Implementing a 3D Rendering Engine with Physically-Based Shading

Ben Cole – Part III

Abstract—**Physically-based shading is a modern approach to shading in 3D graphics that uses physically-inspired models of light and surfaces to achieve enhanced realism over conventional rendering techniques. I explain the theory behind physically-based shading and implement a real-time renderer using C++ and OpenGL, presenting a novel algorithm for interpolating between multiple materials' bidirectional reflectance distribution functions (BRDFs) to combine their features in the process. The renderer achieves realistic graphics with accurate lighting and environment reflections.**

I. Introduction

Physically-based rendering (PBR) is an increasingly popular collection of techniques for producing realistic graphics in computer-generated imagery, used by Disney to create animated films [1] as well as Epic Games in its Unreal Engine for video games [2]. Historical real-time graphics techniques like the Phong shading model [3] were designed primarily for fast and efficient rendering, motivated by a constrained amount of hardware available in typical PCs and games consoles. In contrast, the intent of PBR is to leverage the dramatic increase in processing power seen in recent years to simulate lighting and surfaces accurately according to how they behave in the physical world. Physically-based *shading* is concerned specifically with determining how objects and materials respond to light.

Physically shaded surfaces are typically modelled using the *microfacet surface model* [4], which models surfaces as being mirror-like but containing tiny peaks and troughs controlled by some *roughness* parameter. These variations are too small to be seen individually, but their presence has a perceptible impact on the overall illumination of the surface: for example, a rough surface will experience *self-shadowing* at oblique angles [5].

A common feature of physically-based techniques is that they are *energy-conserving*, meaning that the amount of light released by a surface never exceeds the amount received. Using an energy-conserving *bidirectional reflectance distribution function* (BRDF) that factors in the surface's roughness, combined with lighting and reflection parameters empirically measured from real-world materials, PBR techniques can produce highly accurate and realistic-looking outputs.

In this paper, I explain and evaluate my implementation of a 3D rendering engine using physically-based shading techniques described by Epic Games [6] and a popular computer graphics textbook [7]. Additionally, I propose a new method for combining BRDFs to interpolate between their properties. One application would be to model arbitrary mixtures of materials with different reflectance properties.

The paper proceeds as follows. Section II summarises foundational techniques for 3D rendering and analyses recent research into PBR shading models. Section III explains at a high level the implementation of the renderer, and Section IV evaluates it, comparing its image quality against a state-of-the-art 3D rendering application. Finally, Section V draws conclusions about the state of the field and comments on the future of computer-generated graphics.

II. Related Work

A. The Reflectance Equation

The central task of any computer graphics library is to solve, or at least approximate as accurately as possible, the *rendering equation* [8]. The *reflectance equation* [5], a simplified version for static scenes with no emissive or absorptive surfaces, is shown in equation 1.

$$\underbrace{L_o(\boldsymbol{p}, \boldsymbol{\omega}_o)}_{\text{light out}} = \int_\Omega \underbrace{f_r(\boldsymbol{p}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o)}_{\text{BRDF}} \underbrace{L_i(\boldsymbol{p}, \boldsymbol{\omega}_i)}_{\text{light in}} \underbrace{(\boldsymbol{n} \cdot \boldsymbol{\omega}_i)}_{\cos\theta} \,\mathrm{d}\boldsymbol{\omega}_i \tag{1}$$

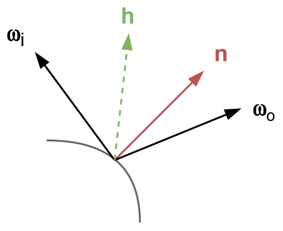
Intuitively, the equation says that the light leaving some point $\boldsymbol{p}$ on a surface in direction $\boldsymbol{\omega}_o$ should equal the total amount of radiance received from all incoming directions $\boldsymbol{\omega}_i$ in the hemisphere Ω, once the angle of incidence θ and BRDF of the surface have been factored in. Later I explain how this integral can be approximated in real-time using precomputed *lookup textures*.

B. Phong Reflection Model

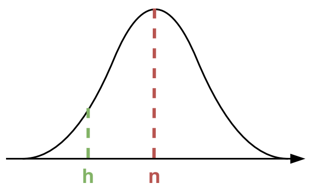
The *Phong reflection model* [3] is a simple shading method that was invented by B. Phong in 1975. The related *Blinn-Phong* reflection model [9], a faster approximation of it, remained the default shading model of OpenGL's fixed-function pipeline until 2009 [10]. The model describes the light emitted from a surface as three additive components: *ambient*, *diffuse* and *specular*, weighted by coefficients k_a, k_d and k_s respectively. The equation for the Phong shading model is given in equation 2: $\boldsymbol{v}$ is the vector to the viewer, and $\boldsymbol{n}$ is the surface normal; $\boldsymbol{l}_i$ denotes the unit vector pointing in the direction of light i, and $\boldsymbol{r}_i$ is the reflection of $\boldsymbol{l}_i$ in $\boldsymbol{n}$.

$$I = \underbrace{k_a I_a}_{\text{ambient}} + \sum_{i \in \text{lights}} (\underbrace{k_d(\boldsymbol{l}_i \cdot \boldsymbol{n}) I_i}_{\text{diffuse}} + \underbrace{k_s(\boldsymbol{r}_i \cdot \boldsymbol{v})^\alpha I_i}_{\text{specular}}) \tag{2}$$

The model bears three weaknesses that prevent it from producing convincingly real computer-generated graphics:



(a) The half-way vector $\boldsymbol{h}$



(b) A density function $D(\boldsymbol{h})$

Figure 1: Visualisation of a normal distribution function $D(\boldsymbol{h})$. Because of the presence of microscopic peaks and troughs in the surface, the local microfacet normal $\boldsymbol{h}$ may deviate from the macroscopic normal of the surface $\boldsymbol{n}$.

- The parameters—particularly the "shininess" parameter α—have no physical basis, so the results often seem artificial to a human.
- Even when $k_a + k_d + k_s = 1$, the model is not necessarily energy-conserving, amplifying the problem.
- Being controlled by a single scene-wide parameter I_a means the ambient illumination term lacks directionality.

My physically-shaded renderer improves on all of these. Its quantities and parameters are grounded in physically-interpretable units; the reflectance models employed are energy-conserving; and it uses *image-based lighting* to illuminate objects.

C. Cook-Torrance Microfacet Model

The *Cook-Torrance microfacet model* [4] is a physically-accurate BRDF used to implement the function $f_r(\boldsymbol{r}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o)$ from the reflectance equation described in Section II-A. It describes the shading of an object just in terms of diffuse and specular illumination, where the specular illumination is in turn composed of three multiplicative factors:

- The *normal distribution function* $D(\boldsymbol{h}; \alpha)$ describes a probability distribution over local surface normals $\boldsymbol{h}$. Because the surface is locally mirror-like, to sample D for light and view vectors $\boldsymbol{\omega}_i$ and $\boldsymbol{\omega}_o$ we use the half-way vector $\boldsymbol{h} = (\boldsymbol{\omega}_i + \boldsymbol{\omega}_o)/|\boldsymbol{\omega}_i + \boldsymbol{\omega}_o|$ because this is the only possible local normal that could provide the required reflection from $\boldsymbol{\omega}_i$ to $\boldsymbol{\omega}_o$. The probability density function $D(\boldsymbol{h})$ reveals the proportion of microfacets oriented with the required local normal for reflection from $\boldsymbol{\omega}_i$ to $\boldsymbol{\omega}_o$, and hence the proportion of rays from $\boldsymbol{\omega}_i$ that will be reflected in direction $\boldsymbol{\omega}_o$, as illustrated in Figure 1. A physical interpretation of the roughness parameter α is that it controls the width of the probability distribution.
- The *geometric attenuation function* $G(\boldsymbol{n}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i; \alpha)$ accounts for the effect of *self-shadowing*, in which the surface's microscopic peaks cause nearby troughs to be shadowed or obscured from view.
- Finally, the *Fresnel function* $F(\boldsymbol{h}, \boldsymbol{\omega}_o; \alpha)$ reports the surface's reflectivity as a function of the viewing angle and the material's base Fresnel reflectivity F_0.

The Fresnel term essentially describes the specular term k_s from the Phong model, except this time capturing more accurately that for real-world surfaces the quantity varies as the viewing angle changes. Setting $k_d = 1 - F$ to enforce the conservation of energy principle and denoting the surface's *albedo* as c, we obtain the final Cook-Torrance BRDF shown in equation 3[1]. Notice how the albedo is divided by π for physically-correct normalisation, unlike in the Phong reflection model.

$$f_r(\boldsymbol{p}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) = k_d \frac{c}{\pi} + \frac{DGF}{4(\boldsymbol{n} \cdot \boldsymbol{\omega}_i)(\boldsymbol{n} \cdot \boldsymbol{\omega}_o)} \tag{3}$$

I chose to implement the Cook-Torrance microfacet model in my renderer because it achieves good results and is representative of current industry-standard techniques. Furthermore, in Section III I provide a generalisation of the model that interpolates between multiple constituent functions D and G while remaining energy-preserving.

III. Method

Following is an explanation of how I implemented the renderer and justification of the high-level design decisions that I made. Later, I present intuitively then describe mathematically how the novel interpolation technique works. My implementation was written in C++ and OpenGL, using GLFW [11] for creating windows and managing user inputs. I employed *tinyobjloader* [12] for loading meshes due to its small size and high performance, and used the *STB Image Loading Library* [13] for texture loading for similar reasons.

A. Modelling of Materials

Materials are modelled using physically-interpretable properties. The *albedo* is the material's base colour when measured using a polarising filter to avoid specular reflections, inspired by [14]. *Roughness* ranges from 0, a perfect mirror, to 1, meaning totally matte. Rather than a binary quantity, I chose to permit the *metallic* level to vary continuously between 0 and 1 as well, so that users can capture partially-metallic materials like rusted iron. The F_0 quantity is the base Fresnel reflectivity: for this I provided a library of empirically-measured values for different materials [15].

B. Image-Based Lighting

Image-based lighting means using a *radiance map* to determine the amount of illumination from each angle. My renderer loads radiance maps from HDR texture files. As was discussed previously, solving the reflectance equation for each fragment in real-time is too expensive; instead, I precomputed certain textures and functions to OpenGL textures for querying at render-time.

[1]The original paper mistakenly divided the DGF term by π rather than the correct divisor of 4. This has been corrected in more recent publications, such as [5].

1) Diffuse: To see that the computation of Cook-Torrance diffuse illumination can be accelerated using precomputed textures, consider equations 4–6:

$$L_o(\boldsymbol{p}, \boldsymbol{\omega}_o) = \int_\Omega f_r(\boldsymbol{p}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) L_i(\boldsymbol{p}, \boldsymbol{\omega}_i)(\boldsymbol{n} \cdot \boldsymbol{\omega}_i) \,\mathrm{d}\boldsymbol{\omega}_i \tag{4}$$

$$= \int_\Omega \left(k_d \frac{c}{\pi} + \frac{DGF}{4(\boldsymbol{n} \cdot \boldsymbol{\omega}_i)(\boldsymbol{n} \cdot \boldsymbol{\omega}_o)} \right) \times L_i(\boldsymbol{p}, \boldsymbol{\omega}_i)(\boldsymbol{n} \cdot \boldsymbol{\omega}_i) \,\mathrm{d}\boldsymbol{\omega}_i \tag{5}$$

$$= \underbrace{k_d \frac{c}{\pi} \int_\Omega L_i(\boldsymbol{p}, \boldsymbol{\omega}_i)(\boldsymbol{n} \cdot \boldsymbol{\omega}_i) \,\mathrm{d}\boldsymbol{\omega}_i}_{\text{irradiance map}} + \underbrace{\int_\Omega \frac{DGF}{4(\boldsymbol{n} \cdot \boldsymbol{\omega}_i)(\boldsymbol{n} \cdot \boldsymbol{\omega}_o)} L_i(\boldsymbol{p}, \boldsymbol{\omega}_i)(\boldsymbol{n} \cdot \boldsymbol{\omega}_i) \,\mathrm{d}\boldsymbol{\omega}_i}_{\text{specular component}} \tag{6}$$

The *irradiance map* captures the quantity of irradiance received for each possible surface normal. Because it does not depend on ω_o, we can precompute it for each possible surface normal $\boldsymbol{n}$ and look this up at runtime. Lambertian diffuse functions (the dot product of $\boldsymbol{n}$ and $\boldsymbol{\omega}_i$) are computationally efficient yet achieve good-looking results [6]. Fortunately, since the irradiance map just looks like a blurred version of the radiance map, I found that storing it in a texture of only 16×16 texels gave satisfactory results. I used OpenGL's bilinear interpolation sampler to determine values between samples.

Many implementations, such as [7] and Epic Games' own, store such intermediate textures in a cubemap so that they can be accessed using Cartesian world-space coordinates. However, since the HDR radiance maps are encoded as a single rectangular texture in polar coordinates, I used a spherical polar coordinate representation throughout, avoiding the need to allocate additional memory for cubemap textures.

2) Specular: Precomputing the specular integral directly is more challenging because the value depends on ω_o as well as ω_i, meaning a 3D lookup texture would be needed. Instead, I used Epic Games' *split sum approximation* [6], shown in equation 7.

$$\int_\Omega \frac{DGF}{4(\boldsymbol{n} \cdot \boldsymbol{\omega}_i)(\boldsymbol{n} \cdot \boldsymbol{\omega}_o)} L_i(\boldsymbol{p}, \boldsymbol{\omega}_i)(\boldsymbol{n} \cdot \boldsymbol{\omega}_i) \,\mathrm{d}\boldsymbol{\omega}_i \approx \underbrace{\int_\Omega L_i(\boldsymbol{p}, \boldsymbol{\omega}_i) \,\mathrm{d}\boldsymbol{\omega}_i}_{\text{prefiltered environment map}} + \underbrace{\int_\Omega \frac{DGF}{4(\boldsymbol{n} \cdot \boldsymbol{\omega}_i)(\boldsymbol{n} \cdot \boldsymbol{\omega}_o)} (\boldsymbol{n} \cdot \boldsymbol{\omega}_i) \,\mathrm{d}\boldsymbol{\omega}_i}_{\text{BRDF specular lookup function}} \tag{7}$$

The *prefiltered environment map* is a texture like the irradiance map but this time factoring in the surface's roughness. The BRDF specular lookup function describes the *scale* and *bias* to be applied to the material's base F_0 Fresnel reflectivity value in terms of the material's roughness and the value of $\boldsymbol{n} \cdot \boldsymbol{h}$. Again, these are precomputed as textures then sampled at runtime so that real-time performance can be achieved[2].

[2]Exact details of the lookup texture are omitted due to space constraints: see the Epic Games paper [6] for more information.

C. Choice of Functions

The Cook-Torrance framework requires functions D, G and F. I implemented two pairs of D and G functions: the *Beckmann* NDF [16] and the *Cook-Torrance* geometry function in the original Cook-Torrance paper [17], and the now de-facto industry standard *Trowbridge-Reitz GGX* normal and geometry functions [18]. Research shows that long-tailed distributions such as GGX can approximate a wide range of real-world materials accurately [1, 17]. For the Fresnel term, I used *Schlick's approximation* [19], which has been empirically demonstrated not to introduce significant error [20].

D. Importance Sampling

I computed the integrals in fragment shaders using *Monte Carlo integration* (integration by random sampling). For the prefiltered environment map and BRDF lookup textures I generated sequences of paired pseudo-random uniformly distributed values using the *Hammersley sequence* [21], then mapped them to points on a hemisphere using *GGX importance sampling* [17]. Many points on the hemisphere will have a low probability density, especially for low-roughness surfaces, and hence contribute little to a probability-weighted integral over that hemisphere. Using importance sampling means that points with a greater PDF value are more likely to be selected, ensuring that most of the points sampled are ones that contribute strongly to the result and hence converging to an acceptable accuracy in fewer iterations.

E. Interpolation of BRDFs

Rather than forcing the user to choose a single normal distribution function and geometry function, my renderer permits them to use a linear combination of multiple, so long as the weights all fall within the range $[0, 1]$ and sum to unity. Intuitively, this is akin to rolling a die to select which probability distribution to sample: the result is a valid probability distribution over the union of the two domains, and hence remains energy-conserving. With this, users can interpolate between arbitrary D and G functions using the formulas given in equations 8 and 9. In the special case with only one D_i function and only one G_i function, one obtains the original Cook-Torrance BRDF.

$$D(\boldsymbol{h}) = \sum_i w_i D_i(\boldsymbol{h}) \tag{8}$$

$$G(\boldsymbol{n}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) = \sum_j w_j G_j(\boldsymbol{n}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) \tag{9}$$

For simplicity I used GGX importance sampling regardless of the ratios of D and G functions selected. This does not impede correctness, since importance sampling only accelerates Monte Carlo integration, but may require more samples for functions that deviate substantially from the GGX ones. Fortunately, the GGX distribution has long tails, so all representable values retain some chance of being sampled.

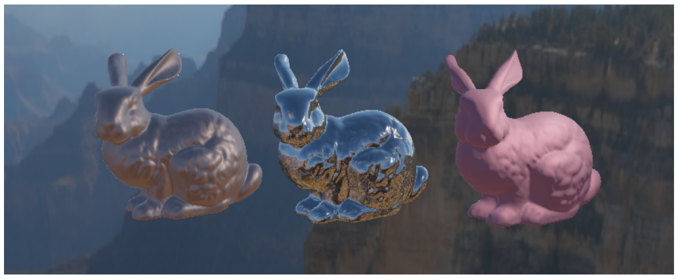


Figure 2: Stanford Bunnies rendered as rough copper (roughness 0.4), smooth silver (roughness 0.1), and matte plastic (roughness 0.7) using the Trowbridge-Reitz GGX functions.

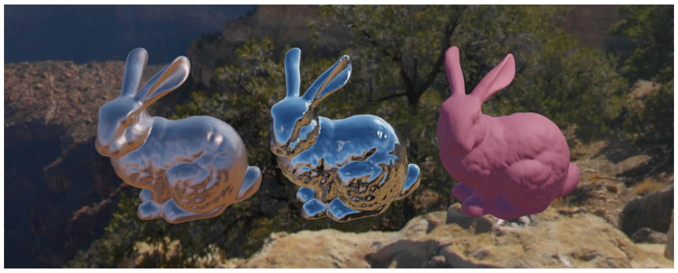


Figure 3: The scene from Figure 2 reproduced in Blender.

IV. Evaluation

I evaluate the renderer in three ways: I compare its outputs against those of Blender, a popular 3D modelling and rendering application [22]; I quantify its performance on a modern laptop, and finally I demonstrate the effectiveness of the BRDF interpolation algorithm. To evaluate the renderer I used HDR environment maps available for free online [23].

A. *Quality of Output*

To compare the rendering performance across a range of materials, I created a scene using three Stanford bunnies [24], shown in Figure 2, and a reference scene rendered using Blender, shown in Figure 3. The output from my renderer is lifelike, with believable lighting on all three meshes and accurate reflections from the silver bunny. Unlike Blender, my renderer has no inter-object shadowing, and its lack of anti-aliasing leaves jagged edges visible at object boundaries. The red bunny unfortunately also seems less vibrant than in the reference image, which I suspect to be due to a difference in tone-mapping that I was unable to diagnose. However, even without computing physically-based light *transport* like Blender does, my render clearly produces accurate outputs.

Figure 4 shows the effect of varying a surface's roughness. The fact that energy is conserved is apparent: concentrating specular illumination into a smaller region leaves the rest of the surface darkened, as would be expected for a constant level of irradiance. This further contributes to the overall effect of realistic and believable lighting.

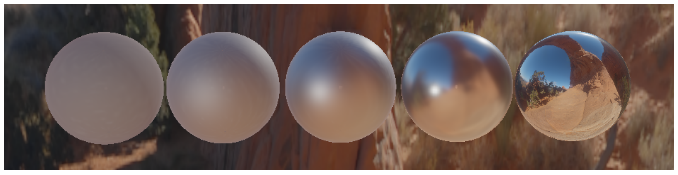


Figure 4: Spheres of decreasing roughness, rendered using the Trowbridge-Reitz GGX normal distribution function.

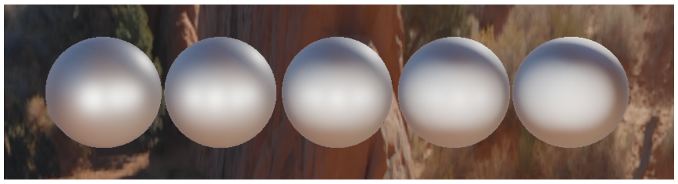


Figure 5: Interpolating between the GGX functions, left, and Beckmann/Cook-Torrance functions, right, with roughness 0.5.

B. *Performance*

The engine maintains 60 frames per second at a resolution of 1920×1080 on my laptop using an integrated Intel Iris Plus 640 GPU, which would more than suffice for gaming.

C. *Effectiveness of BRDF Interpolation*

Figure 5 shows an interpolation between the Trowbridge-Reitz GGX and Beckmann/Cook-Torrance BRDFs. The GGX distribution has brighter but more localised specular highlights, demonstrating again that the interpolation technique and underlying distributions are correctly energy-preserving. The algorithm successfully interpolates between different distributions, allowing artists to combine aspects of multiple BRDFs for finer control of their materials.

D. *Possible Improvements*

Due to time constraints, the renderer does include any anisotropic normal distribution or geometric attenuation functions, so it cannot render materials with anisotropic roughness levels (where the horizontal roughness differs from the vertical roughness) such as brushed metal. Additionally, I would have liked to have implemented texture-based materials so that materials are applied per-vertex rather than per-mesh. That said, the renderer already achieves realistic-looking results and effectively demonstrates the proposed interpolation technique.

V. Conclusions

Physically-based shading clearly provides more accurate and realistic-looking graphics than early shading models. Epic Games' work demonstrates the merit of committing to physically-interpretable units, as well as the efficacy of moving computational work out of the critical path. Next-generation rendering engines will likely continue to follow physically-based workflows, with improvements to hardware capabilities facilitating ever more precise modelling of how light behaves in the real world. Physically-based rendering techniques mark an exciting new era for computer-generated graphics.

References

[1] Brent Burley. "Physically-Based Shading at Disney". In: 2012.

[2] *Physically Based Shading in Unreal Engine 4*. 2014. URL: https://www.unrealengine.com/en-US/blog/physically-based-shading-in-ue4.

[3] Bui Tuong Phong. "Illumination for Computer Generated Pictures". In: *Commun. ACM* 18.6 (June 1975), pp. 311–317. ISSN: 0001-0782. DOI: 10.1145/360825.360839. URL: https://doi.org/10.1145/360825.360839.

[4] 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: https://doi.org/10.1145/357290.357293.

[5] Matt Pharr, Jakob Wenzel, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2017.

[6] Brian Karis. *Real Shading in Unreal Engine 4*. Epic Games, 2013. URL: https://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf.

[7] Joey de Vries. *Learn OpenGL: Learn modern OpenGL graphics programming in a step-by-step fashion*. Kendall & Welling, 2020.

[8] James T. Kajiya. "The Rendering Equation". In: *SIGGRAPH Comput. Graph.* 20.4 (Aug. 1986), pp. 143–150. ISSN: 0097-8930. DOI: 10.1145/15886.15902. URL: https://doi.org/10.1145/15886.15902.

[9] James F. Blinn. "Models of Light Reflection for Computer Synthesized Pictures". In: *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '77. San Jose, California: Association for Computing Machinery, 1977, pp. 192–198. ISBN: 9781450373555. DOI: 10.1145/563858.563893. URL: https://doi.org/10.1145/563858.563893.

[10] Dave Shreiner. *OpenGL programming guide*. 7th ed. Addison-Wesley, 2010.

[11] GLFW - An OpenGL Library. URL: https://www.glfw.org/.

[12] tinyobjloader. URL: https://github.com/tinyobjloader/tinyobjloader.

[13] Sean Barrett. *stb library*. URL: https://github.com/nothings/stb.

[14] URL: https://docs.unrealengine.com/en-US/RenderingAndGraphics/Materials/PhysicallyBased/index.html.

[15] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering, Third Edition, 3rd Edition*. 3rd ed. A K Peters/CRC Press, 2008.

[16] Petr Beckmann. "II Scattering of Light by Rough Surfaces". In: ed. by E. Wolf. Vol. 6. Progress in Optics. Elsevier, 1967, pp. 53–69. DOI: https://doi.org/10.1016/S0079-6638(08)70579-1. URL: http://www.sciencedirect.com/science/article/pii/S0079663808705791.

[17] Bruce Walter et al. "Microfacet Models for Refraction through Rough Surfaces". In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR'07. Grenoble, France: Eurographics Association, 2007, pp. 195–206. ISBN: 9783905673524.

[18] T. S. Trowbridge and K. P. Reitz. "Average irregularity representation of a rough surface for ray reflection". In: *J. Opt. Soc. Am.* 65.5 (May 1975), pp. 531–536. DOI: 10.1364/JOSA.65.000531. URL: http://www.osapublishing.org/abstract.cfm?URI=josa-65-5-531.

[19] C. Schlick. "An Inexpensive BRDF Model for Physically-based Rendering". In: *Computer Graphics Forum* 13 (1994).

[20] Addy Ngan, Frédo Durand, and Wojciech Matusik. "Experimental Analysis of BRDF Models". In: *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques*. EGSR '05. Konstanz, Germany: Eurographics Association, 2005, pp. 117–126. ISBN: 3905673231.

[21] J. M Hammersley and D. C Handscomb. *Monte Carlo Methods*. Springer Netherlands, 1964.

[22] Ton Roosendaal. *Blender*. URL: https://www.blender.org.

[23] *sIBL Archive: Free HDRI sets for smart Image-Based Lighting*. 2021. URL: http://www.hdrlabs.com/sibl/archive.html.

[24] Greg Turk and Marc Levoy. *The Stanford 3D Scanning Repository*. 2021. URL: http://graphics.stanford.edu/data/3Dscanrep/#bunny.