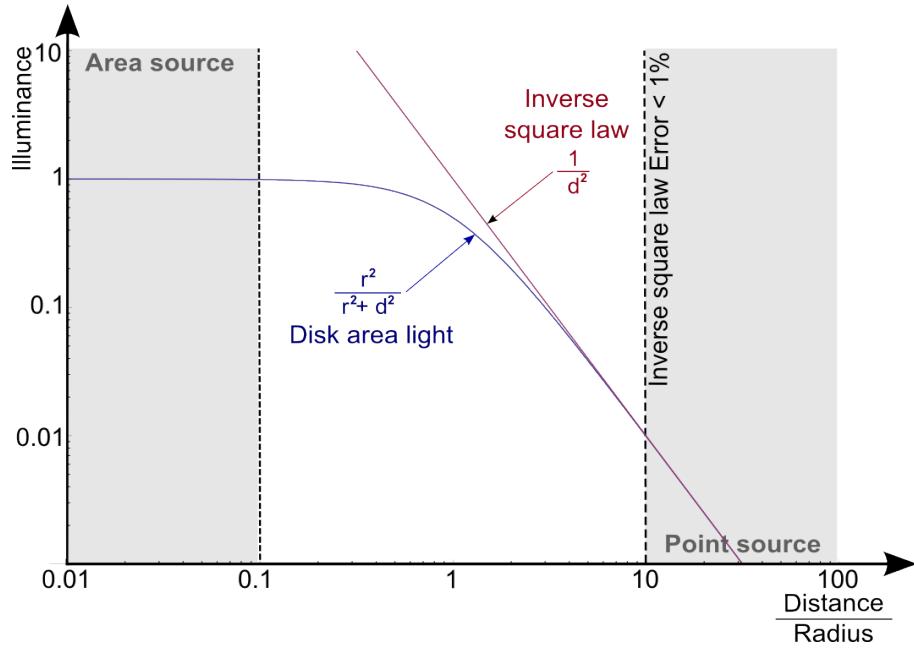
Figure 44: Example of a disk area light (using the frontal configuration for form factor) compared to a point light following the inverse square law. The graph is log scale. When $\frac{Distance}{Radius} > 10$ the error of inverse square law is less than 1%, thus 5 times its diameter.

```
1  float3 unnormalizedLightVector = P - worldPos;
2  float sqrDist = dot(unnormalizedLightVector, unnormalizedLightVector);
3  float3 L = normalize(unnormalizedLightVector);
4
5  // Example of sphere five time rules.
6  // In Frostbite is is still required to compensate for the light unit
7  // when using this rule in order to retrieve a punctual light.
8  // Thus in the case of the sphere, the "optimization" is useless.
9  float irradiance = FB_PI * sqrlightRadius * saturate(cosTheta) / sqrDist;
10
11 if (sqrDist < 100.0f * sqrlightRadius)
12 {
13    irradiance = irradianceSphereOrDisk(cosTheta, sinSigmaSqr);
14 }
15
16 // Example of disk five time rules.
17 // We need to take into account the orientation of the disk
18 // and like for sphere we need to compensate for the light unit.
19 // Note that this time we save a few of ALU but this still a useless optimization
20 float irradiance = FB_PI * sqrLightRadius * saturate(cosTheta) * saturate(dot(planeNormal, -L))
21                   / sqrDist;
22
23 if (sqrDist < 100.0f * sqrLightRadius)
24 {
25    irradiance = irradianceSphereOrDisk(cosTheta, sinSigmaSqr) * saturate(dot(planeNormal, -L));
26 }
```

Listing 14: Five times rules for sphere and a disk light.

4.7.4 Diffuse area light with Disney's diffuse

All previous derivations have been done for a Lambertian BRDF. But as mentioned in the material model section, our standard material uses a Disney diffuse term. The analytical method does not support it and it could be expensive to call an evaluation of the model for each sample in the structured

sampling approach. To handle it, we chose to apply the Disney diffuse evaluation for a single light direction onto the illuminance of Lambertian area light.

$$L_{out} = f_d(\mathbf{v}, \mathbf{l}) \; E(n) \tag{45}$$

The simplest choice for $\mathbf{l}$ would be to use the light position. For some light type, this approximation is good enough, see Figure 45. However a better choice is to take the direction of the most representative point as in Drobot's approach, see Figure 46. In *Frostbite*, we use the light position as input for Disney's diffuse term for performance reasons.
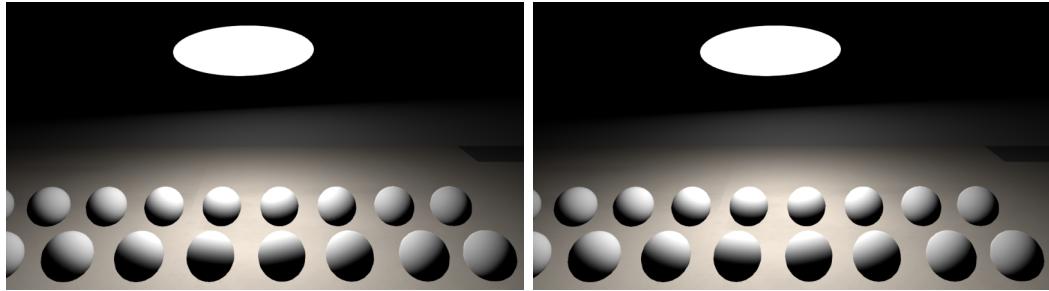
Figure 45: Spheres have increasing roughness from left to right. Left: Disk area lighting with light position as input to Disney's diffuse term. Right: Reference. The result is close.
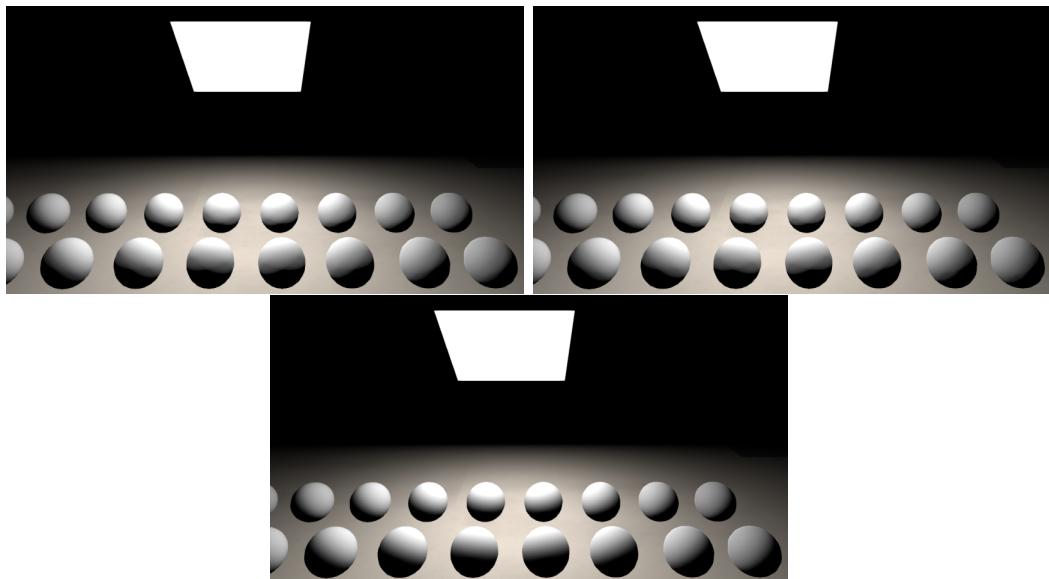
Figure 46: Spheres have increasing roughness from left to right. Top left: Rectangular area lighting with light position as input to Disney's diffuse term. Top right: MRP direction as input for light. Bottom: Reference. The MRP approximation is closer to the the reference.

It is possible to improve this approximation by taking into account the dominant direction of the BRDF lobe. This dominant direction processing is discussed in more detail in Section 4.9.3. We retrieve it by a shift of the surface's normal, see Listing 15. However the difference is too subtle and we chose not to apply the shift, to reduce shader cost.

```
float3 getDiffuseDominantDir(float3 N, float NdotV, float roughness)
{
    float a = 1.02341f * roughness - 1.51174f;
    float b = -0.511705f * roughness + 0.755868f;
```

54

```
5        lerpFactor = saturate((NdotV * a + b) * roughness);
6
7        return normalize(lerp(N, V, lerpFactor));
8    }
```

Listing 15: Function for computing the dominant direction of the lobe for Disney's diffuse term when evaluating area light.

4.7.5   Specular area lights

Specular area lights are a really complex problem within real time constraints. The number of input of the BRDF (view vector, f0, roughness) and the one of standard light's shape like rectangular (width, height, orientation, non constant intensity) make the pre-computation approach similar to light probe (see Section 4.9) difficult to do. The current solutions available in the literature do not stand up to comparison with the reference for a GGX specular model. The MRP approach of Drobot [Dro14b] works well only for the Phong model, and the shortest distance from reflection ray of Karis [Kar13] is lacking a good energy conservation term and doesn't behave well at grazing angles. We have tried another solution inspired by filtered importance sampling for area light [CPF10] as well as other math heavy solutions, but shader cost was prohibitive. We have decided not to match the ground truth and simply take the cheaper solution with the best visual. Our lights use Karis approach, however for disk and rectangle we have not been able to find any good coarse energy conserving term, see Figure 47.
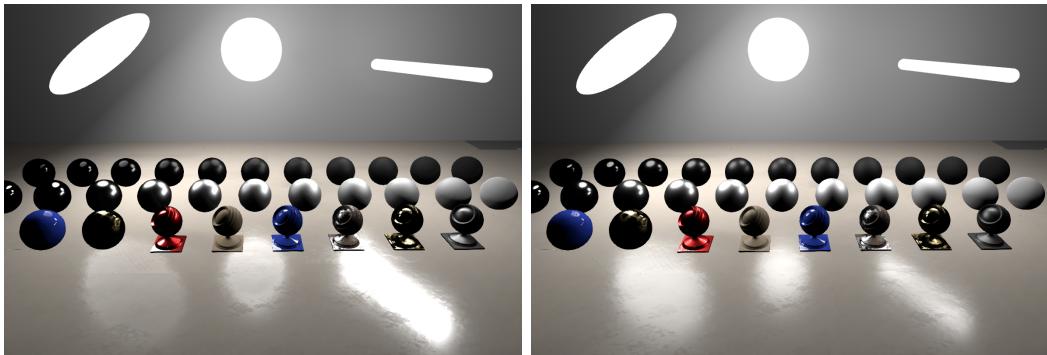
Figure 47: Scene with specular only lighting. Left: Various area lights type using Karis' approach for the specular. Right: reference.

To improve a bit the specular area lights integration for rough surfaces we have taken into account the dominant direction of the BRDF lobe. The dominant direction processing is discussed in more detail in Section 4.9.3. We retrieve it by a shift of the mirror direction, see Listing 16. The version for area light is simpler than the one use for the light probe. We have found it to be sufficient for our cases. Figure 48 show a comparison of mirror and dominant direction usage. This work fine with both G smith correlated term and uncorrelated.

```
1  float3 getSpecularDominantDirArea(float3 N, float3 R, float NdotV, float roughness)
2  {
3      // Simple linear approximation
4      lerpFactor = (1 - roughness);
5
6      return normalize(lerp(N, R, lerpFactor));
7  }
```

Listing 16: Functions for computing the dominant direction of the lobe for a microfacet GGX-based specular term when evaluating area light .
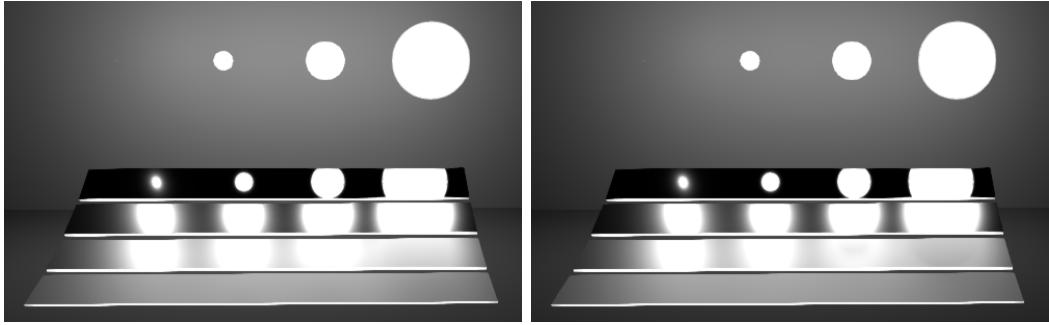
Figure 48: Left: Sphere area lighting approximation for various size with mirror direction integration. Right: With dominant direction. Notice the improvement on the rough surface for large sphere.

**Remark:** It is important to note that depending on light and surface properties, the illuminance calculated for diffuse area lights may not be applicable to the specular area light (Unlike for punctual light where Lambert's cosine always applies on both diffuse and specular terms). Indeed the solid angle subtended by the intersection of the importance cone of the BRDF and the light's shape could differ and should result in a different integration regarding Lambert's cosine.

4.8 Emissive surfaces

In the real world, emissive surfaces are similar to area lights. They emit light which illuminates nearby surfaces like any other light source. In games, it is not possible to treat the emissive part of a surface as a traditional light due to performance constraints. Within *Frostbite*, we made a distinction between: 1) emissive lights which display the source surface, and 2) area lights which emit light[35]. Emissive lights are generated at pixel precision inside shaders by providing emissive color and emissive intensity values, as for any other lights. The intensity is provided either in luminance ($cd.m^{-2}$) or in exposure value (EV). These emissive lights do not produce any lighting, only a visual color. They do however produce blooming when their intensity saturates the camera sensor. See Section 5.1. We can roughly designate three cases of emissive surfaces (see Figure 49):

    A The emissive material is underneath a non-emissive material.
    B The emissive material is on top of a non-emissive material.
   C The entire material is emissive.
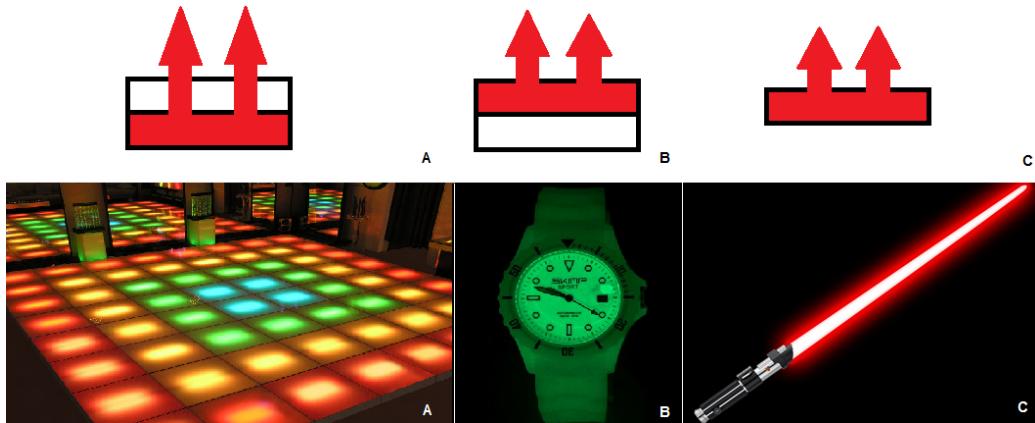
Figure 49: Three cases of emissive materials. A) The emissive material is underneath a non-emissive material, B) the emissive material is on top of a non-emissive material, C) the entire material is emissive.

---

[35]Our radiosity system supports emissive objects emitting light but only for radiosity and only at object precision.

Cases B and C are very close, except that with B, you may potentially see the material under the emissive layer. In *Frostbite*, we handle only case B which implicitly supports case C. Rendering emissive surfaces efficiently is not easy. To properly render emissive opaque objects in a deferred fashion and make them compatible with deferred decals (which are rendered after the GBuffer creation pass), it is necessary to store the emissive information into the GBuffer. This requires an additional buffer which is costly. In *Frostbite* we have different paths:

Transparent objects	Emissive applied during the rendering of the surface.
Forward opaque objects	Emissive applied during the rendering of the surface.
Deferred lit opaque objects with full emissive	Emissive applied in an extra rendering pass of the surface: the surface is rendered twice.
Deferred lit opaque objects with cheap emissive	Emissive stored in the radiosity buffer, and applied at the same time as the indirect lighting.

The indirect diffuse lighting is not composed with diffuse albedo at G-Buffer creation time, but later to allow decals to modify the diffuse albedo. In case artists want to trade performance for accuracy, we allow emissive values to be stored inside the radiosity buffer. The trouble with this approach is that emissive color is now coupled with diffuse albedo. When the radiosity buffer is applied, we multiply the radiosity buffer by the diffuse albedo and thus also the emissive term. Decals modifying diffuse albedo will also modify emissive in this case. That's why we call it "cheap" emissive. Listing 17. Another restriction is that this technique doesn't work with metallic objects as they have no diffuse.

```
1  // During GBuffer creation
2  float3 radiosity = ...
3  // Pack emissive with the radiosity (same unit, luminance)
4  radiosity += emissive * emissiveIntensity;
5  gBufferRadiosity = packLightingRGBA(radiosity);
6
7  // During radiosity application
8  float3 unpackedRadiosity = unpackLightingRGBA(gbufferRadiosity);
9  indirectDiffuse = unpackedRadiosity * data.diffuseAlbedo;
```

Listing 17: Cheap emissive managed for deferred case.

Figure 50 shows an example of results with forward and cheap deferred emissive. In this particular case the visual results match.
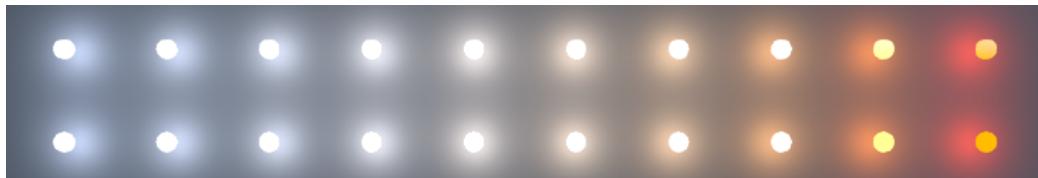
Figure 50: Example of emissive surfaces rendered in both forward (top) and deferred fashion (bottom). The spheres are all emissive objects and sphere area lights have been added at the same position as the emissive spheres.

Remark: We have developed a tool to automatically generate emissive shapes with the correct intensity at the location of area lights. An example is shown in Figure 51. This actually produces a convincing visual scene and also helps artists to debug their lights.
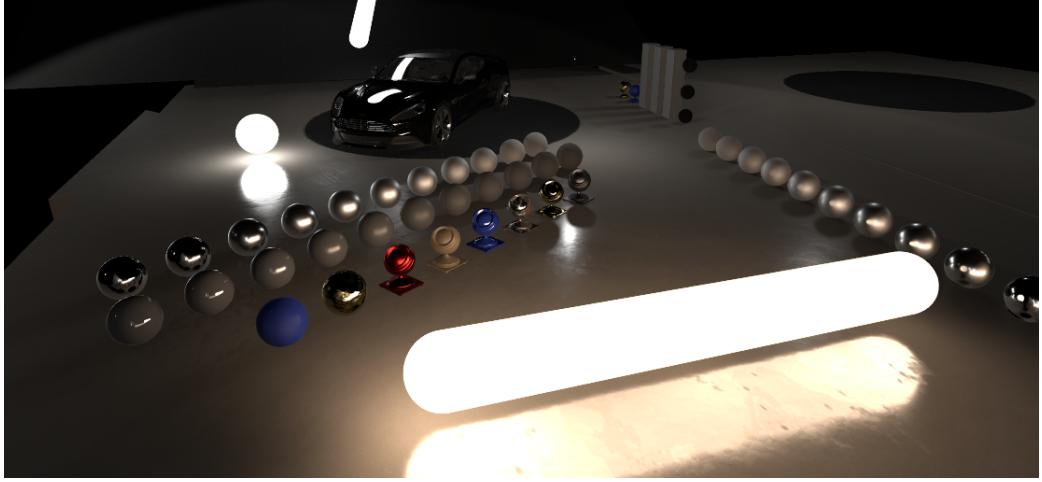
Figure 51: Debug mode for visualizing lights as emissive surfaces.

4.9 Image based lights

Image based lights (IBLs) allow us to represent the incident lighting surrounding a point. This surrounding lighting is important for making an object "fit" into its environment. Often referred to as "reflection" by artists, this incident lighting needs to be consistently applied to all parts of the BRDF equation f, i.e. for a standard material both the specular f_r part but also the f_d part. For more advanced materials like layered materials, all layers need to be affected by this lighting. "Faking" this lighting by adding a reflection texture directly in the shader breaks the *key separation* between lighting and material information, making it hard to reuse an asset in different environments. Computing the interaction between an IBL L and a BRDF f is a costly operation that requires evaluating the following integral:

$$L(\mathbf{v}) = \int_{\Omega} f(\mathbf{l}, \mathbf{v}, \Theta) \ L(\mathbf{l}) \mathrm{d}\mathbf{l} \tag{46}$$

With the view direction $\mathbf{v}$, the material model f, and the parameters Θ (Fresnel, roughness, albedo, etc.). To continuously fit an object into its environment and still provide good reflection approximations, we need to be able to provide reflections in all situations. For this, we rely on four types of IBLs, inspired by Drobot [Dro13]. Each of these types allows us to represent a certain type or range of incident lighting:

- **Distant light probe**: capture the surrounding distant lighting, which does not contains any parallax (i.e. sky, distant buildings, background drops, etc.). This is the least accurate type of reflections but is available at any time.

- **Local light probes**: capture all objects encompassed into a certain area from a single point of view (cube map). These captures are re-projected onto simple proxy geometries (e.g. a sphere or a box) carefully adjusted by artists in order to match the surrounding geometry. This type of reflection is more accurate than distant light probes, but object lighting and parallax are not perfectly captured.

- **Screen Space Reflections**: capture reflections based on the light buffer by ray-marching against the depth buffer. This catches short to medium range reflections and ensures good contact hardening reflections. This is one of our most accurate sources of reflection.

- **Planar Reflections**: capture reflections by rendering the scene mirrored by a plane which is either automatically set up by the engine or manually by artists. This type of reflection makes

the assumption that reflections are lying on a plane which works well for almost flat surfaces like roads, buildings or water.

Static vs. Dynamic: Distant and local light probes contain usually "static" lighting information, as they captured the surrounding lighting at a particular instant. Their content can be refreshed on demand or every frame depending of the requirements and the allocated performance budget. SSR and planar reflections contains "dynamic" lighting information, since they are updated every frame due to their view dependent nature. Their computations can be spread among several frames in order to reduce their cost.

While we will discuss SSR and planar reflections briefly, this section is focused mainly on local and distant light probes. First, we will describe the acquisition of light probes and their lighting units. Then we will describe their filtering, their evaluation, and finally the composition of the different IBLs.

4.9.1 Light probes acquisition and unit

In *Frostbite*, IBLs are by definition related to image data. They all use luminance units as this is the output of our lighting pipeline.

4.9.1.1 Distant light probe

Distant light probes capture the surrounding environment represented as a cube-map. Artists have two ways to acquire a distant light probe:

- Acquiring the lighting through a physically based sky optionally composed with background.
- Using an acquired High Dynamic Range Image (HDRI) from a real world camera.

Frostbite uses a physically based sky that can be used to capture the distant light probe. The distant light probe can be refreshed on demand when dealing with changing condition (time of day cycles, weather changes, etc.). Sky lighting is converted to luminance like any other light. At capture time, we only consider the lighting pipeline. We remove all post-process including any color management operation (Tone mapping, color grading...) and we store the resulting luminance in a HDR texture format RGBA16F.

Making good usage of real world distant light probes requires knowledge of what has been acquired. Acquired HDR images are rarely used for final lighting in-game due to their static nature. They are rather used as incident lighting for designing assets and verifying that their material properties react correctly to natural lighting. However when acquired HDRIs need to be mixed with in-game lighting, one needs to be careful. It is not obvious what is stored in a HRDI texel. One could think a camera would output luminance values but the camera response and post-process steps cut both the highest and the lowest luminance values. Outputted pixels do not represent luminance values anymore but rather a device-dependent value which is related to the original scene luminance. The actual process detailing the conversion from luminance to pixel values is detailed in Section 5.1.

HRDI creation: With an LDR camera, following a complex process of capture and reconstruction allows one to get a HDR image in **absolute luminance range** [DM97]. The process implies taking a scene with multiple exposures and combining them later with software (like Luminance HDR [Ana]). The software reconstructs the original scene luminance by inverting the device-dependent response

curve. The double difficulties of acquiring good image[36] and being able to identify the camera response curve imply that most HDR images produced by artists only represent **relative luminance** or pre-exposed luminance, see Figure 52. In order to mix such a distant light probe with other light types, we provide a multiplier to artists for tweaking the relative luminance value. With manual calibration based on a Mac-Beth chart, camera settings and well known luminance values, artists can find a good factor to recover the absolute luminance[37]. In a correctly reconstructed image, a clear sky should have a luminance around 8000 $cd.m^{-2}$ and an overcast around 2000 $cd.m^{-2}$. For night scenes, values of 3000 to 5000 $cd.m^{-2}$ are not uncommmon under street light illumination. The luminance of the moon is around 2500 $cd.m^{-2}$, other objects are rather quite dark with values $< 1cd.m^{-2}$ when lit only by environmental lighting [McN].

Strong light sources: With acquired HDRI, it is necessary to remove strong light sources from the light probe to avoid noise during pre-integration (see the next section) and to handle their visibility. For instance, applying the sun as a directional light instead of baking it into the light probe, enables us to handle sun shadowing properly. Removing the sun from a HDRI requires us to simply copy and paste a small part of the surrounding sky onto the sun.

Figure 52: Example of HDRI texture used for distant lighting. This HDRI stores relative luminance values.
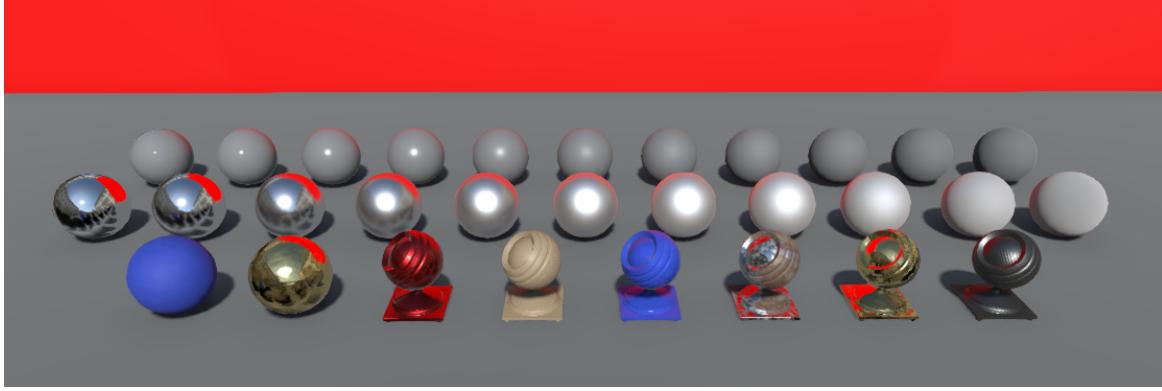
4.9.1.2 Local light probes

Local light probes capture the surrounding objects in a limited area around their volumes. The goal is to match the local environment. This acquisition is always performed in-engine and can be either: "baked" off-line or captured once, refreshed on demand, or refreshed every frame at runtime. The choice depends on budget and requirements (moving objects, change of lighting conditions, etc.). Capturing local light probes causes rendering issues:

- **Order-dependency:** A scene cannot be lit with a local light probe since they have not been captured yet. This chicken and egg issue causes a capturing order dependency if local light probes are captured one after the other.

---

[36]The use of a neutral density filter can help to get proper luminance reconstruction, particularly when trying to capturing the sun [Rei+05].

[37]We have observed that Maxwell and VRay consider the HDRI as a radiance unit, not luminance unit. To match our results we need to divide the HDRI in this software by 683. Canada [Cañ14] has confirmed that for Maxwell this was due to historical reasons regarding badly acquired old HDRI.

(a) Result.

(b) Color channel.   (c) Alpha channel.

Figure 53: Top: Application of a local probe around some object. Notice the red reflection coming from the wall hitting the objects placed on the ground. Bottom left: Color acquired into the local light probe. Notice the yellow color emit by the gold metallic object. Bottom right: Alpha channel information stored into the local light probe.

- **Metallic surfaces:** Metallic surfaces are problematic since they have no diffuse contribution nor specular contribution at capture time, resulting in black appearance. It is possible to capture local light probes several times to simulate light bounces, but a metallic room case will require a large number of bounces. Another possibility is to rely on distant light probe lighting but obvious light leaking can appear for indoor environments.
- **View-dependent effects:** Local light probes are captured from a single point (i.e. the cube map center), and this incident lighting coming from glossy and mirror-like surfaces will be wrong from points of view other than the captured one, due to the view dependency.

To solve these problems, we disable the material specular component during light probe acquisition. Metallic surfaces which only contain a specular term, are approximated as diffuse surfaces by using their Fresnel f_0 as diffuse albedo. During the capture, we also store the visibility of surrounding objects into the alpha channel. This will be used later during the IBLs composition to fade the contribution of local light probes. Figure 53 shows an example of captured local light probe.

Participating media: in foggy environments for instance, one should ideally take into account the medium transmittance when evaluating a local light probe contribution. Acquiring the fog directly into a local light probe does not work correctly since it models a 2D function (theta, phi) instead of a 3D one (theta, phi, depth). Thus, with this simple 2D function, it is not possible to compute the actual transmittance between the shaded point and the local light probe proxy geometry.

|		
(a) 0°	(b) 40°	(c) 80°
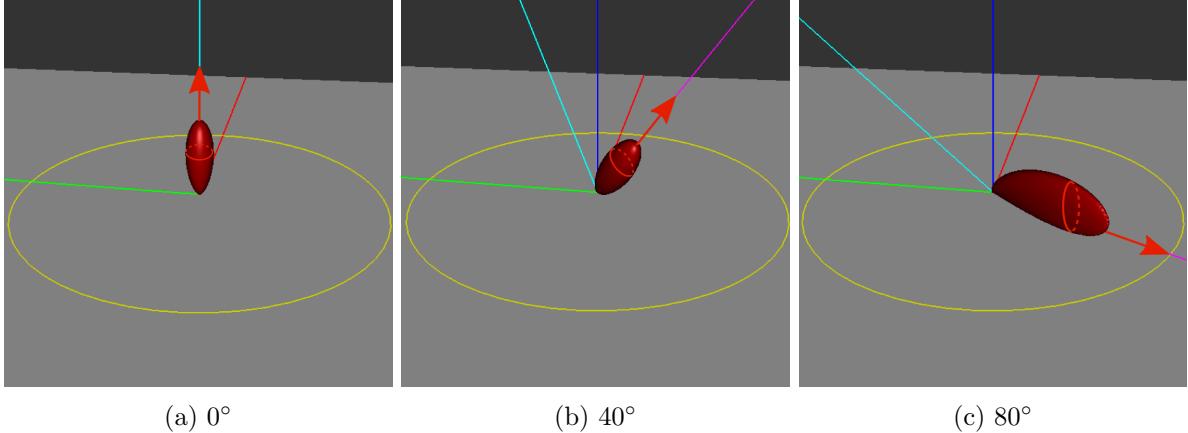
Figure 54: The shape of a microfacet BRDF (with a GGX NDF) for various view angles. Note how the anisotropy of the lobe increases with the view angle.

4.9.2   Light probe filtering
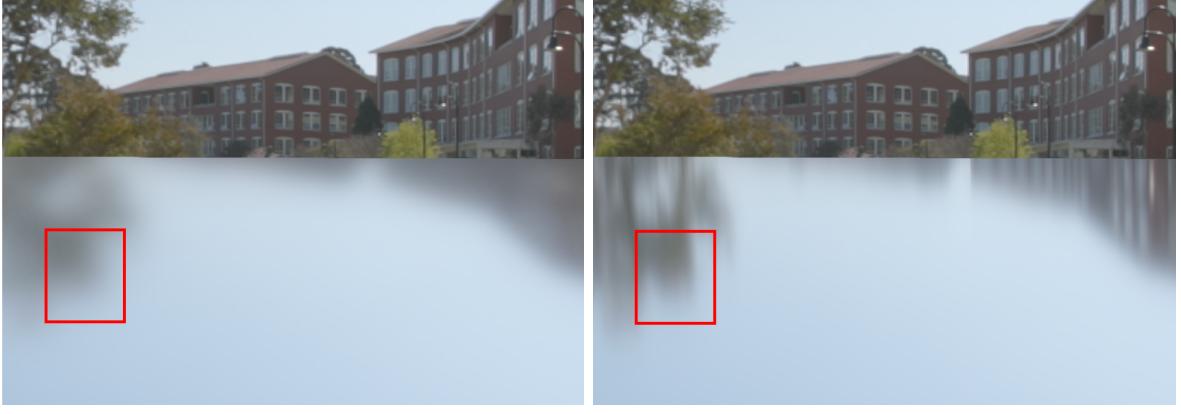
The integral 46 depends on the view direction $\mathbf{v}$, the material model f, and its parameters Θ. Usually such an integral does not have an analytical solution and requires numerical evaluation, typically by a stochastic integration technique such as Monte-Carlo. Evaluating such an integral directly would require a lot of samples for each pixel every frame. While this is possible, it is not realistic in the context of a high performance application such as a game. Importance sampling allows us to reduce the number of samples, but even with multiple-importance sampling (MIS) the required number of samples to evaluate each pixel each frame is too high.

Specular pre-integration: To simplify this evaluation, we can pre-integrate the integral by making some approximations. Pre-integrating this equation for every $\mathbf{v}$ and Θ would require a huge memory footprint. Thus, a first approximation is to remove the view dependency. This leads to a coarse approximation of the BRDF but it is an acceptable trade-off: the shape of a BRDF based on the micro-facets framework and/or half-angle parametrization is strongly dependent on the view angle as shown on Figure 54. At normal incident direction, the shape of a BRDF is isotropic. At grazing angles the shape of a BRDF is anisotropic. Removing the view dependency for pre-integrating Equation 46 would make the assumption that the BRDF shape is isotropic at all view angles. This leads to key visual differences, preventing stretched reflections. This approximation can be quite noticeable on flat surfaces as shown on Figure 55 but less on curvy surfaces[38].

In order to further reduce the dimensionality of the pre-integration, we need to reduce the number of parameters to $\Theta = (f_0, \alpha)$ with α the roughness, and f_0 the Fresnel value at 0° incident angle. As mentioned earlier, Equation 46 needs to be numerically integrated. For "pre-baked" local light probes, this integration can be done without any performance concerns as all computations are done off-line. However, for on-demand light probes, continuously refreshed local light probes, or fast manipulation by artists (placement of local light probes, rotation of the distant light probe) performance is critical, and we need to achieve a good quality/speed ratio. Importance sampling[39] can be used to speed up

---

[38]One could either use pre-filtered importance sampling [KC08] for correctly evaluating the integral and recover the stretched reflection, or use pre-integration and an isotropic lobe decomposition as shown by Green et al. [GKD07].

[39]We have tried to use MIS in the pre-integration but the building cost for probability tables and the binary search

(a) Isotropic lobe assumption.　　　　　　　　(b) Reference.

Figure 55: The stretched reflections at grazing angles for a microfacet BRDF (with a GGX NDF). Left: the approximation. Right: the correct behavior.

the convergence by focusing the calculations on the important parts of the integrand :

$$L(v) = \frac{1}{N} \sum_i^N \frac{f_r(\mathbf{l}_i, \mathbf{v}, \Theta) \ L(\mathbf{l}_i)}{p_r(\mathbf{l}_i, \mathbf{v}, \Theta)} \langle \mathbf{n} \cdot \mathbf{l}_i \rangle \tag{47}$$

p_r represents the PDF of the BRDF. l_i are samples generated from p_r. Karis has shown [Kar13] that in the case of microfacet BRDFs, the integral 47 can be approximated by decomposing it into a product of two terms: LD and DFG. With $p_r = D(\mathbf{h}, \alpha) \langle \mathbf{n} \cdot \mathbf{h} \rangle J(\mathbf{h})$ and the Jacobian J of the transformation from half-vector to lighting vector $J(\mathbf{h}) = \frac{1}{4\langle \mathbf{v} \cdot \mathbf{h} \rangle}$, we get:

$$L(v) = \frac{1}{N} \sum_i^N \frac{f_r(\mathbf{l}, \mathbf{v}, \Theta) \ L(\mathbf{l})}{p_r(\mathbf{l}, \mathbf{v}, \Theta)} \ \langle \mathbf{n} \cdot \mathbf{l} \rangle \tag{48}$$

$$= \frac{1}{N} \sum_i^N \frac{D(\mathbf{h}, \alpha) \ F(\mathbf{v}, \mathbf{h}, f_0, f_{90})) \ G(\mathbf{l}, \mathbf{v}, \mathbf{h}, \alpha)}{4\langle \mathbf{n} \cdot \mathbf{l} \rangle \langle \mathbf{n} \cdot \mathbf{v} \rangle} \ \frac{1}{p_r(\mathbf{l}, \mathbf{v}, \Theta)} \ L(\mathbf{l}) \ \langle \mathbf{n} \cdot \mathbf{l} \rangle \tag{49}$$

$$= \frac{1}{N} \sum_i^N \frac{D(\mathbf{h}, \alpha) \ F(\mathbf{v}, \mathbf{h}, f_0, f_{90})) \ G(\mathbf{l}, \mathbf{v}, \mathbf{h}, \alpha)}{4\langle \mathbf{n} \cdot \mathbf{v} \rangle} \ \frac{1}{p_r(\mathbf{l}, \mathbf{v}, \Theta)} \ L(\mathbf{l}) \tag{50}$$

$$= \frac{1}{N} \sum_i^N \frac{D(\mathbf{h}, \alpha) \ F(\mathbf{v}, \mathbf{h}, f_0, f_{90}) \ G(\mathbf{l}, \mathbf{v}, \mathbf{h}, \alpha)}{4\langle \mathbf{n} \cdot \mathbf{v} \rangle} \ \frac{4 \ \langle \mathbf{v} \cdot \mathbf{h} \rangle}{D(\mathbf{h}, \alpha) \ \langle \mathbf{n} \cdot \mathbf{h} \rangle} \ L(\mathbf{l}) \tag{51}$$

$$= \frac{1}{N} \sum_i^N \frac{F(\mathbf{v}, \mathbf{h}, f_0, f_{90}) \ G(\mathbf{l}, \mathbf{v}, \mathbf{h}, \alpha)}{\langle \mathbf{n} \cdot \mathbf{v} \rangle \langle \mathbf{n} \cdot \mathbf{h} \rangle} \ \langle \mathbf{v} \cdot \mathbf{h} \rangle \ L(\mathbf{l}) \tag{52}$$

$$\approx \underbrace{\frac{1}{N} \sum_i^N \frac{F(\mathbf{v}, \mathbf{h}, f_0, f_{90}) \ G(\mathbf{l}, \mathbf{v}, \mathbf{h}, \alpha)}{\langle \mathbf{n} \cdot \mathbf{v} \rangle \langle \mathbf{n} \cdot \mathbf{h} \rangle} \ \langle \mathbf{v} \cdot \mathbf{h} \rangle}_{\text{DFG term}} \ \underbrace{\frac{1}{\sum_i^N \langle \mathbf{n} \cdot \mathbf{l} \rangle} \sum_i^N L(\mathbf{l}) \langle \mathbf{n} \cdot \mathbf{l} \rangle}_{\text{LD term}} \tag{53}$$

This decomposition leads to the two independent terms, *DFG* and *LD*, which can be pre-computed separately. The *LD* term needs to be computed for each light probe, while the *DFG* can be computed

cost for each sample was too high compared to the convergence gain.

once and reused for all light probes. Listing 18 shows the DFG term. One can notice an extra $\langle \mathbf{n} \cdot \mathbf{l} \rangle$ in the LD term as well as a different weighting $\frac{1}{\sum_i^N \langle \mathbf{n} \cdot \mathbf{l} \rangle}$. These empirical terms have been introduce by Karis to allows to improve the reconstructed lighting integral which suffers from coarse hypothesis of separability of this integral. There is no mathematical derivation for these terms, goal was to have an exact match with a constant $L(\mathbf{l})^{40}$. As shown by [Kar13], by using the Schlick formulation of the Fresnel term:

$$F(\mathbf{v}, \mathbf{h}, f_0, f_{90}) = f_0 + (f_{90} - f_0)(1 - \langle \mathbf{v} \cdot \mathbf{h} \rangle)^5 \tag{54}$$

The DFG term can be represented by a 2D function which only depends on the view angle v and the roughness α, leaving f_0 and f_{90} out of the pre-computation. This 2D function stores two terms: DFG_1 and DFG_2.

$$DFG(\mathbf{v}, \mathbf{l}, f_0, f_{90}, \alpha) = \frac{1}{N} \sum_i^N \frac{(f_0 + (f_{90} - f_0)(1 - \langle \mathbf{v} \cdot \mathbf{h} \rangle)^5) \, G(\mathbf{l}, \mathbf{v}, \mathbf{h}, \alpha)}{\langle \mathbf{n} \cdot \mathbf{v} \rangle \langle \mathbf{n} \cdot \mathbf{h} \rangle} \langle \mathbf{v} \cdot \mathbf{h} \rangle \tag{55}$$

$$= \frac{1}{N} \sum_i^N (f_0 + (f_{90} - f_0) \, Fc) \, G_{Vis} \text{ with } G_{Vis} = \frac{G(\mathbf{l}, \mathbf{v}, \mathbf{h}, \alpha)}{\langle \mathbf{n} \cdot \mathbf{v} \rangle \langle \mathbf{n} \cdot \mathbf{h} \rangle} \langle \mathbf{v} \cdot \mathbf{h} \rangle, Fc = (1 - \langle \mathbf{v} \cdot \mathbf{h} \rangle)^5 \tag{56}$$

$$= \frac{1}{N} \sum_i^N f_0 \, G_{Vis} + f_{90} \, Fc \, G_{Vis} - f_0 \, Fc \, G_{Vis} \tag{57}$$

$$= f_0 \underbrace{\frac{1}{N} \sum_i^N (1 - Fc) \, G_{Vis}}_{DFG_1 \text{ term}} + f_{90} \underbrace{\frac{1}{N} \sum_i^N Fc \, G_{Vis}}_{DFG_2 \text{ term}} \tag{58}$$

```
1
2  void importanceSampleCosDir (
3      in float2 u,
4      in float3 N,
5      out float3 L,
6      out float NdotL,
7      out float pdf)
8  {
9      // Local referencial
10     float3 upVector = abs(N.z) < 0.999 ? float3(0,0,1) : float3(1,0,0);
11     float3 tangentX = normalize( cross( upVector, N ) );
12     float3 tangentY = cross( N, tangentX );
13
14     float u1 = u.x;
15     float u2 = u.y;
16
17     float r = sqrt(u1);
18     float phi = u2 * FB_PI * 2;
19
20     L = float3(r*cos(phi), r*sin(phi), sqrt(max(0.0f,1.0f-u1)));
21     L = normalize(tangentX * L.y + tangentY * L.x + N * L.z);
22
23     NdotL = dot(L,N);
24     pdf = NdotL * FB_INV_PI;
25 }
26
27 float4 integrateDFGOnly (
```

---

[40]We have try few different better mathematically define decomposition. But even if theory were better the visual result was always worse than the one get with Karis decomposition.

```
28        in float3 V,
29        in float3 N,
30        in float roughness)
31    {
32        float NdotV            = saturate(dot(N, V));
33        float4 acc             = 0;
34        float accWeight        = 0;
35
36        // Compute pre-integration
37        Referential referential = createReferential(N);
38        for (uint i=0; i<sampleCount; ++i)
39        {
40            float2 u    = getSample(i, sampleCount);
41            float3 L    = 0;
42            float NdotH = 0;
43            float LdotH = 0;
44            float G     = 0;
45
46            // See [Karis13] for implementation
47            importanceSampleGGX_G(u, V, N, referential, roughness, NdotH, LdotH, L, G);
48
49            // specular GGX DFG preIntegration
50            float NdotL = saturate(dot(N, L));
51            if (NdotL>0 && G > 0.0)
52            {
53                float GVis  = G * LdotH / (NdotH * NdotV);
54                float Fc    = pow(1-LdotH, 5.f);
55                acc.x       += (1-Fc) * GVis;
56                acc.y       += Fc*GVis;
57            }
58
59            // diffuse Disney preIntegration
60            u = frac(u + 0.5);
61            float pdf;
62            // The pdf is not use because it cancel with other terms
63            // (The 1/PI from diffuse BRDF and the NdotL from Lambert's law).
64            importanceSampleCosDir(u, N, L, NdotL, pdf);
65            if (NdotL>0)
66            {
67                float LdotH = saturate(dot(L, normalize(V + L)));
68                float NdotV = saturate(dot(N, V));
69                acc.z += Fr_DisneyDiffuse(NdotV, NdotL, LdotH, sqrt(roughness));
70            }
71
72            accWeight += 1.0;
73        }
74
75        return  acc * (1.0f / accWeight);
76    }
```

Listing 18: Pre-integrated DFG function for both specular GGX and diffuse Disney BDRF

The LD term needs to be recomputed each time the incident lighting changes, such as a time of day change. This means that LD needs to be computed at runtime and thus needs to be fast and robust. Using importance sampling improves the convergence but requires quite a few samples. Pre-filtered importance sampling introduced by Krivànek et al. [KC08] allows us to reduce the number of samples by relying on pre-filtered values for low probability samples. This remarkably improves the convergence, see Figure 56 at the cost of introducing a small bias[41]. Listing 19 shows the LD term. In a highly contrasted environment, some noise can still be observed, especially for moderate and high roughness α values. This is caused mainly by the large support of the BRDF lobe over the hemisphere (especially for long tailed NDFs, like GGX) with highly contrasted light probe (a few pixels with really high intensity). For removing this noise, one can either increase the number of samples, or trade this noise for correlated bias, by not rotating/jittering the sample pattern per texel. This will lead to some

---

[41]In *Frostbite* we use 32 samples for our integration.

Filtered Importance Sampling
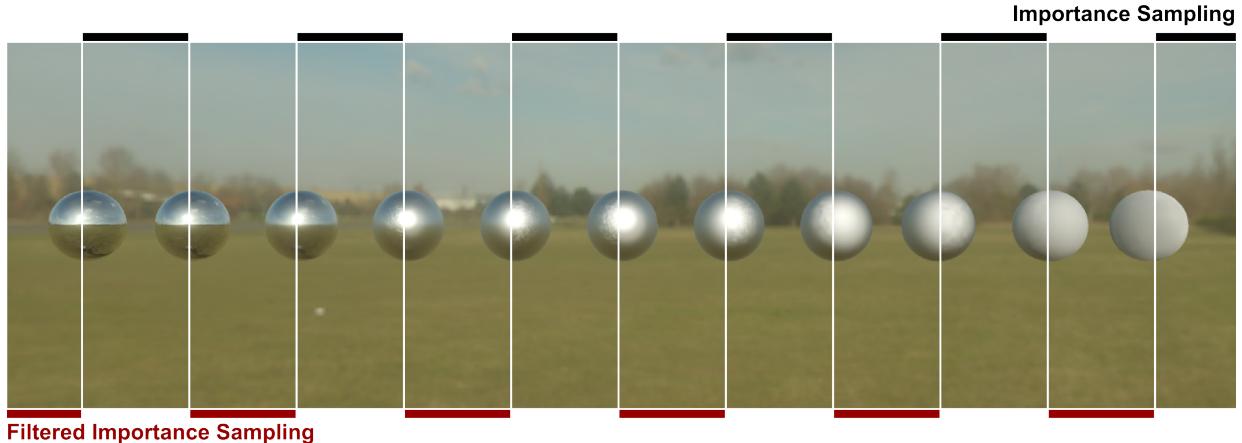
Figure 56: Compare pre-convolution results using importance sampling and pre-filtered importance sampling. Both images use the same number of samples (64 samples per texel).

ghosting artifacts but they are usually less noticeable than noise.

```
float3 integrateCubeLDOnly(
    in float3 V,
    in float3 N,
    in float roughness)
{
    float3 accBrdf = 0;
    float  accBrdfWeight = 0;
    for (uint i=0; i<sampleCount; ++i)
    {
        float2 eta = getSample(i, sampleCount);
        float3 L;
        float3 H;
        importanceSampleGGXDir(eta, V, N, roughness, H, L);
        float NdotL = dot(N,L);
        if (NdotL>0)
        {
            // Use pre-filtered importance sampling (i.e use lower mipmap
            // level for fetching sample with low probability in order
            // to reduce the variance).
            // (Reference: GPU Gem3)
            //
            // Since we pre-integrate the result for normal direction,
            // N == V and then NdotH == LdotH. This is why the BRDF pdf
            // can be simplifed from:
            //     pdf = D_GGX_Divide_Pi(NdotH, roughness)*NdotH/(4*LdotH);
            // to
            //     pdf = D_GGX_Divide_Pi(NdotH, roughness) / 4;
            //
            // The mipmap level is clamped to something lower than 8x8
            // in order to avoid  cubemap filtering issues
            //
            //  - OmegaS: Solid angle associated to a sample
            //  - OmegaP: Solid angle associated to a pixel of the cubemap
            float NdotH = saturate(dot(N, H));
            float LdotH = saturate(dot(L, H));
            float pdf = D_GGX_Divide_Pi(NdotH, roughness) * NdotH/(4*LdotH);
            float omegaS = 1.0 / (sampleCount * pdf);
            float omegaP = 4.0 * FB_PI / (6.0 * width * width);
            float mipLevel = clamp(0.5 * log2(omegaS/omegaP), 0, mipCount);
            float4 Li = IBLCube.SampleLevel(IBLSampler, L, mipLevel);

            accBrdf += Li.rgb * NdotL;
            accBrdfWeight += NdotL;
```

```
44          }
45      }
46      return accBrdf * (1.0f / accBrdfWeight);
47 }
```

Listing 19: Pre-filtered importance sampling function.

Diffuse pre-integration: Until now, we have only considered the integration of the incident lighting with the specular f_r part. As mentioned earlier, it is crucial that all material parts receive the same incident lighting. Thus lighting needs to also be integrated against the diffuse part f_d. Since f_d depends on the view angle $\mathbf{v}$ and the roughness α as we use the diffuse Disney BRDF, we rely on a similar pre-integration as for the specular part, by decomposing the integral in two terms DFG and LD. For the LD term, the incident lighting is integrated with a constant Lambertian lobe weighted by $\langle \mathbf{n} \cdot \mathbf{l} \rangle$. This can be done efficiently in a spherical harmonic basis as shown by Ramamoorthi et al. [RH01], or by importance sampling. We use the importance sampling with sample distribution following a Cosine distribution (instead of following the Disney lobe distribution), as it is adapted to this low angular frequency lobe[42]. Mean $p_r = \frac{\langle \mathbf{n} \cdot \mathbf{l} \rangle}{\pi}$:

$$L(v) = \frac{1}{N} \sum_i^N \frac{f_d(\mathbf{l}, \mathbf{v}, \Theta)\ L(\mathbf{l})}{p_r(\mathbf{l}, \mathbf{v}, \Theta)}\ \langle \mathbf{n} \cdot \mathbf{l} \rangle \tag{59}$$

$$= \frac{1}{N} \sum_i^N \frac{\pi\ f_d(\mathbf{l}, \mathbf{v}, \Theta)\ L(\mathbf{l})}{\langle \mathbf{n} \cdot \mathbf{l} \rangle}\ \langle \mathbf{n} \cdot \mathbf{l} \rangle \tag{60}$$

$$= \frac{1}{N} \sum_i^N \pi\ f_d(\mathbf{l}, \mathbf{v}, \Theta)\ L(\mathbf{l}) \tag{61}$$

$$\approx \underbrace{\frac{\pi}{N} \sum_i^N f_d(\mathbf{l}, \mathbf{v}, \Theta)}_{\text{DFG term}}\ \underbrace{\frac{1}{N} \sum_i^N L(\mathbf{l})}_{\text{LD term}} \tag{62}$$

The LD term computation can be seen in Listing 20. The DFG term is computed by integrating the diffuse Disney BRDF for various view angles $\mathbf{v}$ and roughness α, see Listing 18.

```
1  float4 integrateDiffuseCube(in float3 N)
2  {
3      float3 accBrdf = 0;
4      for (uint i=0; i<sampleCount; ++i)
5      {
6          float2 eta = getSample(i, sampleCount);
7          float3 L;
8          float NdotL;
9          float pdf;
10         // see reference code in appendix
11         importanceSampleCosDir(eta, N, L, NdotL, pdf);
12         if (NdotL>0)
13             accBrdf += IBLCube.Sample(incomingLightSampler, L).rgb;
14     }
15     return float4(accBrdf * (1.0f / sampleCount), 1.0f);
16 }
```

Listing 20: diffuse importance sampling function.

---

[42]In [AS00] Ashikhmin makes the same observation for his diffuse model which has similarities with ours.

(a) Linear (Exponent 1)   (b) Exponent 1.5   (c) Exponent 2
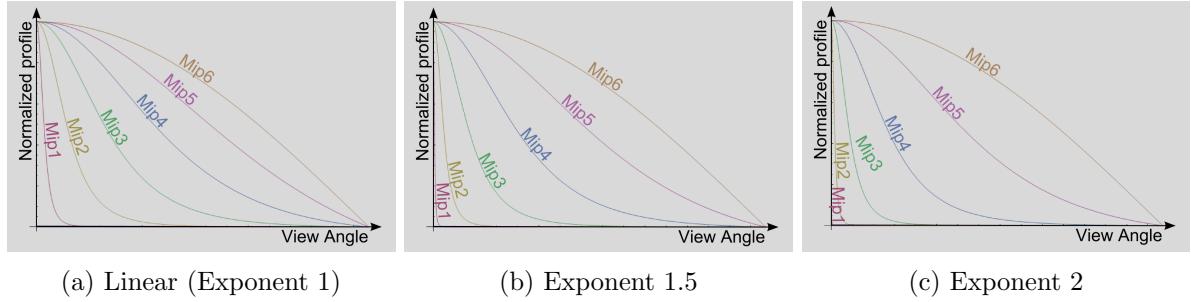
Figure 57: Various mapping functions between roughness and mip level. An exponent-2 mapping is able to get a better distribution of the profiles among all mip levels.

Linear (Exponent 1)

**Squared** (Exponent 2)
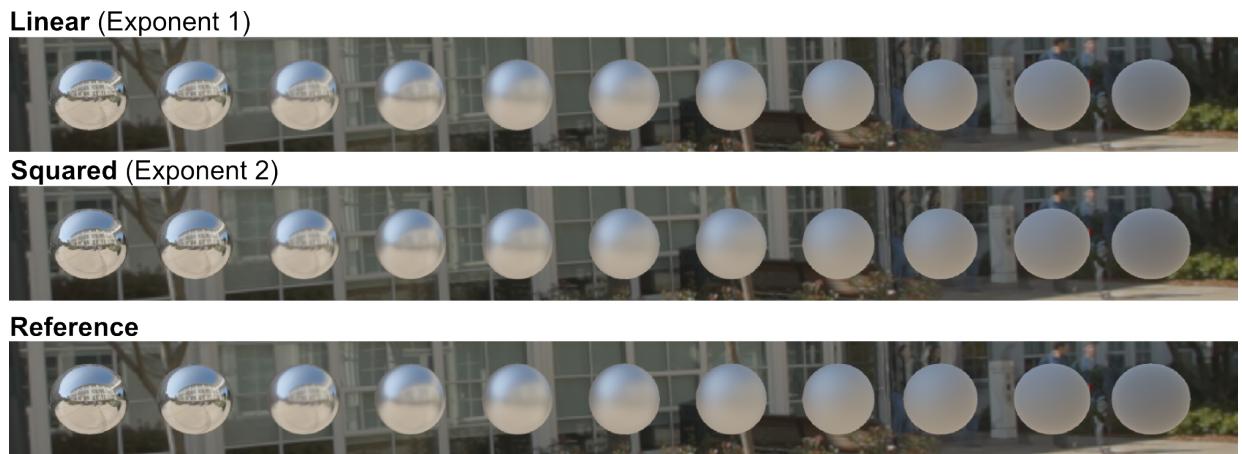
Reference

Figure 58: Comparison of two mapping functions for storing roughness into mip-level, against reference. While differences are subtle, a squared mapping gives slightly better distribution of the profiles among all mip levels

Storage: Storing the result of these pre-integrations is often bypassed in the literature but is an important detail. The DFG term can easily be stored into a 2D texture with a resolution of 128^2. The LD term is a 3D function which can be stored into a set of 16 bit floating-point cube textures (In *Frostbite* we use resolutions from 128^2 to 512^2 depending on the desired quality, but found 256^2 as the minimum to be able to represent smooth/mirror-like surfaces with good quality). Since the frequency of the pre-integrated lighting is lower and lower as α increases, we can store the pre-integrated result into mip-levels[43]. This storage is defined by a function mapping α_{lin} (the perceptually-linear roughness) to a *mipLevel*. As shown on Figure 57 and Figure 58, an exponent-2 mapping of the α_{lin} gives good results. While a linear mapping gives a better overall distribution and preserves the medium range roughness, it introduces a slight blur for low roughnesses. We decided to go with exponent-2:

$$mipLevel = \sqrt{\alpha_{\text{lin}}} \; mipCount \tag{63}$$

4.9.3 Light probe evaluation

At runtime, both local and distant light probes are evaluated to take into account light coming from the environment. While both diffuse and specular pre-integrations are done in a view independent way, the actual diffuse f_d and specular f_r terms are view dependent. This means that their lobe direction

---

[43]The first mip level is not pre-convolved in order to support perfect specular surfaces. This allows us to save some computation at the same time.

depends on the view vector $\mathbf{v}$. For the specular f_r term, instead of using the mirror reflection for fetching the pre-integrated value, we evaluate the BRDF in its principal direction (i.e. "the off-specular peak" seen in Section 3.1).

To model this shifting, we have observed the highest value deviation depending on the view angle and the material roughness. A complete analysis is available as a Mathematica companion file of this document. Our proposed model poorly captures the correct behaviour for dielectric materials at low view angles. This is because the actual lobe only "emerges" at a certain angle. Fortunately, since Fresnel values slowly increase with the view angle, this error is barely noticeable and this approximation remains good enough, see Listing 21. We have also observed after several experiments that a simpler approximation of the specular lobe dominant direction gives better results than our original best fit approximation, see Listing 22. The result of the evaluation and comparison of the method are shown in Figure 59. We have perform an approximation of the dominant direction for a G Smith correlated term and an uncorrelated one. It appear that our simple formula work nicely for both.

```
1  // This is an accurate fitting of the specular peak,
2  // but due to other approximation in our decomposition it doesn't perform well
3  float3 getSpecularDominantDir(float3 N, float3 R, float NdotV, float roughness)
4  {
5  #if GSMITH_CORRELATED
6      float lerpFactor = pow(1 - NdotV, 10.8649) * (1 - 0.298475 * log(39.4115 - 39.0029 *
           roughness)) + 0.298475 * log(39.4115 - 39.0029 * roughness);
7  #else
8      float lerpFactor = 0.298475f * NdotV * log(39.4115f - 39.0029f * roughness) + (0.385503f -
           0.385503f * NdotV) * log(13.1567f - 12.2848f * roughness);
9  #endif
10
11     // The result is not normalized as we fetch in a cubemap
12     return lerp(N, R, lerpFactor);
13 }
```

Listing 21: Function for computing the dominant direction of the specular microfacet GGX-based specular term with lightprobe.

```
1  // We have a better approximation of the off specular peak
2  // but due to the other approximations we found this one performs better.
3  // N is the normal direction
4  // R is the mirror vector
5  // This approximation works fine for G smith correlated and uncorrelated
6  float3 getSpecularDominantDir(float3 N, float3 R, float roughness)
7  {
8      float smoothness = saturate(1 - roughness);
9      float lerpFactor = smoothness * (sqrt(smoothness) + roughness);
10     // The result is not normalized as we fetch in a cubemap
11     return lerp(N, R, lerpFactor);
12 }
```

Listing 22: Simple function for computing the dominant direction of the specular microfacet GGX-based specular term with lightprobe.

For pure Lambertian surfaces, there is no direction shifting since the BRDF is view independent. However for the retro-reflective Disney diffuse term, the dominant direction of the lobe depends on the view direction. The dominant direction shifts in a non-linear fashion away from it. Applying the same analysis as for the specular term reveals that a simple linear model can capture this behavior relatively accurately, see Listing 23. In *Frostbite* the indirect diffuse lighting is apply during the GBuffer creation. Thus to correctly handle the dominant direction it is necessary to apply it at this step. The differences obtained by accurate dominant direction handling are subtle and we have decided not to

adopt it. Moreover decals will not be supported correctly as highlight in Section 3.3.

```
1  // N is the normal direction
2  // V is the view vector
3  // NdotV is the cosine angle between the view vector and the normal
4  float3 getDiffuseDominantDir(float3 N, float3 V, float NdotV, float roughness)
5  {
6      float a = 1.02341f * roughness - 1.51174f;
7      float b = -0.511705f * roughness + 0.755868f;
8      lerpFactor = saturate((NdotV * a + b) * roughness);
9       // The result is not normalized as we fetch in a cubemap
10     return lerp(N, V, lerpFactor);
11 }
```

Listing 23: Functions for computing the dominant direction of the diffuse retro-reflection Disney lobe with lightprobe.

The code for evaluating the lighting for distant light probe for both specular and diffuse term is shown in Listing 24.
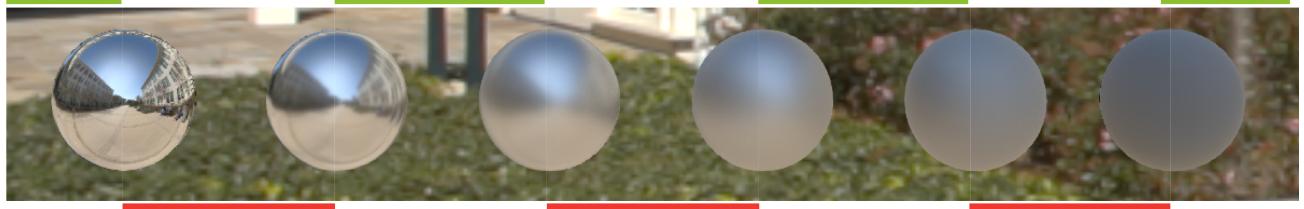
```
1  float3 evaluateIBLDiffuse(...)
2  {
3      float3 dominantN = getDiffuseDominantDir(N, V, NdotV, roughness);
4      float3 diffuseLighting = diffuseLD.Sample(sampler, dominantN);
5
6      float diffF = DFG.SampleLevel(sampler, float2(NdotV, roughness), 0).z;
7
8      return diffuseLighting * diffF;
9  }
10
11 float3 evaluateIBLSpecular(...)
12 {
13     float3 dominantR = getSpecularDominantDir(N, R, NdotV, roughness);
14
15     // Rebuild the function
16     // L . D . ( f0.Gv.(1-Fc) + Gv.Fc ) . cosTheta / (4 . NdotL . NdotV)
17     NdotV = max(NdotV, 0.5f/DFG_TEXTURE_SIZE);
18     float mipLevel = linearRoughnessToMipLevel(linearRoughness, mipCount);
19     float3 preLD = specularLD.SampleLevel(sampler, dominantR, mipLevel).rgb;
20
21     // Sample pre-integrate DFG
22     //   Fc = (1-H.L)^5
23     //   PreIntegratedDFG.r = Gv.(1-Fc)
24     //   PreIntegratedDFG.g = Gv.Fc
25     float2 preDFG = DFG.SampleLevel(sampler, float2(NdotV, roughness), 0).xy;
26
27     //   LD . ( f0.Gv.(1-Fc) + Gv.Fc.f90 )
28     return preLD * (f0 * preDFG.x + f90 * preDFG.y);
29 }
```

Listing 24: Distant light probe evaluation code for both specular and diffuse parts.

Energy conservation: As mentioned, light probes are evaluated for both the specular and diffuse parts of a material in order to get coherent lighting. One could think in this case we would apply the lighting twice, particularly in the case of rough surfaces where both integrations are performed on the hemisphere. But thanks to our modification of the Disney's diffuse BRDF ensuring the energy conservation of material models, the incident energy affects specular and diffuse terms in correct proportions without adding any energy.
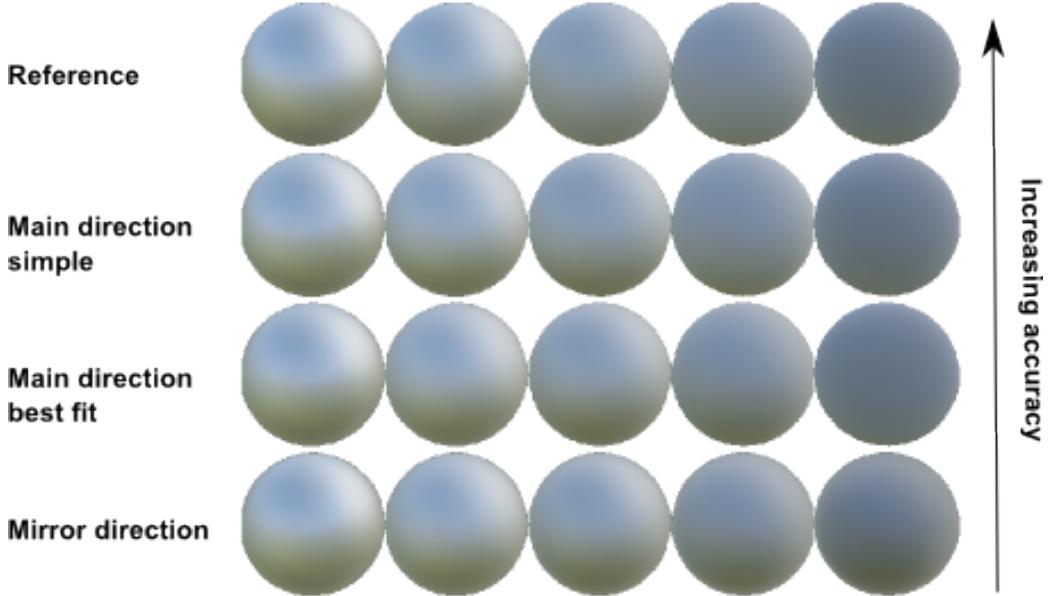
Re-projection: In order to take into account the parallax of the surrounding environment, local light probes use proxy geometry for re-projecting the acquired lighting information. In *Frostbite*, we support two types of re-projection volumes: sphere and oriented box. These volumes are manually placed and

(a) Reference vs. lobe's dominant direction. Differences are mainly visible for rough surfaces.

(b) Close-up of rough surface with different methods, ordered by accuracy.

Figure 59: Comparison between the reference and the lobe's dominant direction (top). The differences are noticeable for high roughness ($\alpha_{\text{lin}} \geq 0.6$). And close-up of a rough surface with different methods ordered by accuracy. All methods other than reference use a pre-integration scheme.

set up by artists to approximate as close as possible the surrounding geometry. At runtime, an object inside a local light probe will compute the intersection between the sampling direction and the proxy geometry and will evaluate the local light probe with a corrected direction based on this intersection [LZ12]. Since the light probe is pre-convolved from center of the volume, correcting the intersection direction can create artifacts for high roughness values. To limit artifacts (sharp discontinuities) we smoothly interpolate the corrected direction to the original direction based on the roughness. Moreover, we provide an offset vector to move the center of the cubemap capture. This allow artists to increase resolution where it matter and allow to get around undesired objects place at the center of the volume.

Our local light probe are only used for the specular part of a material. The code for evaluating the lighting for local light probe for specular term is provided in Appendix F. For the diffuse component, the lighting information comes from our radiosity system, which allows to query the incident lighting through light maps or probe volumes. This diffuse lighting is evaluated in the dominant direction of the retro-reflective lobe.

Distance based roughness: A BRDF lobe describes how the incident lighting is integrated over

the hemisphere. This angular description makes the BRDF footprint distance-dependent. For a given shaded point, the reflection of an object will be sharp when this object is close to the shaded point and will become blurrier and blurrier as it moves away, see Figure 60 and 61. Local light probe proxy geometry allows us to compute the distance between a shaded point and the incident lighting. With this information, we can coarsely approximate the BRDF footprint cast onto the environment. During the evaluation, we modify the BRDF roughness to match a similar footprint and approximating this "distance based roughness", see Listing 25.
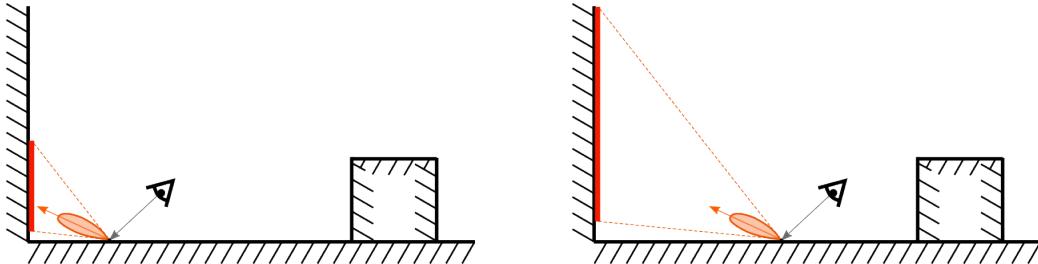

Figure 60: Illustration of the distance based roughness concept. Left: the projection of the BRDF lobe for a point close to a wall will produce a small footprint on it. Right: the projection of the same BRDF lobe but further away will produce a larger footprint on it.

For computing this roughness correction, we compute a cone which bounds the NDF and which is aligned on the dominant BRDF direction. The bounding cone angle θ_{lim} is related to the roughness α by $\theta_{lim} = \mathrm{atan}(\frac{E\alpha^2}{1-E})$, with E the percentage of preserved energy, i.e. if E = 75% the cone will encompass 75% of the NDF volume (detailed plots are available in a Mathematica companion file). The intersection of this cone with the proxy geometry is approximated by a circle of radius R. From this radius R, we can compute a cone starting from the light probe center, from which we deduce the corresponding roughness. This full chain operation allows us to eliminate most of the terms and the "distance-based roughness" can be related to the roughness with this simple formula: $\alpha' = \frac{distanceInteresectionToShadedPoint}{distanceInteresectionToProbeCenter} \alpha$. It is important to note that this is a coarse approximation: in addition to the approximated intersection, we try to use pre-convoloved luminance as scaled footprints. This works decently for low roughness values but becomes inaccurate for high roughness. In order to avoid this issue and since this effect is more visible for low roughness values, we linearly interpolate the "distance based roughness" to the original roughness, based on roughness values. We also found that working with the linearRoughness give better result.

```
1  float computeDistanceBaseRoughness(
2      float distInteresectionToShadedPoint,
3      float distInteresectionToProbeCenter,
4      float linearRoughness)
5  {
6      // To avoid artifacts we clamp to the original linearRoughness
7      // which introduces an acceptable bias and allows conservation
8      // of mirror reflection behavior for a smooth surface.
9      float newLinearRoughness = clamp(distInteresectionToShadedPoint /
           distInteresectionToProbeCenter * linearRoughness, 0, linearRoughness);
10     return lerp(newLinearRoughness, linearRoughness, linearRoughness);
11 }
```

Listing 25: Distant based roughness computation.

As described later in Section 4.9.5, the SSR falls back to local light probes when the SSR pass fails to provide reflection information for a given pixel. Due to the different lighting integrations (SSR properly integrates the BRDF and takes care of the full parallax, while local light probes are pre-integrated from a single point, and parallax is approximated), taking into account this distance based
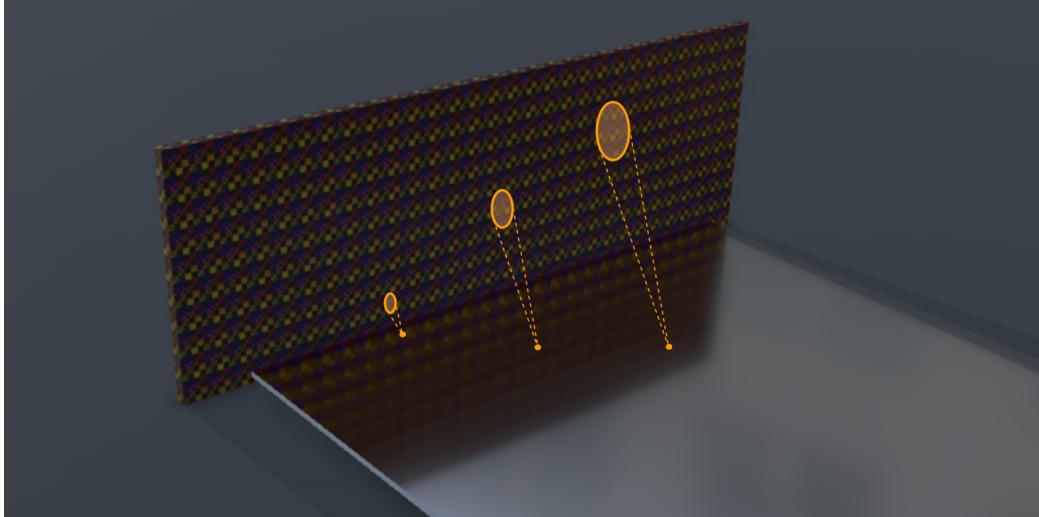
Figure 61: Example of distance-based roughness with a textured wall and a glossy metallic plane. With distance, reflections get blurrier as the BRDF footprint gets wider.

roughness allows to have a match between SSR and our local light probe, resulting in a better transition.

4.9.4 Screen space reflections

Screen-space reflection (SSR) allows capture close to middle range reflections and is really important for small objects/details, as it brings accurate occlusions. Our technique relies on a hierarchical-Z structure which is built on top of the scene depth buffer. This allows us to trace quickly long rays, and use as much as possible of the available scene information. The incident lighting is then integrated against the material BRDF. Our approach is describe by Uludag in "Hi-Z Screen-Space Cone-Traced Reflections" [Ulu14].

SSR and emissive: As seen in Section 4.8, area lights should be composed of two parts:
- An **analytical light** which describes the emission properties and which is integrated with a BRDF.
- An **emissive surface** which describes the actual shape of the emitting light.

The first part is evaluated during the analytical lighting evaluation pass. The second part which is purely visual, can be captured by the SSR pass or the local light probes pass. In this case a light will contribute two times: once during the analytic evaluation, and once during the image based evaluation, as here the capture of emissive shapes will be treated as incident lighting. Unfortunately, there is no easy solution for addressing this issue.

4.9.5 Image based lights composition

Each IBLs type represents a different incident lighting and has their own limitations. To be able to have an object continuously fitting inside its environment we combine all these IBLs in a hierarchical way.

SSR is a good technique for getting correct reflections, but it fails quite often due to its screen space limitations (information limited to the current frustum, and the single depth layer information). It is possible to detect the main fail cases by using various heuristics (proximity to screen border, ray

orientation, intersection type and more). When SSR fails, we fall back to local light probe information. While captured from a single point of view, light probes allow us to re-project the captured information onto their proxy geometry to smoothly fill in any missing information from the SSR pass. Local light probes are placed manually by artists all over levels. Local light probes are hierarchically evaluated, from small to big volumes until the reflection information is fully recovered, allowing artists to nest them and thus locally refine reflection in certain locations. If reflection information is still missing, the distant light probe is evaluated.

Figure 62 gives an overview of this composition and the pseudo code of the different steps is:

```
// Short range reflections
Evaluate SSR
RGB = SSR.rgb
Alpha = SSR.a

// Medium range reflections
While local light probes And Alpha < 1 do
    Evaluate local light probe
    a = saturate(localLightProbe.a - Alpha)
    RGB += localLightProbe.rgb * a
    Alpha = saturate(a + Alpha)

// Large range reflections
If Alpha < 1 Then
    Evaluate distant light probe
    RGB += distantLightProbe.rgb * (1-Alpha)
```
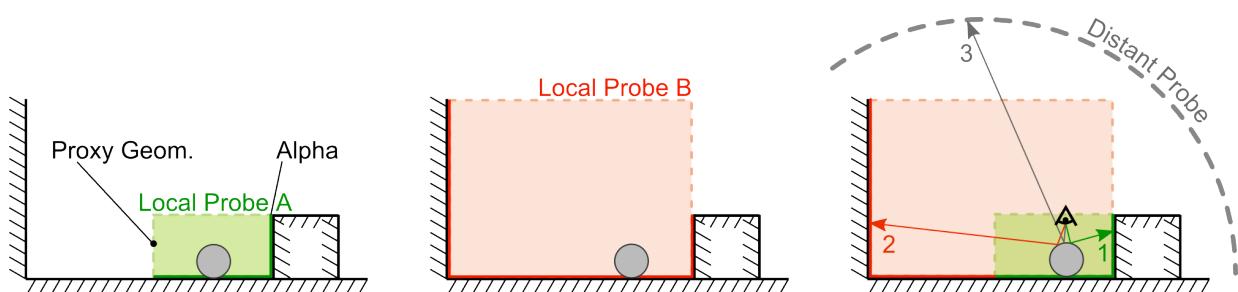

Figure 62: Composition of two local light probes (A and B) and a distant light probe. Both local light probes A and B have a proxy geometry, and acquire a part of the scene (bright green and bright red) in their alpha channel. When a ray intersects a local light probe, if its alpha is one then the ray evaluates the light probe value (1), otherwise the ray keeps going until it reaches another proxy (2), or the distance light probe (3).

The alpha channel of a local light probe indicates if a given pixel belongs to the close surrounding environment. This tagging allows us to reject sky pixels, and enforce fetching the distant light probe. This has several advantages: The distant light probe can contains dynamic elements (such as moving clouds[44]) at a lower cost; The distant light probe is usually higher resolution than the local light probe, making details more crisp. The downside is that resulting incident lighting integration is not totally correct when combined in this way.

---

[44]It is possible to give the impression of moving clouds in a HDRI with the flow map technique [Gue14a]

About the medium range reflections weight calculation. We assume that each local light probe over-lapping contain the same visibility information and occlude the BRDF lobe the exact same way. So if we add 10 overlapping local light probes with visibility 0.1 we should get 0.1. This scheme allow to always have the distant light probe contributing if none of the visibility of light probes of the hierarchy is 1. Note that this algorithm is local light probes order dependent.

Planar reflections: In addition, when planar reflection information is available, it can be applied either in forward on a per-object basis, or in deferred by tagging which objects will receive it. Planar reflections are usually a good alternative to SSR for flat surfaces as it alleviates both the single depth layer constraints and the frustum limitation, making it more robust.

4.10   Shadow and occlusion

4.10.1   Diffuse occlusion

McGuire [McG10] formalizes ambient occlusion and gives it a physical basis. The visibility function is defined as $V(\mathbf{l}) = 1$ if there is an unobstructed line of sight from the surface in direction $\mathbf{l}$ and 0 otherwise. The ambient term of the rendering equation is:

$$L(\mathbf{v}) = \int_{\Omega} f(\mathbf{l}, \mathbf{v}) \ L_a(\mathbf{l}) \ V(\mathbf{l}) \ \langle \mathbf{n} \cdot \mathbf{l} \rangle \ d\mathbf{l} \tag{64}$$

with L_a the ambient lighting. A coarse approximation is to separate out the visibility term from the BRDF and the incident lighting:

$$L(\mathbf{v}) = \left[\pi \int_{\Omega} f(\mathbf{l}, \mathbf{v}) \ L_a(\mathbf{l}) \ d\mathbf{l} \right] \left[\frac{1}{\pi} \int_{\Omega} V(\mathbf{l}) \ \langle \mathbf{n} \cdot \mathbf{l} \rangle \ d\mathbf{l} \right] \tag{65}$$

This separation is only correct if $f_r(\mathbf{l}, \mathbf{v})$ and $L_a(\mathbf{l})$ are constant. This means a Lambertian surface lit by a constant distant light. This approximation is reasonable if both functions are smooth around the sphere. The right term is a scalar factor between 0 and 1, indicating the fractional accessibility of a point. Ambient occlusion is defined as the opposite of this accessibility:

$$AO = 1 - \frac{1}{\pi} \int_{\Omega} V(\mathbf{l}) \ \langle \mathbf{n} \cdot \mathbf{l} \rangle \ d\mathbf{l} \tag{66}$$

In games, for performance reasons, it is common to capture the ambient lighting $L_a(\mathbf{l})$ at various loca-tions and times, and bake it into light maps, cube maps, or spherical harmonics. However this baking lacks the knowledge of dynamic objects in a scene. At runtime, this information is used for shading the scene by interpolating the baked information at shaded points. This implies that shaded points can have a different accessibility than at bake time (e.g. composition static and dynamic objects). To compensate for this, we typically apply an ambient occlusion term to reconstruct the incident lighting[45]. The separability hypothesis between visibility and incident lighting is not accurate but a trade-off made for performances reasons.

4.10.2   Specular occlusion

Ambient occlusion derivations assume Lambertian surfaces, i.e. it is "valid" only for indirect diffuse lighting. What about glossy or specular surfaces, i.e for indirect specular lighting? Not having accessi-bility knowledge when reconstructing indirect specular lighting is far worse than with indirect diffuse.

---

[45]The ambient occlusion term can only darken the baked lighting, not create light. As the baked lighting is often processed without obstructing objects, so with high accessibility, it is reasonable to darken it in a reduced accessibility context during the game.

The high intensity of the lighting and the physically based BRDF causes a lot of visible light leaking, see Figure 63. Moreover, the high frequency of the indirect specular lighting requires a large amount of storage, thus limiting the number of capture points, causing even larger accessibility variations.

Figure 63: An example of a specular light leaking artifact. Look at the specular reflection on the tire and inside the wheel.

Using the ambient occlusion term directly for glossy or specular occlusion to solve this is not ideal. The ambient occlusion represents the opposite of the accessibility of a cosine lobe shape, i.e. a wide lobe over the hemisphere. On the contrary, glossy surfaces exhibit a narrower BRDF lobe shape with decreasing roughness. Figure 64 illustrates the accessibility cone[46] associated with a lobe's BRDF for different roughness values. This is closely related to the way we pre-integrate a cube map for specular lighting.
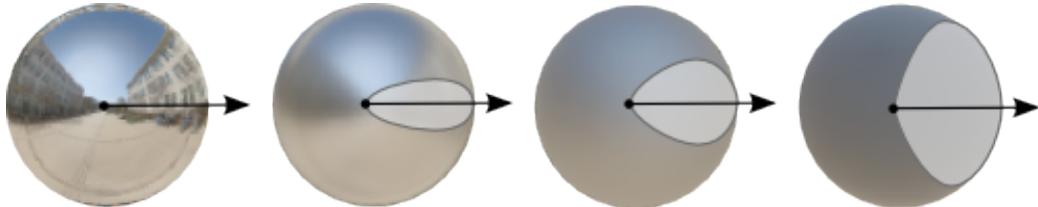
Figure 64: Different accessibility cones based on the shape of the BRDF for different roughness values (represented as a gray cone).

But even with access to such a cone of accessibility, it will not provide the expected result. For instance, for a perfectly smooth surface like a chrome sphere, the accessibility will be tested for a single direction and will give binary result: 0 or 1. Consequently, the chrome sphere using this accessibility information will be black where obstructed. This small example simply highlights that separating the

---

[46]A BRDF lobe is defined on the whole hemisphere but some values far away from the peak are so tiny that we can discard them. This allows us to convert roughness into a cone angle. See Section 4.9.

visibility in the ambient lighting integral is wrong with glossy and specular surface.

However Kozlowski [KK07] highlighted that vast majority of glossy scenes can be approximated to some degree and that approximation with directional ambient occlusion using spherical harmonics is the most effective method.

Gotanda [Got13] has proposed a specular occlusion term derived from the ambient occlusion term. He noticed that ambient occlusion for high intensities does not have the right scale and should take into account the BRDF lobe shape. In *Frostbite*, we have currently adopted a similar approach, empirically adapted to a GGX roughness, see Listing 26. The result was visually pleasant even if uncorrelated to any physical approach. It would benefit from further research to improve its quality. Figure 65 highlights the behavior of this function for an ambient occlusion of 0.5. When the surface is totally rough, the function returns the ambient occlusion term unmodified. For smooth surfaces, it reduces the influence of the ambient occlusion at normal incidence but increases it at grazing angles.

```
float computeSpecOcclusion (float NdotV , float AO , float roughness)
{
    return saturate(pow(NdoV + AO , exp2(-16.0f * roughness - 1.0f)) - 1.0f + AO);
}
```

Listing 26: Function for computing specular occlusion for a given roughness.
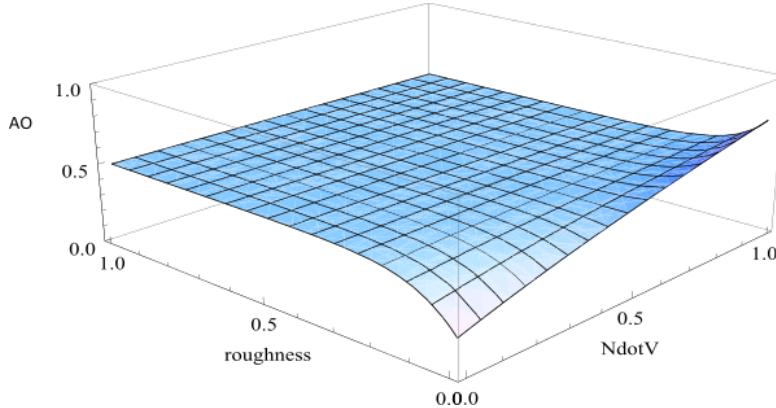
Figure 65: A 3D plot of specular occlusion for an ambient occlusion of 0.5.

Remark: As we see, specular occlusion is not physically based. However the value provided by our screen space reflection is, see Section 4.9.4. It evaluates the ambient rendering equation integral without separating out the visibility. So our SSR term is a better term than specular occlusion in theory. Nevertheless if we do not consider the specular occlusion, the SSR will suffer from light leaking artifacts caused by the indirect cube map in addition to the common screen space artifacts. Thus applying specular occlusion is still useful.

4.10.3 Multi resolution ambient occlusion

In the previous section, we saw that diffuse occlusion and specular occlusion are darkening factors for reconstructing the incident lighting, bypassing the baked light limitation, and reducing light leaking. These factors are not physically based but are desirable for image quality. There are two types of techniques which are used nowadays to produce diffuse occlusion factors:
- **Offline pre-computation**: captures occlusion information from medium to far range.
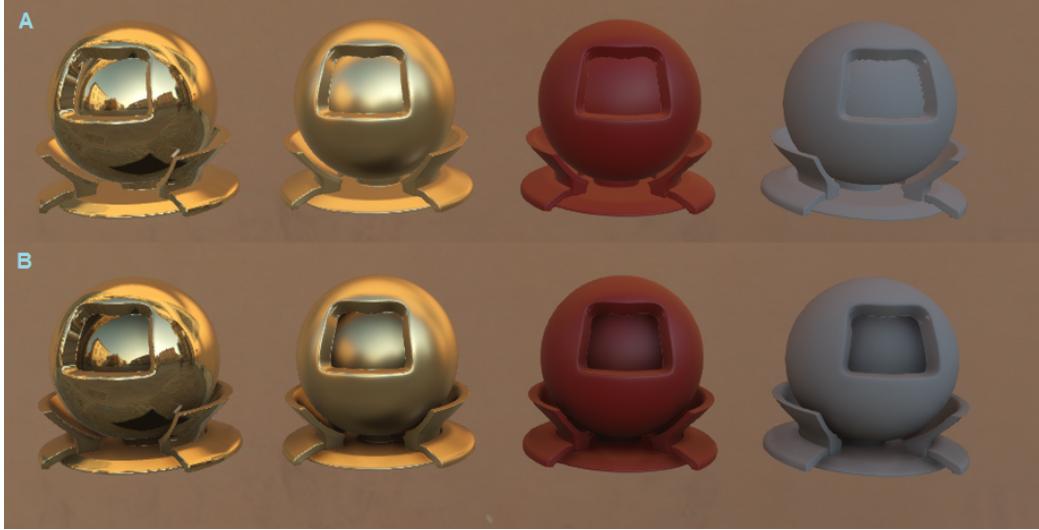
Figure 66: Sample of specular occlusion results: a) without specular occlusion, b) with specular occlusion.

- **Screen space techniques**: captures occlusion from medium distances. This covers lot of different techniques: HBAO, SSAO, VolumetricAO, ambient obscurance, etc.

These techniques allow us to manage medium and large occlusion scale, but neither of them handle small scale occlusions. Creases, cavities and small holes cannot be handled automatically by the engine nor by these medium to large range occlusion techniques. This observation highlights the need for a "multi-resolution occlusion" [Quí12]. We can classify occlusion for both specular and diffuse into three ranges: small, medium, and large.

Small scale occlusion: In *Frostbite* we handle small occlusion by letting artists bake micro-occlusion directly inside textures. Cavities, creases or cracks are also too small to be handled by the shadow map. Thus, the micro-occlusion is applied on both the direct and indirect lighting. We chose to separate micro-occlusion into two parts: diffuse and specular micro-occlusion. They are both derived from the same micro-occlusion information[47].

- **Diffuse micro-occlusion**: The diffuse micro-occlusion is view-independent and is pre-multiplied with the diffuse albedo texture to save instructions, see Figure 67.

- **Specular micro-occlusion**: The specular micro-occlusion is view dependent and relies on a solution provided by Schüler in ShaderX7 [Sch09]. The goal is to modify the Schlick approximation so that any value under a certain threshold has gradually less Fresnel reflectance. Since we know that no real-world material has a value of f_0 lower than 0.02 any values of reflectance lower than this can be assumed to be the result of pre-baked occlusion and thus smoothly decreases the Fresnel reflectance contribution, see Listing 27. Thus specular micro-occlusion is pre-baked inside the reflectance texture.

```
1 f90 = saturate(50.0 * dot(fresnel0, 0.33));
2
3 float3 F_Schlick(in float3 f0, in float f90, in float u)
4 {
5     return f0 + (f90 - f0) * pow(1.f - u, 5.f);
```

---

[47]This micro-occlusion information could be generated with an offline process like a traditional ambient occlusion texture but with a shorter ray range.

```
6  }
```

Listing 27: Christian Schüler solution's for specular micro-occlusion.

Remark: This solution implies a modification of the Fresnel reflectance curve of both f_0 and f_{90} values because it is not really possible to pre-bake the reflection occlusion information without altering the f_0 value. Modifying f_0 means modifying the index of refraction which characterizes the material[48]. Thus what was supposed to affect the lighting only, due to accessibility, is now affecting the properties of the material and sometimes will not even change the Fresnel at grazing angles. We have investigated using a separate dedicated specular micro-occlusion texture which also means an extra parameter to store in the GBuffer. This term was applied to the lighting calculation using a formulation similar to our specular occlusion rather than on the material properties. However, we have found that the perceived result was not worth the cost.

Figure 67: Left: Traditional baked ambient occlusion texture. Middle: Diffuse micro-occlusion. Right: Diffuse micro-occlusion texture pre-multiplied with the albedo.

Medium and large scale occlusion: Medium to large scale occlusion is only applied on indirect lighting. We support HBAO which brings a medium range ambient occlusion but more importantly, provides shadow contact between dynamic objects. We also support a baked medium to large ambient occlusion with two options. Classic offline baking or the dynamically baked term provided by our radiosity system [49]. Baked ambient occlusion needs to be stored in the GBuffer. Game teams can enable their choice of options, but we wanted to prevent over-darkening. To do this, we take the minimum of all applied terms, from medium to large ambient occlusion. This nicely deals with baked ambient occlusion coupled with dynamic ambient occlusion for contact. Micro-occlusion is at a very different scale and it is better to conserve its influence[50]. For specular indirect lighting we convert the resulting ambient occlusion term through the formula provided in the previous section.

To summarise, our various forms of occlusion are:

---

[48] f_0 as we use it is calculated from an air-material interface with the index of refraction from the air and the material.
[49] Our radiosity system provides an ambient occlusion term relying on static data, meaning a simplified scene without dynamic objects. However this term is already included in the lighting data so it should not be applied on indirect diffuse again.
[50] In *Frostbite* as micro-occlusion is baked inside other textures it is not accessible anyway.

Direct diffuse	Diffuse micro-occlusion
Indirect diffuse	Diffuse micro-occlusion, min(bakedAO, HBAO)
Direct specular	Fresnel reflectance modification through specular micro-occlusion
Indirect specular	Fresnel reflectance modification through specular micro-occlusion then computeSpecularOcclusion(NdotV, min(bakedAO, HBAO), roughness)

4.10.4 Shadows

Shadows are an important visual cue for PBR. The literature is quite extensive on this topic and thus we will not discuss it here. Ideally all lights should have shadows but the performance cost makes them often unaffordable on all lights. Usually game teams rely on artists to hide this lack of information. Soft shadows are important for area lights in order to provide the soft look and support the wrapped lighting. Using regular "punctual" shadow maps with an area light would remove the wrapped lighting, see Figure 68. A good solution for area shadow handling is a voxelization technique described by Kasyan in [Kas13]. Cheaper techniques with limited results can be achieved by slightly reducing the size of the geometry during the shadow map calculation [Bre00].
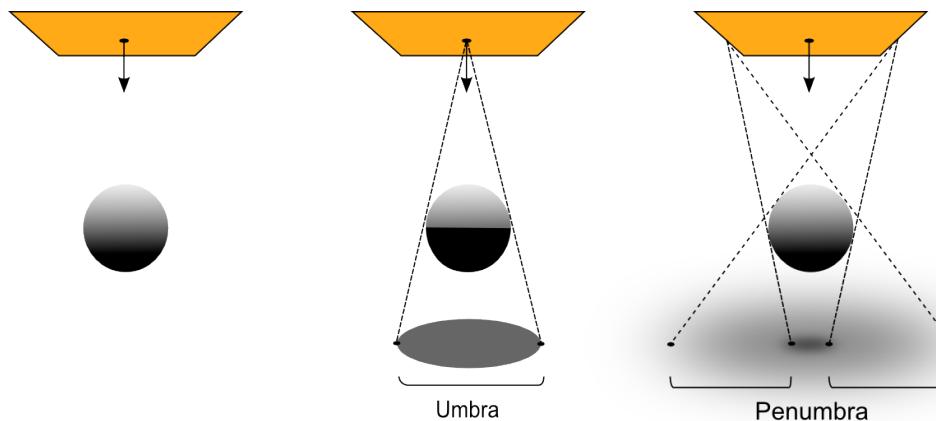

Figure 68: Left: Area light without shadow. Middle: Area light with non-area shadow, all the gradient wrapped lighting is lost. Right: Area light with area shadow.

In *Frostbite*, we currently support point and spot light shadow maps. These kind of shadow maps are used to cover all our light types. To fake area shadows we move the projection center back in the inverse forward direction. This allow the frustum to cover/span the entire area light shape. Heavy blurring and a low shadow resolution are then used to get the soft look, see Figure 69. It would benefit from further research to improve quality.

Remark: It is important to use shadows on all lights as a lighting reference, in order to get a true vision of what should be achieved.

4.11 Deferred / Forward rendering

Frostbite supports a hybrid engine with both forward and deferred tiled renderers [And09]. The literature is quite extensive on these topics thus we will not discuss them here. All our light types, including shadowed lights, are supported using a tiled path. The choice of such an architecture is motivated by performance. The light culling uses tight bounding volumes respecting the light's shape
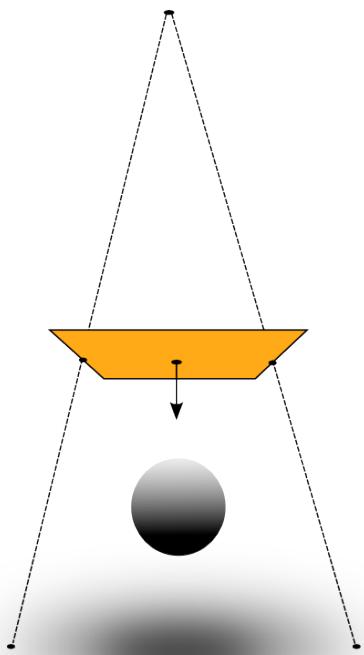
Figure 69: Faking an area shadow by moving the center of projection to a virtual location where it contains the area light's shape, and heavily blurring the shadow map.

to avoid wasting pixel evaluation. We evaluate all lights inside one big loop to avoid consuming bandwidth reading the GBuffer and to increase the precision of calculations.

5 Image

5.1 A Physically Based Camera

All the previous parts have been focused on how light interacts in a scene in a physically based way. Another important consideration to be able to achieve believable result is to consider the entire chain of transformations from the scene luminance until the final pixel value.

5.1.1 Camera settings

Since we deal with photometric units all through the rendering pipeline in *Frostbite*, the light energy reaching the camera is expressed in luminance. The incident light usually covers a large range of values, from a few $cd.m^{-2}$ for dark scenes, to multi-billion $cd.m^{-2}$ when looking at the sun, see Figure 71. These values need to be remapped onto a normalized pixel value for producing the final image visible on a display. In a digital camera, this process is done by "*exposing*" the digital sensor for a certain time and applying post-processes. The purpose of this *exposition* is to maximize the sensor latitude by centring the current light range to halfway between white and dark (middle grey) and setting up the object of interest in the middle of the image range. The full pipeline of transformation from incident light to final pixel value is described by Figure 70.
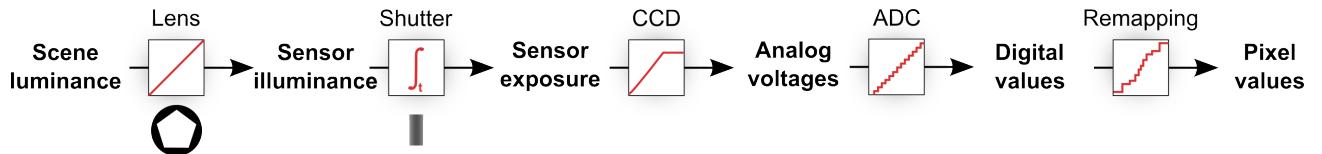
Figure 70: A camera pipeline showing the different transformation steps of the light.

Three main parameters are controllable by an artist for setting the *exposition*:
- **Relative aperture** (N, in f-stops): controls how wide the aperture is opened. Impacts the depth of field.
- **Shutter time** (t, in seconds): controls how long the aperture is opened. Impacts the motion blur.
- **Sensor sensitivity/gain** (S, in ISO): controls how photons are counted/quantized on the digital sensor.
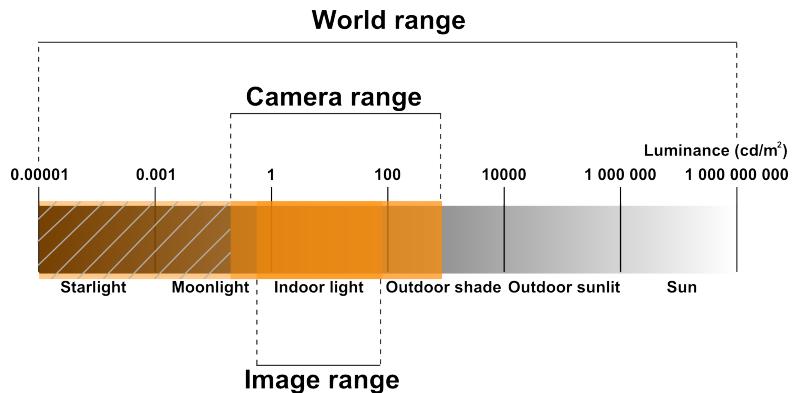
Figure 71: How the incident light range is remapped from world range, to camera range, to image range. The hashed section is the camera range which is cut off due to the finite precision of the camera sensor compared to the low value of the incident lighting.

A given combination of these parameters can be summarized by an *Exposure Value* (EV). An EV is per convention defined for ISO 100, noted EV_{100}, thus the following relationship:

$$EV_{100} = \log_2(\frac{N^2}{t}) - \log_2(\frac{S}{100})$$ (67)

Different combinations of settings can give the same EV, allowing an artist to make different trade-offs between motion blur, depth of field and noise or grain as shown in Figure 72.
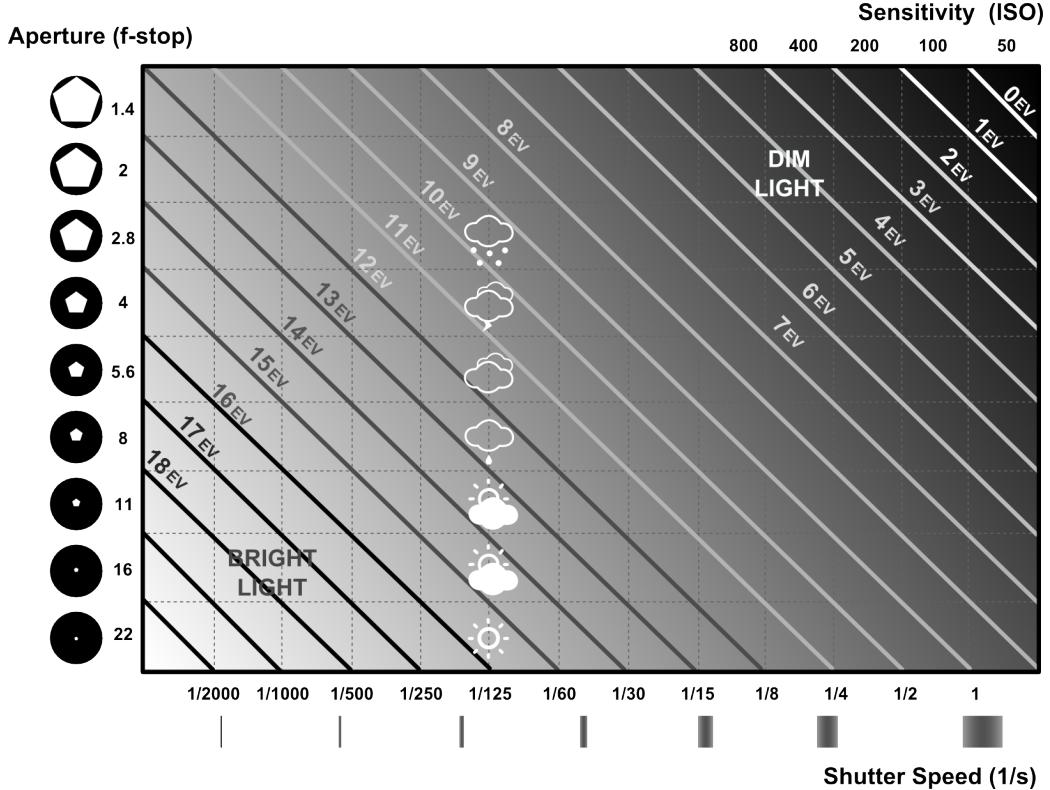

Figure 72: Different configurations of exposition settings. It shows the equivalent settings for different EV values between aperture, shutter speed, and sensitivity.

EV_{100} is also used by artists to describe or measure light intensity as we explained in Section 4.3[51]. In addition to aperture, shutter speed, and sensitivity, artists can apply an exposure compensation (EC, in f-stops) in order to over-expose (i.e. brighten) or under-expose (i.e. darken) the image. This EC is just an offset to the exposure value:

$$EV'_{100} = EV_{100} - EC$$ (68)

The negative sign comes from the fact that EC is in f-stops, meaning an increasing value will increase the aperture size. This goes in the inverse direction as EV, for which increasing the value means reducing the aperture size. Manual exposure settings are convenient for getting the exact look wanted by an artist but can be tedious to set up. Digital cameras offer an auto-exposure mode to ease their manipulation. Finding the correct camera settings (i.e. EV) for a given scene requires some knowledge about the scene luminance. A camera, similar to a spot meter, is able to measure the average incoming luminance and convert it to an EV. This has been detailed in Section 4.3, Equation 11, but reproduced here for readability, with K the reflected-light meter calibration constant (equal to 12.5):

---

[51]However it is not an accurate measure as it is device dependent.

83

| (a) f/1.4, ISO 100 | (b) f/2.8, ISO 100 | (c) f/5.6, ISO 100 |

| (d) f/1.4, ISO 100 | (e) f/2.8, ISO 400 | (f) f/5.6, ISO 1600 |
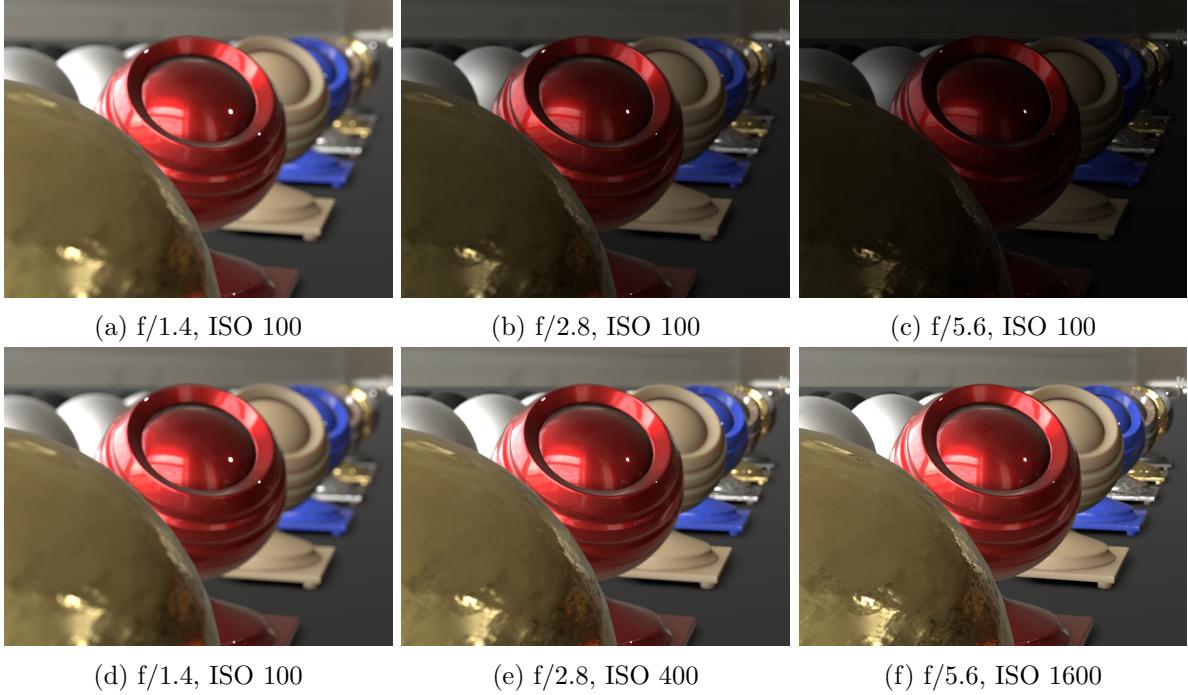
Figure 73: Comparison of settings in manual mode. Top: Only the aperture size changes, the ISO settings remains constant. Due to this the EV_{100} value increases and thus the intensity decreases. Bottom: ISO settings change to compensate for the decreasing size of the aperture in order to achieve the same EV_{100}.

$$EV_{100} = \log_2\left(\frac{L_{\text{avg}}\, S}{K}\right) \tag{69}$$

In games, the usual way to get L_{avg} is to take the average log luminance of all the image pixels. This proves to be a bit unstable with high intensity flickering lights or highlights. A better way is to smooth the value over time, or use a histogram of the luminance to remove extreme values [Vla08]. However these methods suffer from exposing based on pixel luminance multiplied by albedo. This means that if the scene is fully 10% grey or fully 90% grey, it will result in the same color. An alternative is to expose based on the luminance before the multiplication with albedo [Koj+13]. However in practice it is expensive to maintain a separate lighting buffer with white-only luminance. In *Frostbite*, we have adopted the histogram method.

5.1.2 Exposure

The EV quantity must not be confused with the luminous exposure or photometric exposure H (in *lux.s*) which describes the scene luminance reaching the sensor. H is defined by:

$$H = \frac{q\, t}{N^2}\, L \tag{70}$$
$$= t\, E \tag{71}$$

with L the incident luminance and q the lens and vignetting attenuation (a typical value is $q = 0.65$, [Wikc]). The actual value recorded by the sensor will depend on its sensitivity/gain. The ISO standard[52] defines three different ways to relate photometric exposure and sensitivity [Wikc]:

---

[52]Not to be confused with the ISO settings.

- **SOS**: Standard Output Sensitivity
- **SBS**: Saturation Based Sensitivity
- **NSB**: Noise Based Sensitivity

SBS, which is the easiest one to understand, is defined as the maximum possible exposure that does not lead to a clipped or bloomed camera output and relates both terms as:

$$H_{\text{sbs}} = \frac{78}{S_{\text{sbs}}} \tag{72}$$

The factor 78 is chosen so that exposure settings based on a standard light meter and an 18% reflective surface will result in an image with a grey level of $18\%/\sqrt{2} = 12.7\%$ of saturation. The factor $\sqrt{2}$ indicates that there is half a stop of headroom to deal with specular reflections that would appear brighter than a 100% reflecting white surface[53]. By combining Equation 72 and Equation 71, one can determine the maximum luminance L_{max} for saturating the sensor:

$$H_{\text{sbs}} = \frac{78}{S} \tag{73}$$

$$\frac{q\,t}{N^2}\,L_{\text{max}} = \frac{78}{S} \tag{74}$$

$$L_{\text{max}} = \frac{78}{S}\,\frac{N^2}{q\,t} \tag{75}$$

The final pixel value p can then be computed by normalizing the incident luminance L with the maximum luminance L_{max} as shown by Listing 28:

```
float computeEV100(float aperture, float shutterTime, float ISO)
{
    // EV number is defined as:
    //   2^EV_s = N^2 / t    and    EV_s = EV_100 + log2(S/100)
    // This gives
    //   EV_s = log2(N^2 / t)
    //   EV_100 + log2(S/100) = log2(N^2 / t)
    //   EV_100 = log2(N^2 / t) - log2(S/100)
    //   EV_100 = log2(N^2 / t . 100 / S)
    return log2(sqr(aperture) / shutterTime * 100 / ISO);
}

float computeEV100FromAvgLuminance(float avgLuminance)
{
    // We later use the middle gray at 12.7% in order to have
    // a middle gray at 18% with a sqrt(2) room for specular highlights
    // But here we deal with the spot meter measuring the middle gray
    // which is fixed at 12.5 for matching standard camera
    // constructor settings (i.e. calibration constant K = 12.5)
    // Reference: http://en.wikipedia.org/wiki/Film_speed
    return log2(avgLuminance * 100.0f / 12.5f);
}

float convertEV100ToExposure(float EV100)
{
    // Compute the maximum luminance possible with H_sbs sensitivity
    // maxLum = 78 / (  S * q  ) * N^2 / t
    //        = 78 / (  S * q  ) * 2^EV_100
    //        = 78 / (100 * 0.65) * 2^EV_100
    //        = 1.2 * 2^EV
    // Reference: http://en.wikipedia.org/wiki/Film_speed
    float maxLuminance = 1.2f * pow(2.0f, EV100);
```

---

[53]This is similar to the relation between EV and luminance L shown by Equation 71.

```
33      return 1.0f / maxLuminance;
34  }
35
36  // usage with manual settings
37  float EV100           = computeEV100(aperture, shutterTime, ISO);
38  // usage with auto settings
39  float AutoEV100       = computeEV100FromAvgLuminance(Lavg);
40
41  float currentEV       = useAutoExposure ? AutoEV100 : EV100;
42  float exposure        = convertEV100toExposure(currentEV);
43
44  // exposure can then be used later in the shader to scale luminance
45  // if color is decomposed into XYZ
46  ...
47  float exposedLuminance = luminance * exposure;
48  ...
49  // or it can be applied directly on color
50  ...
51  finalColor = color * exposure
52  ...
```

Listing 28: Helper functions for computing the exposure value for remapping the incident lighting L into the camera latitude with the current camera settings: the aperture N, the shutter time t, and the sensitivity S.

This value is then transformed into a digital value by the sensor and the analogue-to-digital converter. The physical properties of the sensor (CCD technology, bit resolution, etc.) will influence slightly how light is quantized and stored. This step outputs linear values which are directly stored in a *RAW* file on digital cameras [Wikh]. Then several transformations are applied to convert this linear data into a "look" pleasing to humans:

- White balancing
- Color grading
- Tone mapping
- Gamma correction

All these transformations can be baked into a single LUT taking the exposed light (H) as input and outputting final pixel values[54]. This LUT, specific to each camera manufacturer and sometimes called *film stock*, is applied automatically on each image when outputting JPEG images from a camera, i.e. when not outputting in RAW. Since the exposed light can still cover a wide range, called "latitude", even after exposure, the LUT needs to be high precision and support values outside of the normalized range [0,1], i.e. "HDR" values. This single transformation can also be broken into different steps for offering more control to artists.

Remark: In order to take into account the full light transformation and the sensor size, one can compute the image plane illuminance E by integrating L over the aperture shape. Remember that H describes the light density received during the entire exposure time. Thus given a photometric exposure H, one can compute the actual amount of energy Q (in $lm.s$) recorded by a sensor cell:

$$Q = H\ A \tag{76}$$

with A the sensor cell size (in m^2) which depends on the total sensor size (e.g. 36mm x 24mm) and sensor resolution (e.g. 5760 x 3240).

5.1.3 Emissive and bloom effects

Amongst the different camera artifacts that game engines try to reproduce (including depth of field, motion blur, noise grain and lens flares), bloom is an interesting one. Bloom is an important effect

---

[54]This LUT can optionally contain the gamma correction depending on whether the hardware supports this final step.

observable through both the human eye and digital cameras conveying high intensity information. The causes of this artifact is multi-fold:

- **High intensity values** that saturate sensor cells and leak to neighboring sensor cells.
- **Inter-reflections** inside the lens barrel.
- **Imperfections** and dust in and on the lens.

The final brightness of a pixel is dependent on the camera settings. Depending on the exposure value, emissive values will or will not produce blooming pixels, i.e. their incident luminance L will be above the maximum luminance $L_{\max}$ of the sensor. This is a problem particularly with time of day where VFX artist want consistent bloom strength for their effect either in the day or in the night [Vai14]. For emissive surfaces, it is handy to provide tools to artists to allow them to control when an emissive surface will bloom or not. For doing so, emissive surfaces can be expressed in terms of exposure compensation in order to adjust their current intensity and ensure it is above the intensity saturation point, and thus forcing the pixels to bloom[55].

```
float3 computeBloomLuminance(   float3 bloomColor,
                                float bloomEC, float currentEV)
{
    // currentEV is the value calculated at the previous frame
    float bloomEV = currentEV + bloomEC;
    // convert to luminance
    // See equation (12) for explanation about converting EV to luminance
    return bloomColor * pow(2, bloomEV-3);
}
```

Listing 29: Function for computing an emissive surface luminance to ensure it blooms at all exposures.

5.1.4 Sunny 16

In order to validate our computations, we have used the "Sunny 16" rule. The Sunny 16 rule is often used by photographers to quickly find camera settings suitable for the lighting conditions. This rule states: *"On a sunny day, set aperture to f/16 and shutter speed to the ISO settings speed for a subject in direct sunlight"*. This means, on a sunny day and with ISO 100 setting, set the aperture to f/16 and the shutter speed to $\frac{1}{100}$ or $\frac{1}{125}$. This rule is extended to other lighting environments as shown by Table 13 [Wikj]. As for the Sunny 16 rule, the ISO and the shutter time have same value. Figure 74 shows an example of such a rule for a sunny day lighting condition.
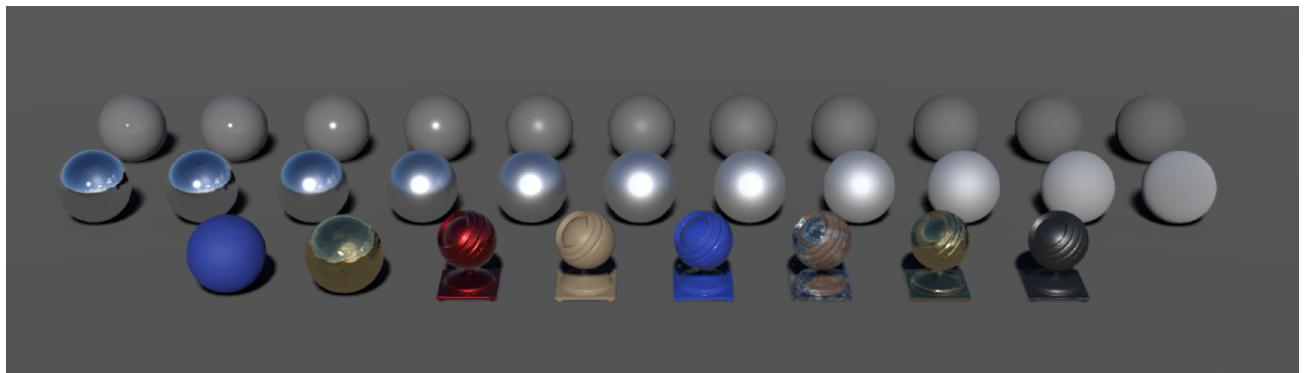

Figure 74: Example of a Sunny 16 test where the aperture is set to f/16, ISO to 100, and shutter speed to 1/125s.

---

[55]This approach could create a feedback loop since exposure is injected back into the scene to make pixels brighter or darker but in practice this does not occur.

Aperture	Lighting conditions	Shadow details
f/22	Snow and sand	Dark with sharp edges
f/16	Sunny	Distinct
f/11	Slightly overcast	Soft around edges
f/8	Overcast	Barely visible
f/5.6	Heavily overcast	No shadows
f/4	Open shade or sunset	No shadows

Table 13: Examples of the Sunny 16 rule extended to other lighting conditions [Wikj]. These values can be used with an ISO 100 and a shutter speed of 1/125s.

5.1.5 Color space

Our pipeline is able to output the final result either in sRGB space, Rec709 space[56] or any other space. See Section 5.1.

A common pitfall in game engines is to approximate the sRGB conversion as a gamma 2.2 curve (often for performance reasons). This gamma 2.2 should be avoided and be replaced with an accurate sRGB conversion formula which is provided in Listing 30. Graphing these two functions (Figure 75) highlights the inaccuracy of the approximation for dark values, which is critical since eyes are very sensitive to variations in the low range. For example Figure 76 shows that the approximation to sRGB can not create true black.

```
float3 approximationSRgbToLinear(in float3 sRGBCol)
{
    return pow(sRGBCol, 2.2);
}

float3 approximationLinearToSRGB(in float3 linearCol)
{
    return pow(linearCol, 1 / 2.2);
}

float3 accurateSRGBToLinear(in float3 sRGBCol)
{
    float3 linearRGBLo  = sRGBCol / 12.92;
    float3 linearRGBHi  = pow((sRGBCol + 0.055) / 1.055, 2.4);
    float3 linearRGB    = (sRGBCol <= 0.04045) ? linearRGBLo : linearRGBHi;
    return linearRGB;
}

float3 accurateLinearToSRGB(in float3 linearCol)
{
    float3 sRGBLo = linearCol * 12.92;
    float3 sRGBHi = (pow(abs(linearCol), 1.0/2.4) * 1.055) - 0.055;
    float3 sRGB   = (linearCol <= 0.0031308) ? sRGBLo : sRGBHi;
    return sRGB;
}
```

Listing 30: Conversion from/to sRGB.

---

[56]sRGB and Rec709 have same primaries but they differ by their approximate gamma curve of exponent: 2.2 for sRGB, 2.4 for Rec709.
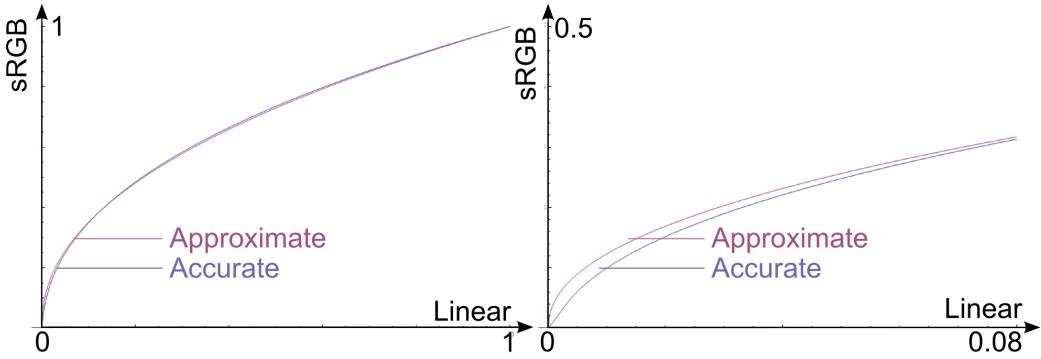
Figure 75: Comparison of linear to sRGB curves. Blue: accurate linear to sRGB conversion. Red: (1/2.2) approximation. On the right the graph is zoomed in to highlight the inaccuracy of the approximation for common dark albedo values.
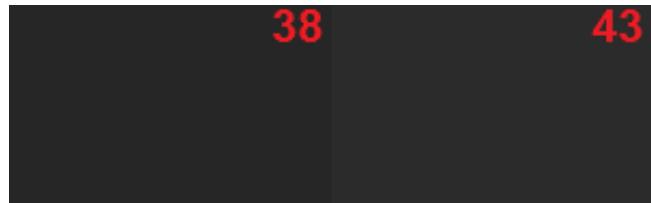

Figure 76: Left: linear RGB 0.02 value converted to sRGB with the accurate conversion. Right: the same conversion but with the approximate curve.

5.2 Manipulation of high values

As seen in Section 5.1, the dynamic range of the real world luminance is huge. It is common for a simple scene to have a dynamic range of 100000:1. We have seen how to deal with such high values for outputting values on the screen, but how manipulating and storing these values before? No particular care of precision issues will cause banding, infinite numbers, and NaNs[57].

For lighting buffer storage there are a few MDR to HDR float formats available. The most common are Float32, Float16, R11F_G11F_10F and RGB9_E5. The two latter formats have no sign bit and the RGB9_E5 format has individual mantissa bits for the RGB components but they all share a single exponent (14 bit float). The numerical precision and limits of these formats are provided in Table 14.

---

[57]Not a Number.

Bit Depth	Bit Sign/Mantissa/Exponent	Largest value	Smallest value*	Decimal digits of precision
32-bit Float	1 bit / 23 bit / 8 bit	3.4028237×10^{38}	1.175494×10^{-38}	7.22
16-bit Float	1 bit / 10 bit / 5 bit	65504	6.10×10^{-5}	3.31
14-bit Float	none / 9 bit / 5 bit	65408**	6.10×10^{-5}	3.16
11-bit Float	none / 6 bit / 5 bit	65024	6.10×10^{-5}	2.5
10-bit Float	none / 5 bit / 5 bit	64512	6.10×10^{-5}	2.32

Table 14: Numeric limits and precision of small float formats [Opeb]. * Smallest in this case meaning the value nearest zero, ignoring denormalization. ** This format use (mantissa) and not (1+mantissa) for the reconstruction.

Float formats: are appropriate for storing lighting information as they are non-linear, similar to the perceived brightness of light. Having less precision in high values is therefore less of a problem than one would expect. The dynamic range of all the small float formats is about 30 stops[58], which covers the range of the 20 stops of the human eye (but only 6.5 stops are available simultaneously [Wike]). Note that there is a small hue shift can occur with the R11F_G11F_10F format as 10-bit floats are less precise than 11-bit floats and the rounding may differ.

Integer formats: are also available, such as RGBM, RGBD, RGBE, LogLUV, LUV[59] [Gue14b] or 10_10_10_2, but they suffer more from banding. The custom formats also require extra ALU and prevent good hardware filtering.

In *Frostbite* we use the following formats for lighting storage:

Lightmap	RGB9_E5. We offer different quality options but we have found that RGB9_E5 is good enough as our lightmap which only stores indirect lighting
HDRI	Float16. We import an HDR format (either OpenEXR (Float16 or Float32) or HDR (RGBE)) and convert them to Float16 in the engine. Care must be taken when importing such a format as conversion to Float16 will result in infinite numbers for values larger than 65504.
Light buffer	Float16. Exposure is pre-applied.

The lighting buffer requires some more attention. Despite the large dynamic range of Float16, extreme values appear often. The maximum value is too low (65504) and the minimum value is too small (6.10×10^{-5}). In addition, accumulating lighting in several passes could easily saturate the buffer and create infinite values. The 30 stops offered by the Float16 parameter can be seen as the exposure range of a camera. To really benefit from this range, we could use the exposure in a way similar to a camera trying to maximize film usage. The desired exposure used for a scene will result in in-focus objects having values about halfway between black and white (middle-grey 18%). Except here we apply the concept at the beginning of the lighting pipeline instead of at the end. The result is similar to neutralizing the lighting of these in-focus objects and thus, more or less, corresponds to retrieving the diffuse albedo[60] and thus limiting numerical issues.

We use the exposure information described in Section 5.1 to pre-expose all our lighting before storing it. We either use a fixed exposure or use the exposure calculated by auto-exposure of the previous frame. There are common pitfalls of using the previous frame's exposure with quick or large camera movements (including teleporting) and as well as the first rendered frame. But it can be perceived as eye adaptation time and thus is acceptable. The pre-exposure is performed inside an epilogue present in all lighting shaders, see Listing 31.

```
float3 epilogueLighting(float3 color, float exposureMultiplier)
{
    return color * exposureMultiplier;
}

// Shaders with lighting
...
outColor0 = epilogueLighting(outColor0, g_exposureMultiplier);
```

---

[58]$\log_2\left(\frac{6.55 \times 10^4}{6.10 \times 10^{-5}}\right)$.

[59]Luminance = 16 bits, Red = 8 bits, Green = 8 bits.

[60]In the comment of [Ree14], Hoffman explains a similar concept.

With pre-exposure there is no need to apply exposure at the end of the pipeline. If required, the pre-exposure can be undone to retrieve the absolute luminance value. In addition to the pre-exposure, we chose to process all our lights in one pass in order to benefit from Float32 precision inside the shader and thus avoiding any loss of information due to read/write from low precision buffer.

5.3   Antialiasing

Aliasing is a common and known issue in real-time rendering. While not new, PBR can increase the amount of aliasing due to the usage of a normalized NDF (which creates high values for low roughnesses), large intensity for incoming lighting, and having "reflections" on everything (i.e. surrounding incident lighting). In this section we will briefly remind the reader the causes of these issues, and the main solutions for addressing them.

The majority of observable aliasing comes from the lack of samples over a pixel's surface for estimating Equation 77. Usually games take a single sample per pixel for estimating the incident lighting reaching a given pixel, mainly for performance reasons.

$$I(\mathbf{v}) = \int_{\text{pixel}} L(\mathbf{v}) \, \mathrm{d}A \tag{77}$$

This sparse sampling makes lighting evaluation heavily dependent on the camera position and surface orientation. For a near-specular surface, a normal can be perfectly aligned with the incident light and the view vector to create a highlight. A slight variation in the view position will prevent this alignment, suddenly removing the highlight. While this flickering is observed in nature (e.g. sun lighting a moving water surface), most of the time it is unrealistic, and really noticeable and distracting for the user. To address this, the literature exposes two main categories of techniques:

- **Supersampling**: Techniques like MSAA (MultiSample Anti-Aliasing), SSAA (SuperSample Anti-Aliasing), and TAA (Temporal Anti-Aliasing) evaluate the incident lighting at multiple points over a pixel, thus catching small variations and thus reducing the visible aliasing. The differences between these techniques depend on how these samples are generated and accumulated. MSAA increases the number of visibility samples, whilst keeping only one shading sample. This will help with geometric discontinuities, but will not help with shading issues. SSAA increases the number of samples spatially over a pixel, allowing a better estimate of the integral 77. While theoretically ideal, its cost prevents SSAA being used in practice. Alternatively, TAA techniques increase the number of samples per pixel over time. This spreads the cost over multiples frames whilst still maintaining a high number of samples per pixel. Due to the temporal nature of TAA techniques, previous samples need to be re-projected from the previous to the current frame when the camera or objects move. Several heuristics are usually used for rejecting samples which contain lighting variations and for handling transparent surfaces, which can not be re-projected. For more details about TAA techniques, see [Sou13; Kar14].

- **Pre-Filtering**: Techniques like Toksvig [Tok05; Han+07], LEAN [OB10], and LEADR [Dup+13] try to keep the number of samples low (ideally one sample per pixel) by transforming macro-geometric (curvature) and meso-geometric variations (normal maps, displacement maps), encompassed by the pixel footprint, into material properties (i.e. statistical micro-geometric variations). This transfer allows for an easier and faster evaluation of all the interactions happening

inside the pixel footprint. Techniques like Toksvig and LEAN focus on the filtering of normal maps [BN12], while LEADR addresses the filtering of displacement maps. Other works approximate the macro-geometric filtering by transferring curvature into material properties. When using deferred rendering this macro-geometric filtering can happen during the G-Buffer pass, as proposed by Baker and Hill [HB12], or as a post-process, as proposed by Mittring [Mit12] and Schulz [Sch14].

Both of these categories of techniques have some limitations which makes combining them a good solution. For instance, pre-filtering techniques do not take macro-geometric discontinuities, like silhouettes, into account. Supersampling techniques are able to solve this type of issues. Technically, the supersampling techniques could be standalone but they could require a lot of samples to converge towards a stable pixel value. Combining both approaches is able to handle most cases efficiently. In addition, a multitude of post-process techniques exist [Jim+11], which try to enhance the image by reducing aliased edges. While they can provide cleaner images, they do not solve the actual issue, but just try to minimize it.

In *Frostbite*, we support various types of anti-aliasing techniques: FXAA, SMAAT1x, SMAAT2x [Jim+12], and MSAA. For normal map pre-filtering, we adopt the approach described by Neubelt and Pettineo in [NP13]. We calculate the NDF based on the per-texel normal, then we combine it with the current roughness to produce a new "*effective*" roughness. In addition to Neubelt and Pettineo, who store the new roughness inside the various levels of a roughness map, we chose to also offer the possibility to apply this process on-the-fly inside the shader before any lighting occurs (for a deferred material model this is applied before storing the roughness in the GBuffer layout). Our motivation is to be able to correctly handle the filtering of normal composition (e.g. with normal maps, details map, or procedural normals) happening inside shaders, Listing 22. Moreover, this allows us to decouple the normal map and the roughness map allowing different resolution and texture reusability. To help with this process, we store an additional average normal length alongside the normal map when we process the texture off-line[61]. The NDF manipulation and computation is handled transparently for artists.

```
float adjustRoughness(float inputRoughness, float avgNormalLength)
{
    // Based on The Order: 1886 SIGGRAPH course notes implementation
    if (avgNormalLength < 1.0f)
    {
        float avgNormLen2 = avgNormalLength * avgNormalLength;
        float kappa = (3 * avgNormalLength - avgNormalLength * avgNormLen2) /
                      (1 - avgNormLen2);
        float variance = 1.0f / (2.0 * kappa);
        return sqrt(inputRoughness * inputRoughness + variance);
    }
    return (inputRoughness);
}

void packGbufferData(PBRMaterialRootData data,
                     out GBufferData gbData, in float opacity)
{
    data.roughness = adjustRoughness(data.roughness, data.normalLength);
    gbData.GBuffer0 = float4(data.worldNormal, data.roughness);
    (...)
}
```

Listing 32: Roughness adjustment based on average normal length before storage in the GBuffer

---

[61]As for LEAN, we should ideally store the square of average normal length and the average squared normal length in order to reconstruct the normal variance linearly. However, this would require two channels instead of one. This is a small accuracy trade-off as the introduced error is contained to interpolation between two mip levels. The compression format is also important as it can create some quantization artifacts.

Remark: In addition to removing flickering specular highlights, all these anti-aliasing techniques help to recover the correct highlight shape due to the micro/meso/macro-variations of the geometry underlying the pixel footprint. The pre-filtered techniques are really efficient for doing this[62]. However, most of the normal map filtering techniques assume a(n) (anisotropic) Gaussian distribution of the normals underlying a pixel footprint. At the same time these techniques assume a material with a Gaussian NDF, making it possible to combine both normal distributions in an analytic fashion (since the convolution of two spherical Gaussian is a spherical Gaussian). However for materials with a GGX NDF, this hypothesis is no longer valid. While the convolution between a GGX distribution and a spherical Gaussian is not analytic, the result can be reasonably approximated by a GGX distribution with a pre-computed table.

---

[62]Without this, a mirror-like bumpy surface would look like a flat mirror at far distances due to the averaging of all normals, instead of having the appearance of rough surface

6   Transition to PBR

When considering moving an engine to PBR, both the technical side and the artistic side need to be taken into account. From the beginning, we have trained technical artists from several game teams in parallel to the technology development.

Moving an engine already used in production for multiple titles to PBR has not been a simple task. A large number of assets have already been authored and artists' knowledge relies on obsolete practices. An early version of the PBR engine has been developed in a branch separate from production. The first step was to develop tools to be able to train the technical artists. For this, we made a simple in-engine object viewer capable of lighting objects with a single distant light probe[63]. In addition to the distant light probe, this viewer contained a sun light sun calibrated to π so that an object lit by the sun would return the diffuse albedo[64]. With the help of this tool, we have been able to start training and producing assets while developing the other PBR features.

The second step has been to maintain a PBR and a non-PBR version of the renderer running inside the same engine with the help of the material system described in Section 3.2. This was important to be able to move the PBR engine into the production branch. In order to be able to launch already created levels with decent visuals we developed automatic conversions:

- **For material parameters**: Fortunately, the non-PBR *Frostbite* engine was already using smoothness and was already separating metallic surfaces with a specific *materialId*. It has been possible to convert parameters inside shaders on the fly to get approximate PBR textures. Our artists have also tried to convert textures manually to improve the quality of the conversion. Our experience with these conversion methods is that artists can get good result but each texture needs a specific treatment. Unfortunately, there is no automatic way to convert all textures with good quality.
- **For lights**: We have found lights to be extremely hard to convert from arbitrary units to physical based units. Most of the non-PBR lights are tweaked for specific lighting conditions and have relative ratios. Thus several parts of our already existing levels were not lit correctly.

The third step was to evangelize PBR to game teams. The trained technical artists from various game teams were key for this acceptance. All the *Frostbite* rendering team has progressively switched to the PBR version and dropped the support of the non-PBR renderer. We have also developed support for our base material model inside external tools used by the game teams like Mari, Substance, and Marmoset to ease the transition.

Finally, to help artists during the transition period, we have developed validation view modes for checking the range material textures or lighting illuminance, see Figure 77.

---

[63]At the time, no public tools (Marmoset, Substance...) or engine (UE4, Unity...) were available.
[64]At the time we were lacking knowledge on the units of lighting and good HDRI acquisition.
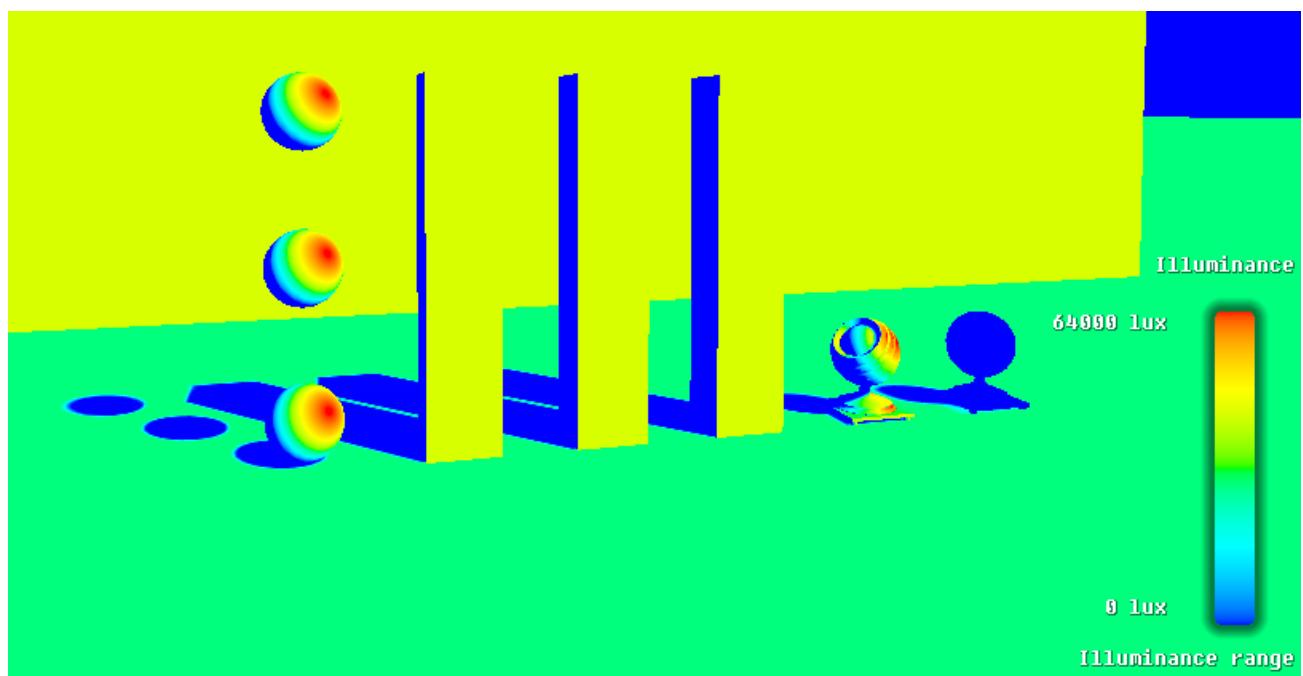
Figure 77: The illumination debug view mode showing the intensity of the incident illuminance. This mode is used by lighting artists to validate their lighting rig.

Acknowledgments

Version

Following is listing the correction done for the version 2.0 of this document.

- Section 3.2.1 - Corrected wrong statement for describing the micro-specular occlusion of the Reflectance parameters: The lower part of this attribute defines a micro-specular occlusion term used for both dielectric and metal materials.. Description of BaseColor and Reflectance parameters have been updated.
- Section 3.2.1 - Removed reference on Alex Fry work of normal encoding as it has not been done.
- Section 4.2 - Updated the description of color temperature for artificial lights sources. Including the concept of color correlated temperature (CCT).
- Section 4.4 - Clarified what is lightColor in Listing 4
- Section 4.5 - Clarified what is lightColor in Listing 5
- Section 4.6 - Updated and explained the computation of the Sun solid angle and the estimated illuminance at Earth surface.
- Section 4.7.2.2 - Added comment in Listing 7: FormFactor equation include a invPi that needs to be canceled out (with Pi) in the sphere and disk area light evaluation
- Section 4.7.2.2 - Clarified in which case the diffuse sphere area formula is exact above the horizon
- Section 4.7.2.3 - Clarified in which case the diffuse disk area formula is exact above the horizon
- Section 4.7.4 - Correct listing 15. getDiffuseDominantDir parameter N is float3
- Section 4.7.5 - Correct listing 16. getSpecularDominantDirArea parameters N and R are float3
- Section 4.9.2 - Corrected the PDF of the specular BRDF and equations from 48 to 60. They had missing components or mistakes. The code was correct.
- Section 4.9.3 - Correct listing 21/22/23. getSpecularDominantDir parameters N and R are float3. getDiffuseDominantDir parameters N and V are float3
- Section 4.9.5 - Added and update comment about reflection composition: The composition weight computation for medium range reflections was causing darkening if several local light probes were overlapping. The previous algorithm was considering that each local light probes visibility was covering a different part of the BRDF lobe (having 10 overlapping local light probes of 0.1 visibility result in 1.0). The new algorithm considers that it covers the same part of the BRDF lobe (Adding 10 overlapping local light probes of 0.1 visibility result in 0.1).
- Section 4.10.2 - Corrected listing 26. Roughness and smoothness were inverted. The listing have been updated and an improve formula have been provided. Figure 65 has been updated accordingly.
- Section 4.10.2 - Added a reference to "Is Accurate Occlusion of Glossy Reflections Necessary" paper.
- Section 5.2 - Table 14: Fixed wrong largest value for 14-bit float format. 16-bit float format is a standard floating point format with implied 1 on the mantissa. Max exponent for 16-bit float is 15 (not 16, because 16 is reserved for INF). Largest value is $(1+m)^{maxExp} = (1+\frac{1023}{1024})*2^{15} = 65504$. Whereas 14-bit float format has no leading 1, but a max exponent of 16. Largest value is $m*2^{maxExp} = (\frac{511}{512})^{16} = 65408$. 10-bit and 11-bit float format follow same rules as 16-bit float format.

Following is listing the correction done for the version 3.0 of this document.

- Section 5.1.1 - Fix equation 67

Bibliography

[Ana]   D. Anastasia. *Luminance HDR*. URL: http://qtpfsgui.sourceforge.net/.

[And09]   J. Andersson. "Parallel Graphics in Frostbite - Current & Future". In: *Beyond Programmable Shading (Parts I and II), ACM SIGGRAPH 2009 Courses*. SIGGRAPH '09. New Orleans, Louisiana: ACM, 2009, 7:1–7:312. DOI: 10.1145/1667239.1667246. URL: http://s09.idav.ucdavis.edu/.

[Arv94]   J. Arvo. "The Irradiance Jacobian for Partially Occluded Polyhedral Sources". In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '94. New York, NY, USA: ACM, 1994, pp. 343–350. ISBN: 0-89791-667-0. DOI: 10.1145/192161.192250. URL: http://www.graphics.cornell.edu/pubs/1994/Arv94.html.

[AS00]   M. Ashikhmin and P. Shirley. "An Anisotropic Phong BRDF Model". In: *Journal of Graphics Tools* 5.2 (2000), pp. 25–32. DOI: 10.1080/10867651.2000.10487522. URL: http://www.cs.utah.edu/~michael/brdfs/.

[Ash14]   I. Ashdown. *Personal Communication*. 2014.

[Ash98]   I. Ashdown. *Parsing The IESNA LM-63 Photometric Data File*. 1998. URL: http://lumen.iee.put.poznan.pl/kw/iesna.txt.

[BH12]   C. Barré-Brisebois and S. Hill. *Blending in Detail*. 2012. URL: http://colinbarrebrisebois.com/2012/07/17/blending-normal-maps/.

[BN12]   É. Bruneton and F. Neyret. "A Survey of Non-linear Pre-filtering Methods for Efficient and Accurate Surface Shading". In: *IEEE Trans. Vis. Comput. Graph.* 18.2 (2012), pp. 242–260. URL: http://maverick.inria.fr/Publications/2011/BN11/.

[Bre00]   R. Bredow. "Fur in Stuart Little". In: *Advanced Renderman 2: To RI_INFINITY and Beyond, ACM SIGGRAPH 2000 Courses*. SIGGRAPH '00. New Orleans, Louisiana: ACM, 2000. URL: http://www.renderman.org/RMR/Publications/.

[Bur12]   B. Burley. "Physically Based Shading at Disney". In: *Physically Based Shading in Film and Game Production, ACM SIGGRAPH 2012 Courses*. SIGGRAPH '12. Los Angeles, California: ACM, 2012, 10:1–7. ISBN: 978-1-4503-1678-1. DOI: 10.1145/2343483.2343493. URL: http://selfshadow.com/publications/s2012-shading-course/.

[Cañ14]   J. Cañada. *Personal Communication*. 2014.

[CH05]   G. Coombe and M. Harris. "Global Illumination Using Progressive Refinement Radiosity". In: *GPU Gems 2*. Ed. by M. Pharr. Addison-Wesley, 2005, pp. 635–647. URL: http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter39.html.

[Cof10]   C. Coffin. "SPU Based Deferred Shading in Battlefield 3 for Playstation 3". In: Game Developers Conference. 2010. URL: http://www.slideshare.net/DICEStudio/spubased-deferredshading-in-battlefield-3-for-playstation-3.

[Con14]   byHeart Consultants Limited. *Helios32 Website*. 1999-2014. URL: `http://www.helios32.com/resources.htm`.

[CPF10]   M. Colbert, S. Premože, and G. François. "Importance Sampling for Production Rendering". In: *ACM SIGGRAPH 2010 Courses*. SIGGRAPH '10. Los Angeles, California: ACM, 2010. URL: `https://sites.google.com/site/isrendering/`.

[CT82]   R. L. Cook and K. E. Torrance. "A Reflectance Model for Computer Graphics". In: *ACM Trans. Graph.* 1.1 (Jan. 1982), pp. 7–24. ISSN: 0730-0301. DOI: `10.1145/357290.357293`. URL: `http://graphics.pixar.com/library/ReflectanceModel/`.

[Dev07]   F. Devernay. *C/C++ Minpack*. 2007. URL: `http://devernay.free.fr/hacks/cminpack/`.

[DM97]   P. E. Debevec and J. Malik. "Recovering High Dynamic Range Radiance Maps from Photographs". In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 369–378. ISBN: 0-89791-896-7. DOI: `10.1145/258734.258884`. URL: `http://www.pauldebevec.com/Research/HDR/`.

[Dro13]   M. Drobot. "Lighting of Killzone: Shadow Fall". In: Digital Dragons. 2013. URL: `http://www.guerrilla-games.com/publications/`.

[Dro14a]   M. Drobot. *Personal Communication*. 2014.

[Dro14b]   M. Drobot. "Physically Based Area Lights". In: *GPU Pro 5*. Ed. by W. Engel. CRC Press, 2014, pp. 67–100.

[Dro14c]   M. Drobot. *ShaderFastLibs*. 2014. URL: `https://github.com/michaldrobot/ShaderFastLibs`.

[Dup+13]   J. Dupuy, E. Heitz, J.-C. Iehl, P. Poulin, F. Neyret, and V. Ostromoukhov. "Linear Efficient Antialiased Displacement and Reflectance Mapping". In: *ACM Transactions on Graphics* 32.6 (Nov. 2013), Article No. 211. DOI: `10.1145/2508363.2508422`. URL: `http://hal.inria.fr/hal-00858220`.

[Gd08]   L. Gritz and E. d'Eon. "The Importance of Being Linear". In: *GPU Gems 3*. Ed. by H. Nguyen. Addison-Wesley, 2008, pp. 529–542. URL: `http://http.developer.nvidia.com/GPUGems3/gpugems3_ch24.html`.

[Gen]   I. Generator. *IES Generator*. URL: `http://www.tom-schuelke.com/ies-gen3.exe`.

[GKD07]   P. Green, J. Kautz, and F. Durand. "Efficient Reflectance and Visibility Approximations for Environment Map Rendering". In: *Computer Graphics Forum (Proc. EUROGRAPHICS)* 26.3 (2007), pp. 495–502. URL: `http://people.csail.mit.edu/green/`.

[Got13]   Y. Gotanda. "Real-time Physically Based Rendering". In: CEDEC 2013. 2013. URL: `http://research.tri-ace.com`.

[Got14]   Y. Gotanda. "Designing a Reflectance Model for New Consoles". In: *Physically Based Shading in Theory and Practice, ACM SIGGRAPH 2014 Courses*. SIGGRAPH '14. Vancouver, Canada: ACM, 2014, 23:1–23:8. ISBN: 978-1-4503-2962-0. DOI: `10.1145/2614028.2615431`. URL: `http://selfshadow.com/publications/s2014-shading-course/`.

[Gue14a]   K. Guerrette. "Moving the Heavens". In: Game Developers Conference. 2014. URL: `http://www.gdcvault.com/play/1020146/Moving-the-Heavens-An-Artistic`.

[Gue14b]   J. Guertault. *Artist-Friendly HDR With Exposure Values*. 2014. URL: `http://lousodrome.net/blog/light/2013/05/26/gamma-correct-and-hdr-rendering-in-a-32-bits-buffer`.

[Han+07]   C. Han, B. Sun, R. Ramamoorthi, and E. Grinspun. "Frequency Domain Normal Map Filtering". In: *ACM SIGGRAPH 2007 Papers*. SIGGRAPH '07. San Diego, California: ACM, 2007. DOI: 10.1145/1275808.1276412. URL: http://www.cs.columbia.edu/cg/normalmap/.

[HB12]   S. Hill and D. Baker. "Rock-Solid Shading: Image Stability Without Sacrificing Detail". In: *Advances in Real-time Rendering in Games Part II, ACM SIGGRAPH 2012 Courses*. SIGGRAPH '12. Los Angeles, California: ACM, 2012. URL: http://advances.realtimerendering.com/s2012/.

[Hei14]   E. Heitz. "Understanding the Masking-Shadowing Function in Microfacet-Based BRDFs". In: *Journal of Computer Graphics Techniques (JCGT)* 3.2 (June 2014), pp. 32–91. ISSN: 2331-7418. URL: http://jcgt.org/published/0003/02/03/.

[Hil10]   S. Hill. "Rendering With Conviction". In: Game Developers Conference. 2010. URL: http://selfshadow.com/publications/.

[HSM10a]   J. R. Howell, R. Siegel, and M. P. Mengüç. 2010. URL: http://www.thermalradiation.net/indexCat.html.

[HSM10b]   J. R. Howell, R. Siegel, and M. P. Mengüç. 2010. URL: http://www.thermalradiation.net/sectionb/B-13.html.

[Iwa13]   M. Iwanicki. "Lighting Technology of the Last of Us". In: *ACM SIGGRAPH 2013 Talks*. SIGGRAPH '13. Anaheim, California: ACM, 2013, 20:1–20:1. ISBN: 978-1-4503-2344-4. DOI: 10.1145/2504459.2504484. URL: http://miciwan.com/.

[Jak+14]   W. Jakob, E. d'Eon, O. Jakob, and S. Marschner. "A Comprehensive Framework for Rendering Layered Materials". In: *ACM Trans. Graph.* 33.4 (July 2014), 118:1–118:14. ISSN: 0730-0301. DOI: 10.1145/2601097.2601139. URL: http://www.cs.cornell.edu/projects/layered-sg14/.

[Jak10]   W. Jakob. *Mitsuba renderer*. 2010. URL: http://www.mitsuba-renderer.org.

[Jen+01]   H. W. Jensen, S. R. Marschner, M. Levoy, and P. Hanrahan. "A Practical Model for Subsurface Light Transport". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 511–518. ISBN: 1-58113-374-X. DOI: 10.1145/383259.383319. URL: http://graphics.ucsd.edu/~henrik/papers/bssrdf/.

[Jim+11]   J. Jimenez, D. Gutierrez, J. Yang, A. Reshetov, P. Demoreuille, T. Berghoff, C. Perthuis, H. Yu, M. McGuire, T. Lottes, H. Malan, E. Persson, D. Andreev, and T. Sousa. "Filtering Approaches for Real-time Anti-aliasing". In: *ACM SIGGRAPH 2011 Courses*. SIGGRAPH '11. Vancouver, British Columbia, Canada: ACM, 2011, 6:1–6:329. ISBN: 978-1-4503-0967-7. DOI: 10.1145/2037636.2037642. URL: http://iryoku.com/aacourse/.

[Jim+12]   J. Jimenez, J. I. Echevarria, T. Sousa, and D. Gutierrez. "SMAA: Enhanced Morphological Antialiasing". In: *Computer Graphics Forum (Proc. EUROGRAPHICS 2012)* 31.2 (2012). URL: http://www.iryoku.com/smaa/.

[Kar13]   B. Karis. "Real Shading in Unreal Engine 4". In: *Physically Based Shading in Theory and Practice, ACM SIGGRAPH 2013 Courses*. SIGGRAPH '13. Anaheim, California: ACM, 2013, 22:1–22:8. ISBN: 978-1-4503-2339-0. DOI: 10.1145/2504435.2504457. URL: http://selfshadow.com/publications/s2013-shading-course/.

[Kar14]   B. Karis. In: *Advances in Real-time Rendering in Games Part I, ACM SIGGRAPH 2014 Courses*. SIGGRAPH '14. Vancouver, Canada: ACM, 2014, 10:1–10:1. ISBN: 978-1-4503-2962-0. DOI: 10.1145/2614028.2615455. URL: http://advances.realtimerendering.com/s2014/index.html.

[Kas13]   N. Kasyan. "Playing with Real-Time Shadows". In: *Efficient Real-time Shadows, ACM SIGGRAPH 2013 Courses*. SIGGRAPH '13. Anaheim, California: ACM, 2013, 18:1–18:54. ISBN: 978-1-4503-2339-0. DOI: 10.1145/2504435.2504453. URL: http://www.realtimeshadows.com/?q=node/16.

[KC08]   J. Křivánek and M. Colbert. "Real-time Shading with Filtered Importance Sampling". In: *Computer Graphics Forum* 27.4 (2008). Eurographics Symposium on Rendering, EGSR '08, pp. 1147–1154. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2008.01252.x. URL: http://dcgi.felk.cvut.cz/publications/2008/krivanek-cgf-rts.

[KK07]   O. Kozlowski and J. Kautz. "Is Accurate Occlusion of Glossy Reflections Necessary ?" In: *Proceedings of Symposium on Applied Perception in Graphics and Visualization 2007* (July 2007), pp. 91–98. URL: http://web4.cs.ucl.ac.uk/staff/j.kautz/publications/glossyAPGV07.pdf.

[Koj+13]   H. Kojima, H. Sasaki, M. Suzuki, and J. Tago. "Photorealism Through the Eyes of a FOX: The Core of Metal Gear Solid Ground Zeroes". In: Game Developers Conference. 2013. URL: http://www.gdcvault.com/play/1018086/Photorealism-Through-the-Eyes-of.

[Kři06]   J. Křivánek. *Radiant Flux Emitted by a VRML SpotLight-like Luminaire*. 2006. URL: http://cgg.mff.cuni.cz/~jaroslav/.

[Leg]   A. Legotin. *IESViewer*. URL: http://www.photometricviewer.com/.

[LH13]   S. Lagarde and L. Harduin. "The Art and Rendering of Remember Me". In: Game Developers Conference Europe. 2013. URL: http://seblagarde.wordpress.com/2013/08/22/gdceurope-2013-talk-the-art-and-rendering-of-remember-me/.

[LZ12]   S. Lagarde and A. Zanuttini. "Local Image-based Lighting with Parallax-corrected Cubemaps". In: *ACM SIGGRAPH 2012 Talks*. SIGGRAPH '12. Los Angeles, California: ACM, 2012, 36:1–36:1. ISBN: 978-1-4503-1683-5. DOI: 10.1145/2343045.2343094. URL: http://seblagarde.wordpress.com/2012/11/28/siggraph-2012-talk/.

[Mad11]   T. Madams. *Improved light attenuation*. 2011. URL: http://imdoingitwrong.wordpress.com/2011/02/10/improved-light-attenuation/.

[Mar14]   I. Martínez. *Radiative view factors*. 1995-2014. URL: http://webserver.dmt.upm.es/~isidoro/tc3/Radiation%5C%20View%5C%20factors.pdf.

[Mat14]   R. J. Mathar. *Solid Angle of a Rectangular Plate*. 2014. URL: http://www.mpia-hd.mpg.de/~mathar/public/index.html.

[McG10]   M. McGuire. "Ambient Occlusion Volumes". In: *Proceedings of High Performance Graphics 2010*. Saarbrucken, Germany, June 2010. URL: http://graphics.cs.williams.edu/papers/AOVHPG10.

[McN]   McNeel. *Working with HDRIs*. URL: http://wiki.mcneel.com/accurender/nxt/documentation/beyond/hdri.

[MER]   MERL. *MERL BRDF Database*. URL: http://www.merl.com/brdf/.

[Mit12]   M. Mittring. "The Technology Behind the "Unreal Engine 4 Elemental demo"". In: *Advances in Real-time Rendering in Games Part II, ACM SIGGRAPH 2012 Courses*. SIGGRAPH '12. Los Angeles, California: ACM, 2012. URL: http://advances.realtimerendering.com/s2012/.

[MP12]   P. Mavridis and G. Papaioannou. "The Compact YCoCg Frame Buffer". In: *Journal of Computer Graphics Techniques (JCGT)* 1.1 (Sept. 2012), pp. 19–35. URL: `http://jcgt.org/published/0001/01/02/`.

[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 2013 Courses*. SIGGRAPH '13. Anaheim, California: ACM, 2013, 22:1–22:8. ISBN: 978-1-4503-2339-0. DOI: `10.1145/2504435.2504457`. URL: `http://selfshadow.com/publications/s2013-shading-course/`.

[OB10]   M. Olano and D. Baker. "LEAN Mapping". In: *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '10. Washington, D.C.: ACM, 2010, pp. 181–188. ISBN: 978-1-60558-939-8. DOI: `10.1145/1730804.1730834`. URL: `http://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. New York, NY, USA: ACM, 1994, pp. 239–246. ISBN: 0-89791-667-0. DOI: `10.1145/192161.192213`. URL: `http://www1.cs.columbia.edu/CAVE/publications/`.

[Opea]   OpenEXR. URL: `http://www.openexr.com`.

[Opeb]   OpenGL. URL: `http://www.opengl.org/wiki/Small_Float_Formats`.

[Per11]   E. Persson. "Volume Decals". In: *GPU Pro 2*. Ed. by W. Engel. A K Peters, 2011, pp. 115–120.

[PH10]   M. Pharr and G. Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010. ISBN: 0123750792, 9780123750792. URL: `http://www.pbrt.org/`.

[Pla]   PlanetMath. *Solid angle of rectangular pyramid*. URL: `http://planetmath.org/solidangleofrectangularpyramid`.

[Quí06]   I. Quílez. *Sphere ambient occlusion*. 2006. URL: `http://www.iquilezles.org/www/articles/sphereao/sphereao.htm`.

[Quí12]   I. Quílez. *Multiresolution ambient occlusion*. 2012. URL: `http://www.iquilezles.org/www/articles/multiresaocc/multiresaocc.htm`.

[Ree14]   N. Reed. *Artist-Friendly HDR With Exposure Values*. 2014. URL: `http://www.reedbeta.com/blog/2014/06/04/artist-friendly-hdr-with-exposure-values`.

[Rei+05]   E. Reinhard, G. Ward, S. Pattanaik, and P. Debevec. *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting (The Morgan Kaufmann Series in Computer Graphics)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005. ISBN: 0125852630.

[Rei+08]   E. Reinhard, E. A. Khan, A. O. Akyuz, and G. M. Johnson. *Color Imaging: Fundamentals and Applications*. AK Peters, June 2008. URL: `http://www.cs.bris.ac.uk/Publications/pub_master.jsp?id=2001311`.

[RH01]   R. Ramamoorthi and P. Hanrahan. "An Efficient Representation for Irradiance Environment Maps". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 497–500. ISBN: 1-58113-374-X. DOI: `10.1145/383259.383317`. URL: `http://cseweb.ucsd.edu/~ravir/papers/envmap/`.

[Rye]   A. Ryer. *Light Measurement Handbook*. International Light Technologies Inc. URL: `http://www.intl-lighttech.com/services/ilt-light-measurement-handbook`.

[Sch09]   C. Schüler. "An efficient and Physically Plausible Real-Time Shading Model". In: *ShaderX7: Advanced Rendering Techniques*. Ed. by W. Engel. Charles River Media, 2009. Chap. 2.5.

[Sch14]   N. Schulz. "Moving to the Next Generation - The Rendering Technology of Ryse". In: Game Developers Conference. 2014.

[Sny96]   J. M. Snyder. *Area Light Sources for Real-Time Graphics*. Tech. rep. MSR-TR-96-11. Microsoft Research, Mar. 1996, p. 30. URL: `http://research.microsoft.com/apps/pubs/default.aspx?id=69565`.

[Sou13]   T. Sousa. "CryENGINE 3 Graphics Gems". In: *Advances in Real-time Rendering in Games Part II, ACM SIGGRAPH 2013 Courses*. SIGGRAPH '13. Anaheim, California: ACM, 2013, 15:1–15:1. ISBN: 978-1-4503-2339-0. DOI: 10.1145/2504435.2504450. URL: `http://advances.realtimerendering.com/s2013/index.html`.

[Sys]   F. Systems. *EULUMDAT Tools*. URL: `http://www.fold1.com/eulumdattools/`.

[Tok05]   M. Toksvig. "Mipmapping Normal Maps". In: *Journal of Graphics, GPU, and Game Tools* 10.3 (2005), pp. 65–71. DOI: 10.1080/2151237X.2005.10129203. URL: `http://www.nvidia.com/object/mipmapping_normal_maps.html`.

[UE4]   UE4Forum. *UE4 private forum support*.

[UFK13]   C. Ureña, M. Fajardo, and A. King. "An Area-Preserving Parametrization for Spherical Rectangles". In: *Computer Graphics Forum* 32.4 (2013), pp. 59–66. DOI: 10.1111/cgf.12151. URL: `https://www.solidangle.com/arnold/research/`.

[Ulu14]   Y. Uludag. "Hi-Z Screen-Space Cone-Traced Reflections". In: *GPU Pro 5*. Ed. by W. Engel. CRC Press, 2014, pp. 149–192.

[Vai14]   M. Vainio. "The Visual Effects of inFAMOUS: Second Son". In: Game Developers Conference. 2014. URL: `http://www.gdcvault.com/play/1020158/The-Visual-Effects-of-inFAMOUS`.

[Val14]   M. Valient. "Making Killzone Shadow Fall Image Quality into the Next Generation". In: Game Developers Conference. 2014. URL: `http://www.guerrilla-games.com/presentations/`.

[Vla08]   A. Vlachos. "Post Processing in the Orange Box". In: Game Developers Conference. 2008. URL: `http://www.valvesoftware.com/publications/`.

[Wal+]   B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. "Microfacet Models for Refraction through Rough Surfaces". In: pp. 195–206. DOI: 10.2312/EGWR/EGSR07/195-206. URL: `https://www.cs.cornell.edu/~srm/publications/EGSR07-btdf.html`.

[Wika]   Wikipedia. *Angular diameter*. URL: `http://en.wikipedia.org/wiki/Angular_diameter`.

[Wikb]   Wikipedia. *Candela*. URL: `http://en.wikipedia.org/wiki/Candela`.

[Wikc]   Wikipedia. *Film speed*. URL: `http://en.wikipedia.org/wiki/Film_speed`.

[Wikd]   Wikipedia. *Fresnel equations*. URL: `http://en.wikipedia.org/wiki/Fresnel_equations`.

[Wike]   Wikipedia. *Human Eye*. URL: `http://en.wikipedia.org/wiki/Human_eye`.

[Wikf]   Wikipedia. *Inverse square law*. URL: `http://en.wikipedia.org/wiki/Inverse-square_law`.

[Wikg]   Wikipedia. *Light meter*. URL: `http://en.wikipedia.org/wiki/Light_meter`.

[Wikh]   Wikipedia. *Raw image format.* URL: http://en.wikipedia.org/wiki/Raw_image_forma
            t.

[Wiki]   Wikipedia. *Solid angle.* URL: http://en.wikipedia.org/wiki/Solid_angle.

[Wikj]   Wikipedia. *"Sunny 16" rule.* URL: http://en.wikipedia.org/wiki/Sunny_16_rule.

Appendix A

Listing for reference mode

Listing A.1 shows a simple kernel for integrating a light probe with a specular GGX microfacet model in order to quickly see the expected results.

```
1  float3 evaluateSpecularIBLReference (
2       in float3 N,
3       in float3 V,
4       in float roughness ,
5       in float3 f0,
6       in float f90)
7  {
8       // Build local referential
9       float3 upVector = abs(N.z) < 0.999 ? float3(0,0,1) : float3(1,0,0);
10      float3 tangentX = normalize( cross( upVector, N ) );
11      float3 tangentY = cross( N, tangentX );
12
13      float3 accLight = 0;
14      for (uint i=0; i<g_sampleCount; ++i)
15      {
16          float2 u = getSample(i, g_sampleCount);
17
18          // GGX NDF sampling
19          float cosThetaH = sqrt( (1-u.x) / (1 + (roughness*roughness-1)*u.x) );
20          float sinThetaH = sqrt(1 - min(1.0,cosThetaH*cosThetaH));
21          float phiH = u.y * FB_PI * 2;
22
23          // Convert sample from half angle to incident angle
24          float3 H;
25          H = float3(sinThetaH*cos(phiH), sinThetaH*sin(phiH), cosThetaH);
26          H = normalize(tangentX * H.y + tangentY * H.x + N * H.z);
27          L = normalize(2.0f * dot(V,H) * H - V);
28
29          float LdotH = saturate(dot(H, L));
30          float NdotH = saturate(dot(H, N));
31          float NdotV = saturate(dot(V, N));
32          float NdotL = saturate(dot(L, N));
33
34          // Importance sampling weight for each sample
35          //
36          //    weight = fr . (N.L)
37          //
38          // with:
39          //    fr   = D(H) . F(H) . G(V, L) / ( 4 (N.L) (N.O) )
40          //
41          // Since we integrate in the microfacet space, we include the
42          // Jacobian of the transform
43          //
44          //    pdf = D(H) . (N.H) / ( 4 (L.H) )
45          float D     = D_GGX(NdotH, roughness);
46          float pdfH  = D * NdotH;
47          pdf         = pdfH / (4.0f * LdotH);
```

```
48
49            // Implicit weight (N.L canceled out)
50            float3 F     = F_Schlick(f0, f90, LdotH);
51            float  G     = G_SmithGGX(NdotL, NdotV, roughness);
52            weight       = F * G * D / (4.0 * NdotV);
53
54            if (dot(L,N)>0 && pdf > 0)
55            {
56                accLight += g_IBL.SampleLevel(sampler, L, 0).rgb * weight / pdf;
57            }
58        }
59    return accLight / g_sampleCount;
60 }
```

Listing A.2 shows a simple kernel for integrating a light probe with a diffuse Disney BRDF model in order to quickly see the expected results.

```
1
2 void importanceSampleCosDir(
3     in float2 u,
4     in float3 N,
5     out float3 L,
6     out float NdotL,
7     out float pdf)
8 {
9     // Local referencial
10    float3 upVector = abs(N.z) < 0.999 ? float3(0,0,1) : float3(1,0,0);
11    float3 tangentX = normalize( cross( upVector, N ) );
12    float3 tangentY = cross( N, tangentX );
13
14    float u1 = u.x;
15    float u2 = u.y;
16
17    float r = sqrt(u1);
18    float phi = u2 * FB_PI * 2;
19
20    L = float3(r*cos(phi), r*sin(phi), sqrt(max(0.0f,1.0f-u1)));
21    L = normalize(tangentX * L.y + tangentY * L.x + N * L.z);
22
23    NdotL = dot(L,N);
24    pdf = NdotL * FB_INV_PI;
25 }
26
27 float3 evaluateIBLDiffuseCubeReference(
28    in TextureCube<float4> incomingLight,
29    in SamplerState incomingLightSampler,
30    in float3 V,
31    in MaterialData data,
32    in float3 brdfWeight = float3(0, 0, 0),
33    in uint sampleCount=1024)
34 {
35    float3 accLight = 0;
36
37    for (uint i=0; i<sampleCount; ++i)
38    {
39        float2 eta = getSample(i, sampleCount);
40        float3 L;
41        float NdotL;
42        float pdf;
43        importanceSampleCosDir(eta, data.worldNormal, L, NdotL, pdf);
44        if (NdotL>0)
45        {
46            // Each sample should be weighted by L * weight / pdf.
47            // With:
48            //   - weight = NdotL
49            //   - pdf = NdoL / Pi
```

```
50              // However the NdoLs (in weight and pdf) and Pi cancel out
51              // This is why all terms disappear here
52              float f = 1.0f;
53
54          #if FB_DIFFUSE_MODEL == FB_DIFFUSE_DISNEY
55              // Half angle formula:
56              // cos(2 theta) = 2cos^2(theta) - 1
57              float cosD = sqrt((dot(V, L) + 1.0f) * 0.5);
58              float NdotV = saturate(dot(data.worldNormal, V));
59              float NdotL_sat = saturate(NdotL);
60              // Disney diffuse BRDF operates in linear roughness,
61              // which is the sqrt of the GGX alpha roughness term
62              float fd90 = 0.5 + 2 * cosD*cosD * sqrt(data.roughness);
63              float lightScatter =  1 + (fd90-1) * pow(1-NdotL_sat, 5);
64              float viewScatter  =  1 + (fd90-1) * pow(1-NdotV, 5);
65              f = lightScatter * viewScatter;
66          #endif
67
68              accLight += incomingLight.SampleLevel(incomingLightSampler, L, 0).rgb * f;
69          }
70      }
71
72      return accLight * (1.0f / sampleCount);
73 }
```

Listing A.2: Runtime reference code for integrating a diffuse Disney BRDF with a distant light probe.

Listing A.3 shows different light importance sampling methods for integrating a microfacet BRDF with an area light in order to quickly see the expected results.

```
1  // our light point backward (-Z)
2
3  float3 uniformSampleSphere(float u1, float u2)
4  {
5      float phi       = 2.0f * FB_PI * u2;
6      float cosTheta  = 1.0f - 2.0f * u1;
7      float sinTheta  = sqrt(max(0.0f, 1.0f - cosTheta * cosTheta));
8
9      return float3(sinTheta * cos(phi), sinTheta * sin(phi), cosTheta);
10 }
11
12 float3 uniformSampleDisk(float u1, float u2)
13 {
14     float r         = sqrt(u1);
15     float phi       = 2.0f * FB_PI * u2;
16
17     return float3(r * cos(phi), r * sin(phi), 0);
18 }
19
20 void sampleSphere(
21     in float2    u,
22     in float4x4 localToWorld,
23     in float     radius,
24     out float    lightPdf,
25     out float3   P,
26     out float3   Ns)
27 {
28     float u1 = u.x;
29     float u2 = u.y;
30
31     // Random point at light surface
32     Ns          = uniformSampleSphere(u1, u2);
33     // Transform point on unit sphere to world space
34     P           = radius * Ns + localToWorld[3].xyz;
35     lightPdf    = 1.0f / (4.0f * FB_PI * radius * radius);
36 }
37
38 // Note: The cylinder has no end caps (i.e. no disk on the side)
```

```
39 void sampleCylinder(
40     in float2 u,
41     in float4x4 localToWorld,
42     in float radius,
43     in float width,
44     out float   lightPdf,
45     out float3  P,
46     out float3  Ns)
47 {
48     float u1 = u.x;
49     float u2 = u.y;
50
51     // Random point at light surface
52     float t         = (u1 - 0.5) * width;
53     float theta     = 2.0f * FB_PI * u2;
54     float cosTheta  = cos(theta);
55     float sinTheta  = sin(theta);
56     //  Cylinder are align on the left axis in Frosbite
57     P               = float3(t, radius * cosTheta, radius * sinTheta);
58     Ns              = normalize(float3(0.0f, cosTheta, sinTheta));
59     // Transform to world space
60     P               = mul(float4(P, 1.0f), localToWorld).xyz;
61     Ns              = mul(Ns, (float3x3)localToWorld);
62     lightPdf        = 1.0f / (2 * FB_PI * radius * width);
63 }
64
65 void sampleCapsule(
66     in float2 u,
67     in float4x4 localToWorld,
68     in float radius,
69     in float width,
70     in int passIt,
71     out float   lightPdf,
72     out float3  P,
73     out float3  Ns)
74 {
75     // Capsules are sampled in two times:
76     //  - Pass 0: Cylinder
77     //  - Pass 1: Hemisphere caps
78     if (passIt == 0)
79     {
80         sampleCylinder(u, localToWorld, radius, width, lightPdf, P, Ns);
81     }
82     else
83     {
84         float u1 = u.x;
85         float u2 = u.y;
86
87         // Random point at light surface
88         Ns          = uniformSampleSphere(u1, u2);
89         P           = radius * Ns;
90
91         // Split the sphere into two hemisphere and shift
92         // each hemisphere on one side of the cylinder
93         P.x         += (Ns.x>0? 1 : -1) * 0.5 * width;
94
95         // Transform to world space
96         P           = mul(float4(P, 1.0f), localToWorld).xyz;
97         Ns          = mul(Ns, (float3x3)localToWorld);
98         lightPdf = 1.0f / (4.0f * FB_PI * radius * radius);
99     }
100 }
101
102 void sampleRectangle(
103     in float2 u,
104     in float4x4 localToWorld,
105     in float width,
106     in float height,
107     out float   lightPdf,
108     out float3  P,
```

```
109       out float3  Ns)
110   {
111       // Random point at light surface
112       P = float3((u.x - 0.5f) * width, (u.y - 0.5f) * height, 0);
113       Ns = float3(0, 0, -1);
114
115       // Transform to world space
116       P = mul(float4(P, 1.0f), localToWorld).xyz;
117       Ns = mul(Ns, (float3x3)localToWorld);
118       lightPdf = 1.0f / (width * height);
119   }
120
121   void sampleDisk(
122       in float2 u,
123       in float4x4 localToWorld,
124       in float radius,
125       out float   lightPdf,
126       out float3  P,
127       out float3  Ns)
128   {
129       // Random point at light surface
130       P = uniformSampleDisk(u.x, u.y) * radius;
131       Ns = float3(0, 0, -1);
132
133       // Transform to world space
134       P = mul(float4(P, 1.0f), localToWorld).xyz;
135       Ns = mul(Ns, (float3x3)localToWorld);
136
137       // pdf is just the inverse of the area
138       lightPdf = 1.0f / (FB_PI * radius * radius);
139   }
140
141   // Compute the number of samples for a given pass and light type
142   uint getSampleCount(
143       in AreaLightInfo light,
144       in float areaLightType,
145       in uint passIt,
146       in uint sampleCountBase)
147   {
148       uint sampleCount = passIt == 0 ? sampleCountBase : 0;
149       if (areaLightType == AreaLightType_Capsule)
150       {
151           float r = light.lightRadius0;
152           float h = light.lightRadius1;
153           float cylinderArea = 2 * FB_PI * r * h;
154           float sphereArea = 4 * FB_PI * r * r;
155           float totalArea = cylinderArea + sphereArea;
156
157           sampleCount = sampleCountBase * ((passIt == 0 ? cylinderArea : sphereArea) / (
158               cylinderArea + sphereArea));
158       }
159
160       return sampleCount;
161   }
162
163   void evaluateAreaLightReference(
164       in float3 worldPos,
165       in float3 V,
166       in MaterialData data,
167       in AreaLightInfo light,
168       inout float3 outBottom,
169       inout float3 outTop,
170       in uint sampleCountBase = 512)
171   {
172     // Sampling is decomposed into several passes, as certain lights are
173     // composed with several shapes (e.g. capsule). In order to sample
174     // these lights correctly, we render a pass per composite shape.
175     // The number of samples is spread among all the shapes proportional
176     // to their area.
177     float areaLightType = light.areaLightType;
```

```
178    uint passCount = areaLightType == AreaLightType_Capsule ? 2 : 1;
179    for (uint passIt = 0; passIt < passCount; ++passIt)
180    {
181      // Accumulate this light locally, then add on at the end
182      // (avoids any accidental multiplies of in/out accumulated variables)
183      float3 accBottom = 0;
184      float3 accTop = 0;
185      uint sampleCount = getSampleCount(light, areaLightType, passIt, sampleCountBase);
186      for (uint sampleIt = 0; sampleIt < sampleCount; ++sampleIt)
187      {
188          float3 P           = 0;     // Sample light point
189          float3 Ns          = 0;     // Unit surface normal at P
190          float  lightPdf    = 0;     // Pdf of the light sample
191          float2 u           = getSample(sampleIt, sampleCount);
192
193          float4x4 localToWorld = light.localToWorld;
194
195          if (areaLightType == AreaLightType_Sphere)
196            sampleSphere(u,localToWorld,light.lightRadius0,lightPdf,P,Ns);
197          else if (areaLightType == AreaLightType_Capsule)
198            sampleCapsule(u,localToWorld,light.lightRadius0,light.lightRadius1,passIt,lightPdf,P,
199                Ns);
199          else if (areaLightType == AreaLightType_Rectangular)
200            sampleRectangle(u,localToWorld,light.lightRadius0,light.lightRadius1,lightPdf,P,Ns);
201          else if (areaLightType == AreaLightType_Disk)
202            sampleDisk(u,localToWorld,light.lightRadius0,lightPdf,P,Ns);
203
204          // Get distance
205          float3 unnormalizedLightVector = P - worldPos;
206          float  sqrDist = dot(unnormalizedLightVector,unnormalizedLightVector);
207          float3 L = normalize(unnormalizedLightVector);
208
209          float illuminance = saturate(dot(Ns, -L)) *
210                      saturate(dot(data.worldNormal, L)) / (sqrDist*lightPdf);
211
212          float3 localDiffuse;
213          float3 localSpecular;
214          BSDF(V, L, data, illuminance, localDiffuse, localSpecular);
215
216          accBottom   += localDiffuse;
217          accTop      += localSpecular;
218      }
219
220      float3 localBottom  = (accBottom / float(sampleCount)) * light.color;
221      float3 localTop     = (accTop / float(sampleCount)) * light.color;
222
223      outBottom   += localBottom;
224      outTop      += localTop;
225    }
226 }
```

Listing A.3: Runtime reference code for integrating a microfacet BRDF with different area lights

Appendix B

Oren-Nayar and GGX's diffuse term derivation

The conversion of the roughness parameter from α (for a GGX-based microfacet model) to α_{ON} (for Oren Nayar) is not straightforward. The Oren-Nayar model uses a spherical space to express its roughness term, meaning that the roughness measures angles, unlike the GGX model which defines roughness in slope space. Equation B.1 describes the transformation function for going from a Beckmann NDF roughness α to an Oren-Nayar based roughness α_{ON}. Both Beckmann NDF and GGX NDF express roughness in slope space. Thus the conversion formula can be reused with GGX roughness. This relationship is accurate for low values of α ($<= 0.4$), then the accuracy decreases progressively, see Figure B.1.

$$\alpha_{ON} = \frac{1}{\sqrt{2}} \arctan(\alpha) \tag{B.1}$$
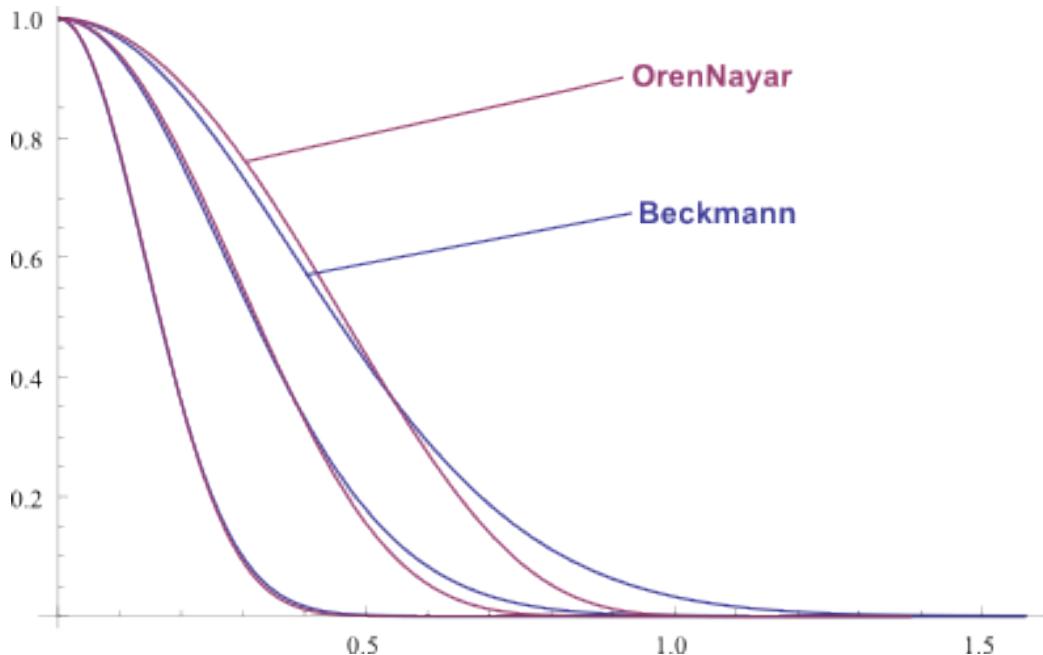
Figure B.1: Comparison of Oren-Nayar NDF with roughness conversion ($\alpha_{ON} = \frac{1}{\sqrt{2}} \arctan(\alpha)$) and Beckmann NDF for various roughness values (From left to right: 0.2, 0.4, 0.6). For $\alpha > 0.4$, the accuracy decreases a lot.

As shown, there is no good conversion between GGX and Oren-Nayar roughness. This motivated us

to try to find a proper derivation of the diffuse term from GGX with Equation 4 (and thus using GGX roughness for the diffuse term). Oren-Nayar have derived an approximation for a Gaussian NDF distribution and a V-cavity G term. We have try to take a similar approach for deriving an approximation with the GGX's derived diffuse term. In parallel of our own work, Gotanda [Got14] have provided an approximation and a detailed comparison of the GGX's derived diffuse term and the Oren-Nayar diffuse. It appears that the new derivation doesn't suffer from discontinuity of the Oren-Nayar model (which is due to the V-cavity visibility) and is darker with high roughness. The approximation still heavy in instructions. We went for the Disney model as its simplicity, its properties and its good comparison with the MERL database were attractive. Futher research is necessary to compare the new GGX's diffuse derivation with real world materials but we feel that it is interesting path to follow.

Appendix C

Energy conservation

Ideally, we would handle the transfer of energy between specular and diffuse correctly, but this can be quite involved. In order to simplify this quick analysis, we will assume that the mean-free-path (i.e. the average distance travelled by a photon in the matter) of the diffuse term is below the size of a microfacet. With this hypothesis, we can consider all light interactions are located into a single microfacet. For the evaluation of a directional light, the light would come from a single direction and hit all microfacets visible to the light. This light would be transmitted inside the diffuse medium with a Fresnel transmittance and be scattered in a diffuse way inside the medium. Once scattered, the light would be transmitted again through the interface with a diffuse Fresnel transmittance [Jen+01]. This light would then be evaluated for all microfacets visible from the view direction. In this context, Equation 4 would become, with F_{dr} the diffuse Fresnel reflectance:

$$f_d(\mathbf{v}) = \frac{\rho}{\pi} \frac{1}{|\mathbf{n} \cdot \mathbf{v}||\mathbf{n} \cdot \mathbf{l}|} \int_\Omega (1 - F(\mathbf{l})) \ (1 - F_{dr}) \ G(\mathbf{v}, \mathbf{l}, \mathbf{m}) \ D(\mathbf{m}, \alpha) \ \langle \mathbf{v} \cdot \mathbf{m} \rangle \langle \mathbf{l} \cdot \mathbf{m} \rangle \ \mathrm{d}\mathbf{m} \qquad \text{(C.1)}$$

For other types of evaluation, like hemispherical evaluation for distant lighting, or specific solid angles for area light evaluation, computations are a bit more involved. The hemispherical case can be pre-computed to reduce the amount of computation needed, whilst the area light case is trickier as pre-computation would require many dimensions or coarse approximations.

Another tricky aspect mentioned by Heitz [Hei14], is that microfacet specular BRDF models only model the first bounce of light. They do not manage inter-reflections which can be important when α increases. One solution suggested by Jakob et al. [Jak+14] is to redistribute the lost energy in a near diffuse distribution in order to not lose energy. While it seems to be a coarse approximation, their results look plausible.

Appendix D

Optimization algorithm for converting to Disney's parametrization

This section presents an optimization algorithm to convert two color values (diffuse albedo and f_0) to Disney's principled parameters. The two color parametrization uses six independent scalar values (diffuse color, f_0 color). The Disney parameterization uses five independent scalar values (baseColor, metal, reflectance). The relationship between the two parameterizations is as follows:

$$diffuseColor = (1 - metalMask)\ baseColor \tag{D.1}$$

$$f_0 = 0.16\ reflectance^2\ (1 - metalMask) + baseColor\ metalMask \tag{D.2}$$

If $diffuseColor$ and f_0 are given, the two equations define an over-constrained non-linear equation system. We minimize error on this with a non-linear least squares minimization algorithm using the Levenberg-Marquardt algorithm to find a best fit for the baseColor, metalMask and reflectance output parameters. Such an algorithm works best by feeding in an analytical Jacobian, which can be easily calculated with Mathematica's derivative function, see Listing D.1.

```
1
2  CalcF0[reflectance_, metalMask_, baseColor_] := 0.16 reflectance^2 (1 - metalMask) + baseColor
       metalMask
3  CalcD[metalMask_, baseColor_] := (1 - metalMask) baseColor
4
5  D[{CalcF0[reflectance, metalMask, {baseColorX, baseColorY, baseColorZ}] - f0, CalcD[metalMask,
       {baseColorX, baseColorY, baseColorZ}] - d}, {{reflectance, metalMask, baseColorX,
       baseColorY, baseColorZ}}]
6
7  // The resulting Jacobian looks like this:
8  {{{0.32 (1 - metalMask) reflectance, baseColorX - 0.16 reflectance^2, metalMask, 0, 0}, {0.32
       (1 - metalMask) reflectance, baseColorY - 0.16 reflectance^2, 0, metalMask, 0}, {0.32 (1 -
       metalMask) reflectance, baseColorZ - 0.16 reflectance^2, 0, 0, metalMask}}, {{0, -
       baseColorX, 1 - metalMask, 0,  0}, {0, -baseColorY, 0, 1 - metalMask, 0}, {0, -baseColorZ,
       0, 0, 1 - metalMask}}}
```

Listing D.1: Mathematica analytical Jacobian for Levenberg-Marquardt algorithm.

When running the minimization algorithm, we want to limit the values of the input parameters to the range of 0 to 1, since we store this back to 8-bit texture files. We use CMinpack [Dev07] to perform the conversion in C++ code. We provide a Mathematica file with this document which contains a similar version of the algorithm described above. It loads, converts and saves the textures.

Appendix E

Rectangular area lighting

Form factor solution without horizon handling for rectangular area lights:

- Arvo [Arv94] provides a form factor without correct horizon handling for an oriented rectangle. This form factor is reported in Listing E.1.

- As far as we are aware, no form factor exists with correct horizon handling for the general case. It is possible to get a semi-analytical solution by clipping the rectangle against the plane defined by the surface normal [McG10], but this is a costly operation for our real time constraints. Some approximations can be achieved by clamping the rectangle's point into the positive normal hemisphere of the surface normal but we have not found an affordable solution.

```
1  // Calculate illuminance above the horizon
2  float illuminance   = 0;
3
4  if (dot(worldPos - lightPos, planeNormal) > 0) // On same side of the plane
5  {
6      float halfWidth = width * 0.5;
7      float halfHeight = height * 0.5;
8      float3 p0 = lightPos + lightLeft * -halfWidth + lightUp * halfHeight;
9      float3 p1 = lightPos + lightLeft * -halfWidth + lightUp * -halfHeight;
10     float3 p2 = lightPos + lightLeft * halfWidth + lightUp * -halfHeight;
11     float3 p3 = lightPos + lightLeft * halfWidth + lightUp * halfHeight;
12
13     // contour integral integration (Lambert)
14     float3 v0 = normalize(p0 - worldPos);
15     float3 v1 = normalize(p1 - worldPos);
16     float3 v2 = normalize(p2 - worldPos);
17     float3 v3 = normalize(p3 - worldPos);
18
19     float fD1 = acos(dot(v0, v1));
20     float fD2 = acos(dot(v1, v2));
21     float fD3 = acos(dot(v2, v3));
22     float fD4 = acos(dot(v3, v0));
23
24     float3 vCross1 = normalize(cross(v0, v1)) * fD1;
25     float3 vCross2 = normalize(cross(v1, v2)) * fD2;
26     float3 vCross3 = normalize(cross(v2, v3)) * fD3;
27     float3 vCross4 = normalize(cross(v3, v0)) * fD4;
28
29     float3 unormLightVector = vCross1 + vCross2 + vCross3 + vCross4;
30     illuminance = FB_PI * 1 / (2 * FB_PI) * saturate(dot(N, unormLightVector));
31
32     L = normalize(unormLightVector);
33 }
```

Listing E.1: Rectangular area light form factor without horizon.

Drobot's Most representative light: Drobot approximates the MRP by finding the point which maximizes the BRDF divided by the squared distance. For a diffuse area light this is $\frac{\langle \mathbf{n} \cdot \mathbf{l} \rangle}{distance^2}$. Following this definition we have the following steps for a point p with a normal $\mathbf{n}$:

- p_0 is the projection of p onto the light plane, minimizing $\frac{1}{distance^2}$.
- $\vec{d_0}$ is the vector $p\vec{p_0}$.
- p_1 is the intersection of a ray from p in direction $\vec{n}$ with the light plane, maximizing $\langle \mathbf{n} \cdot \mathbf{l} \rangle$).
- $\vec{d_1}$ is the vector $p\vec{p_1}$.
- $\vec{d_h}$ is the halfway vector between $\vec{d_0}$ and $\vec{d_1}$.
- p_h is the intersection of $\vec{d_h}$ with the light plane and represents the MRP.
- p_h is moved to the closest point on the rectangular light if it is outside.
- Evaluate the single light at the MRP and multiply by the solid angle.

The MRP always lies between d_0 and d_1 and p_h is the MRP approximation. An improvement over this approach which is more numerically robust and simpler is:

- $\vec{d_0'}$ is the opposite direction to the light plane normal.
- $\vec{d_0'}$ is clamped to the positive hemisphere of the surface plane normal.
- $\vec{d_1'}$ is the direction of the surface normal.
- $\vec{d_1'}$ is clamped to the negative hemisphere of the light plane normal.
- $\vec{d_h}$ is the halfway vector between $\vec{d_0}$ and $\vec{d_1}$.
- p_h is the intersection of $\vec{d_h}$ with the light plane and represents the MRP.
- p_h is moved to the closest point on the rectangular light if it is outside.
- Evaluate the single light at the MRP and multiply by the solid angle.

Theses steps are illustrated in Figure E.1 and the corresponding code is provided in Listing E.2.
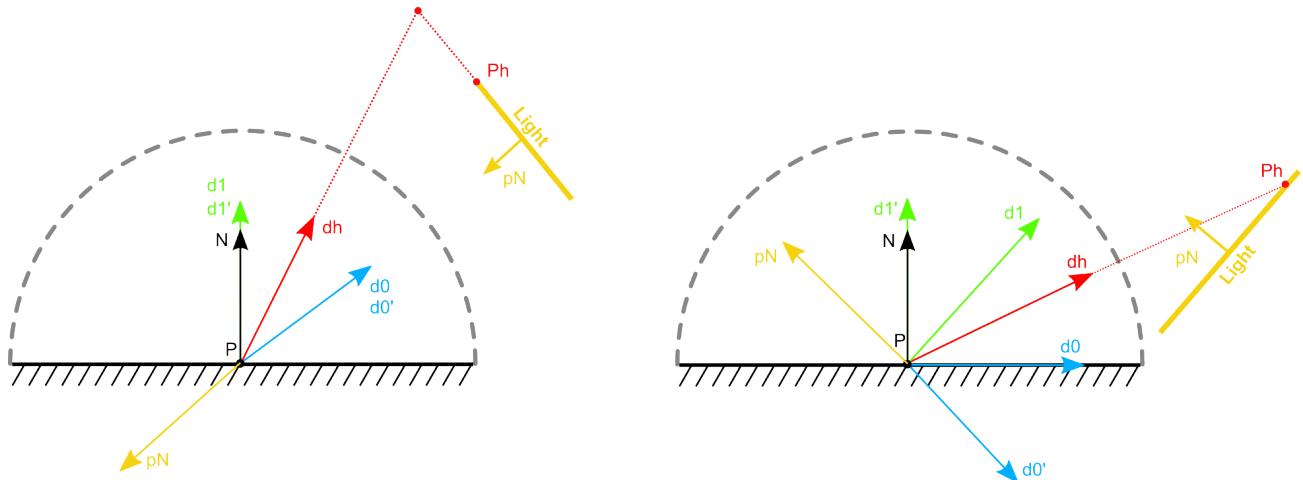
Figure E.1: Left: Light is facing the shaded point. $\vec{d_0'}$ and $\vec{d_1'}$ are not clamped and are equal to $\vec{d_0}$ and $\vec{d_1}$. Intersection of $\vec{d_h}$ and the light plane is outside the light. p_h is moved to the closest light rectangle boundary and is the MRP. Right: Light facing away from the shaded point. $\vec{d_0'}$ and $\vec{d_1'}$ are clamped. p_h is the MRP.

```
float3 rayPlaneIntersect(in float3 rayOrigin, in float3 rayDirection,
                         in float3 planeOrigin, in float3 planeNormal)
{
```

```
 5        float distance = dot(planeNormal, planeOrigin - rayOrigin) /
 6                        dot(planeNormal, rayDirection);
 7        return rayOrigin + rayDirection * distance;
 8 }
 9
10 // Return the closest point to a rectangular shape defined by two vectors
11 // left and up
12 float3 closestPointRect(in float3 pos, in float3 planeOrigin, in float3 left, in float3 up, in
13      float halfWidth, in float halfHeight)
13 {
14      float3 dir = pos - planeOrigin;
15
16      // - Project in 2D plane (forward is the light direction away from
17      //   the plane)
18      // - Clamp inside the rectangle
19      // - Calculate new world position
20      float2 dist2D = float2(dot(dir, left), dot(dir, up));
21      float rectHalfSize = float2(halfWidth, halfHeight);
22      dist2D = clamp(dist2D, -rectHalfSize, rectHalfSize);
23      return planeOrigin + dist2D.x * left + dist2D.y * up;
24 }
25
26 if (dot(worldPos - light.pos, lightPlaneNormal) > 0)
27 {
28    float clampCosAngle = 0.001 + saturate(dot(worldNormal, lightPlaneNormal));
29    // clamp d0 to the positive hemisphere of surface normal
30    float3 d0 = normalize(-lightPlaneNormal + worldNormal * clampCosAngle);
31    // clamp d1 to the negative hemisphere of light plane normal
32    float3 d1 = normalize( worldNormal - lightPlaneNormal * clampCosAngle);
33    float3 dh = normalize(d0 + d1);
34    ph = rayPlaneIntersect(worldPos, dh, lightPos, lightPlaneNormal);
35    ph = closestPointRect(ph, lightPos, lightLeft, lightUp,
36                          lightHalfWidth, lightHalfHeight);
37
38    float3 p0 = lightPos + lightLeft * -lightHalfWidth + lightUp *  lightHalfHeight;
39    float3 p1 = lightPos + lightLeft * -lightHalfWidth + lightUp * -lightHalfHeight;
40    float3 p2 = lightPos + lightLeft *  lightHalfWidth + lightUp * -lightHalfHeight;
41    float3 p3 = lightPos + lightLeft *  lightHalfWidth + lightUp *  lightHalfHeight;
42
43    float solidAngle = rectangleSolidAngle(worldPos, p0, p1, p2, p3);
44
45    float3 unormLightVector = ph - worldPos;
46    float sqrDist = dot(unormLightVector, unormLightVector);
47    L = normalize(unormLightVector);
48
49    illuminance = solidAngle * saturate(dot(worldNormal, L));
50 }
```

Listing E.2: Simple MRP detection method.

The current implementation lacks correct horizon handling. The solid angle of a rectangle, taking into account the horizon is not available causing stronger intensity than expected to the wrap lighting and the MRP varies from pixel to pixel is causing disturbing waving, see Figure E.2.

Diffuse area light framework: A Mathematica comparison framework, that we used to compare our methods, is provided as a companion file to this document. It is difficult to find the right error metric for comparison because the metric can be quickly biased by edge cases. For instance, using a relative error metric results in 100% error if a method produces an illuminance of zero, even if the true result is 0.000001. Thus we provide two error estimations: a relative mean error and an absolute mean error. For far distances, all methods are equivalent (following the principle of the five times rules), so we chose to only compare close range configurations. Also, as some methods do not support the horizon, we chose to separate error estimation for lights above horizon and lights crossing the horizon. Table E.1 shows the estimation errors of various methods, performed on 1000 random close-range configurations
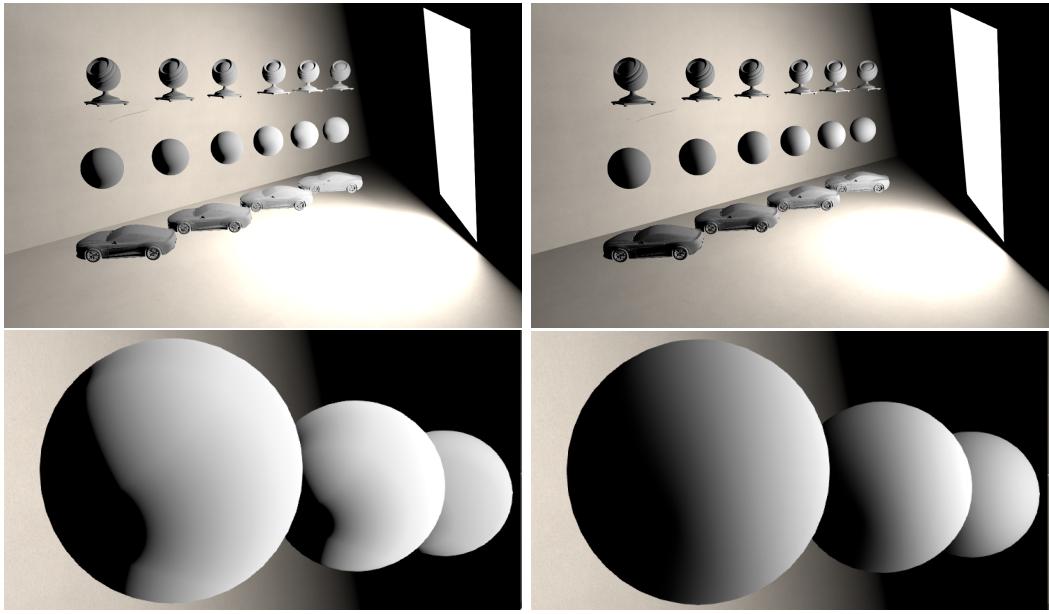
Figure E.2: Top left: Result of a rectangular area light with incorrect horizon handling. Top right: Reference. Bottom left: Closer shot of top left. See how the wrapped lighting is stronger and wavier than expected. Bottom right: Closer shot of top right.

for both cases. The methods are:

- Form factor: Method using the form factor equation based on Arvo, presented at the beginning of this appendix.
- Reference MRP: NRP determined by brute force.
- Drobot's MRP: MRP approximation as proposed by Drobot.
- Simplified MRP: Same as Drobot's with code simplification.
- Light position MRP: MRP is chosen to be the light's position.
- Structured sampling: Structured sampling of the light shape.

The numbers need to be read with care, as they do not show what the visual result will be. Some methods can have edge cases and others exhibit banding, whilst others do not handle the horizon but their mean errors are accurate. The table shows that:

- Structured sampling of the light shape is an accurate approach in all cases.
- Handling the horizon case for solid angle calculations for the MRP approach is important.
- Using the light's position as the MRP could be the best choice. However this needs some attention. The mean error here is low but does not handle wrapped lighting. So this approach is best for lights with small subtended solid angles.
- The brute force MRP approach is sometime less accurate than the Drobot's approximate MRP. This highlights that there are pitfalls in the MRP approach (as the best MRP doesn't always give you the best result). Thus the method should be avoided for diffuse area lights.

Method	Relative (AH)	Absolute (AH)	Relative (CH)	Absolute (CH)
Form factor	0.0024	0.0005	-	-
Reference MRP	0.1832	0.0209	1.3229	0.1409
Reference MRP (H)	-	-	0.8879	0.0835
Drobot's MRP	0.1890	0.0237	1.1347	0.1936
Drobot's MRP (H)	-	-	0.8487	0.1260
Simplified MRP	0.1905	0.0230	1.3106	0.1848
Simplified MRP (H)	-	-	0.9025	0.1154
Light position MRP	0.0435	0.0094	-	-
Structured sampling	0.0372	0.0123	0.1863	0.0559

Table E.1: Absolute and relative mean error for various methods with cases where the area light is above the horizon (AH) or crossing the horizon (CH). Methods using MRP are presented in two versions based on their support of horizon case for solid angle calculation (H) or not. The sign "-" means "not applicable". .

Appendix F

Local light probe evaluation

In *Frostbite*, we support two types of local light probes: sphere and oriented bounding box. Code to handle parallax correction for these two cases is provided in Listing F.1 and Listing F.2.

```
float3 dominantR = getSpecularDominantDir(N, R, NdotV, roughness);
float2 intersections;

if (sphereRayIntersect(intersections, worldPos - spherePos, dominantR, sphereRadius))
{
    // Compute the actual direction to sample, only consider far intersection
    // No need to normalize for fetching cubemap
    float3 localR = (worldPos + intersections.y * dominantR) - spherePos;

    // We use normalized R to calc the intersection, thus intersections.y is
    // the distance between the intersection and the receiving pixel
    float distanceReceiverIntersection = intersections.y;
    float distanceSphereCenterIntersection = length(localR);

    // Compute the modified roughness based on the travelled distance
    float localRoughness = evaluateDistanceBasedRoughness(roughness,
        distanceReceiverIntersection, distanceSphereCenterIntersection);

    // Specular sampling
    // Limit artifacts introduce with high roughness
    localR = lerp(localR, MainR, linearRoughness);
    float4 result = evaluateIBLSpecular(NdotV, localR, localRoughness, f_0, f_90);
    specularResult = result.rgb;
    specularWeight = result.a;

    // Reflection.a contains fading information for reflection data not
    // available or barely accurate.  Here we want to manage a soft
    // transition at boundary of the sphere shape to avoid hard cutoff.
    // Decrease the sphere radius with the influence distance for the test
    float localDistance = length(worldPos - spherePos);
    float alpha = saturate((sphereRadius - localDistance) /
                max(fadeDistance, 0.0001));

    // Get local weight taking into account IBL alpha and receiver alpha
    // Smoothstep provide a nicer transition
    float alphaAttenuation = smoothstep(alpha);
    specularWeight *= alphaAttenuation;
}
```

Listing F.1: Sphere local light probe evaluation code for both specular and diffuse parts.

```
1
2  float3 dominantR = getSpecularDominantDir(N, R, NdotV, roughness);
3
4  // Perform Box collision to parallax correct the cubemap
5  // invTransform go from worldspace to local box space without scaling
6  float3 localPos = mul(float4(worldPos, 1), lightInvTransform).xyz;
7  float3 localDir = mul(dominantR, (float3x3)lightInvTransform);
8
9  float2 intersections = boxRayIntersect(localPos, localDir, -lightExtend, lightExtend);
10
11 // Specular contribution
12 specularResult = float3(0, 0, 0);
13 specularWeight = 0;
14
15 if (intersections.y > intersections.x)
16 {
17     // retrieve local intersection position (Localized cubemap are generated in local space not
            world space)
18     float3 localR = localPos + intersections.y * localDir;
19     // take into account offset (in local space) to retrieve the correct reflection vector
20     localR = localR - light.localOffset;
21
22     // We use normalized R to calc the intersection, thus "intersections.y" is the distance
            between the intersection and the receiving pixel
23     float distanceReceiverIntersection = intersections.y;
24     float distanceBoxCenterIntersection = length(localR);
25
26     // Compute the modified roughness based on the traveled distance
27     float localRoughness = evaluateDistanceBasedRoughness(data.roughness,
            distanceReceiverIntersection, distanceBoxCenterIntersection);
28
29     // Specular sampling
30     // Limit artifacts introduce with high roughness
31     localR = lerp(localR, MainR, linearRoughness);
32     float4 result = evaluateIBLSpecular(NdotV, localR, localRoughness, f_0, f_90);
33     specularResult = result.rgb;
34     specularWeight = result.a;
35
36     // Reflection.a contain fading information for reflection data not available or barely
            accurate
37     // Here we want to manage a soft transition at boundary of the box shape to avoid hard
            cutoff. Do it from the boundary of the box.
38     float boxPointDistance = distanceBoxPoint(light.influenceFadeDistance.xxx - light.extend,
            light.extend - light.influenceFadeDistance.xxx, localPos);
39     float alpha = 1.0f - saturate(boxPointDistance / max(light.influenceFadeDistance, 0.0001));
40
41     // Get local weight taking into account IBL alpha and receiver alpha
42     // Smoothstep provide a nicer transition
43     float alphaAttenuation = smoothstep_(alpha);
44     specularWeight *= alphaAttenuation;
45 }
```

Listing F.2: Box local light probe evaluation code for both specular and diffuse parts.