Bridging the Gap between Offline and Real Time with Neural Materials

ANDREA WEIDLICH, NVIDIA

1 INTRODUCTION

In recent years, significant advancements in rendering algorithms have been made. For example, path tracing, once limited to offline rendering due to its computational intensity, is now feasible in real-time applications and games. This leap has greatly enhanced visual realism and simplified artist workflows, allowing for more physically accurate lighting.

However, material models in real-time rendering have not seen the same level of progress. While offline renderers have adopted increasingly complex and physically based material representations, real-time engines still mostly rely on simplified models. The typical real-time material consists of a diffuse term, a single GGX microfacet [Wal+07] reflection lobe, and sometimes an additional "sheen" component [ZBC22] for edge reflections. To blend between dielectrics (non-metals) and conductors (metals), real-time pipelines often use "metalness", an artificial parameter that has no physical meaning, as it interpolates between two fundamentally different material types.

Without realistic and robust material models, improvements in light transport algorithms alone cannot achieve truly photorealistic results. Accurate simulation of light is only as effective as the fidelity of the surfaces it interacts with. Therefore, advances in material modeling are essential for the next leap in rendering realism.

2 FILM-QUALITY VS. GAMES-QUALITY MATERIAL SYSTEMS

Real-time and offline material systems are often perceived as distinct, however, the boundary between them is not well defined. Both share many fundamental requirements, and their core design is guided by similar principles:

- *Flexibility.* A material system must be capable of representing a broad spectrum of visual complexities. A typical scene may include hero characters requiring detailed materials, as well as background characters and environments with varying demands.
- *Storage Efficiency.* Realistic appearance models frequently require significant precomputation and data storage. However, practical constraints preclude the use of lengthy precomputation or large per-vertex or per-asset storage footprints.
- *Creative Expression.* While physically based material models enforce principles such as energy conservation and Helmholtz reciprocity, production workflows often require the depiction of appearances that do not occur in nature, or the adjustment of materials for artistic purposes. Therefore, while striving for physical plausibility, a material system must also allow deviations from strict physical correctness to enable creative expression.

Part of *Physically Based Shading in Theory and Practice*, SIGGRAPH 2025 Course.

- *Performance.* Both real-time and offline rendering are subject to computational constraints, albeit on different scales. Material systems must be designed to permit selective activation of effects based on available computational resources, and must prioritize efficient sampling and evaluation.

Although real-time and offline material systems are governed by the same core design considerations, their primary differences arise from available computational budgets. As such, they differ in two aspects: the complexity of the shader code and the input data.

2.1 BSDF Models and Layering

At the BSDF level, many of the fundamental components are shared between real-time and offline material systems. Offline renderers often support a wider variety of BSDFs, ranging from a handful (as in the Disney Principled BSDF [Bur12]) to hundreds of specialized models (as discussed in the Manuka renderer paper [Fas+18]). But despite this larger selection, in practice both real-time and offline workflows typically rely on only a smaller set of core BSDFs for most materials. For example, Oren–Nayar [ON94] and GGX [Wal+07] are BSDFs that can be commonly found in both systems. However, the main distinction lies in how these BSDFs are combined within materials. In real-time systems, material definitions are usually constrained by "uber-shaders" with a hard-coded combination of BSDFs—often limited to a single diffuse, a single GGX, and optionally a sheen layer. This approach is dictated by the need to avoid code divergence and the goal of efficient computation. While they are good at simulating certain materials such as ceramics, paint or rust, others are hard to achieve. Examples include fabric, organic materials such as skin and hair, or granular materials.

Offline renderers, in contrast, have for years supported arbitrary and physically plausible combinations of BSDFs and emission distribution functions (EDFs) through flexible layering systems [Wei19]. This allows for more complex and realistic material appearances, as well as more intuitive workflows rooted in physics.

Layering enables the combination of multiple BSDFs using operations such as mixing and coating, which allows for the creation of complex effects like clear coats and stains, and blending materials in a more natural way though blend weights instead of blending on the parameter level. Unlike parameter-level blending, layering allows each BSDF to retain its own set of textures, such as albedo and normal maps. In addition to supporting diverse material effects, layering is also used to improve the overall shape and behavior of a BSDF.

2.2 Textures

In offline rendering, the use of UDIMs (U-DIMension) is standard practice for handling high-resolution textures. UDIMs allow a single asset to be covered by an array of texture tiles, each corresponding to a unique region in UV space. This approach enables artists to assign hundreds of 4K or 8K textures to large assets, ensuring fine details and surface imperfections are accurately captured across the entire model. As a result, much of the realism in offline production is driven by the extensive use of detailed texture maps.

In contrast, real-time rendering is constrained by bandwidth and memory limitations, making such high-resolution, asset-specific texturing impractical. Instead, real-time pipelines often rely on procedural techniques to blend multiple lower-resolution textures or tile generic detail maps, rather than using unique, high-resolution textures for each asset. While this approach is efficient, it can make it challenging to achieve the same level of fine detail and realism as offline rendering, as texture resolution is the primary limiting factor.

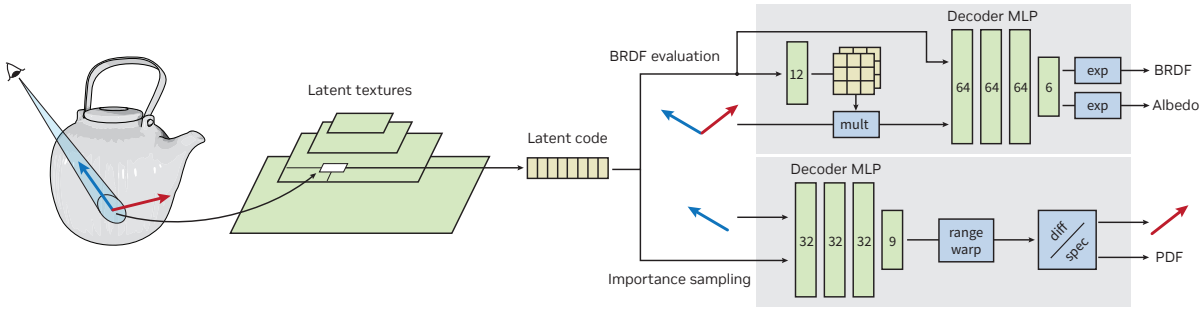


Figure 1: We integrate our neural BRDFs into the renderer as follows: for each ray that intersects a surface with a neural BRDF, we compute the standard texture coordinates and use these to query the latent textures of the neural material. The resulting latent code is then passed into one or two neural decoders, depending on the requirements of the rendering algorithm. The BRDF or evaluation decoder (top box) predicts the BRDF value and optionally the directional albedo. The importance sampler (bottom box) derives parameters for an analytical two-lobe distribution. See Zeltner et al. [Zel+24] for more information.

3 NEXT-GEN NEURAL MATERIALS

This talk addresses a central challenge: how to bring the rich complexity and variation of offline-rendered materials into real-time rendering. Traditional approaches, i.e. relying on intricate shader code and large texture sets, are impractical. Neural materials offer a promising alternative, encoding appearance efficiently through compact neural representations and bypassing many of the traditional performance bottlenecks.

3.1 Neural Materials Architecture

In academic research, neural materials [Szt+21; Kuz+21; Zel+24] were introduced as a new way of capturing appearances. The goal is not to improve the performance of analytical shading models but to convert them into a different and more efficient representation. As such, neural materials represent a new paradigm in material modeling, leveraging neural networks to approximate complex appearance functions. Unlike traditional techniques, which rely on explicit shader code and extensive sets of textures, neural materials encode the entire behavior of a material within a trained neural network and thus can bypass expensive shader code and texture evaluation.

The core theory behind neural materials is to represent the mapping from surface properties and lighting/viewing directions to outgoing radiance using a compact, learned model. In our work, we use a small MLP with an encoder-decoder structure. The encoder maps BSDF parameters to a latent space, i.e., it uses a set of BSDF parameters from the reference material as input and converts them into a multi-channel latent texture (we commonly use 8 channels).

This additional encoder network and the original list of BRDF parameters only need to be available during the optimization. Afterwards, we can still explicitly generate a set of latent textures to use during inference. During rendering, the latent textures, which are fetched and read like traditional textures, are decoded with the evaluation decoder network that infers BRDF values for a given pair of directions. In addition to the evaluation decoder, we also have a sampling decoder, which is used to determine the next direction of a ray so that the neural material can be used inside a path tracer. The sampling network reverts to a simplified diffuse + GGX representation for importance sampling. A schematic illustration of the process can be seen in Figure 1.

Training happens sequentially. We first train the evaluation decoder, and in a second step we use the sampling decoder. The number of training iterations depends on the type of material and typically ranges from 100K to 500K. For the sampling decoder, we use a much smaller number (usually around 20–50K) since it doesn't have to be as accurate.

3.2 Decoder Size Quality Control

By changing the size of the evaluation decoder, we can automatically adjust the performance and quality of a trained material. We commonly have three different configurations: 2x16 as the smallest, 2x32 as the middle and 3x64 as the largest with the highest quality. The first number indicates the number of hidden layers while the second one is their width.

An example can be seen in Figure 2. As reference material we chose one that was created with MaterialX [SS16] and composed of 9 different BSDFs. The asset has 6 UDIMs and reads data from 6 different textures (i.e., 36 textures in total) plus two detail textures; each texture is 4K resolution. The smallest network is 3.5x faster in overall render time than the reference while the largest network is on average 1.21x faster. The scene is rendered at 2K and fully path traced in Falcor, our research renderer. For comparison, an untextured asset with a specialized material of just a diffuse and a single GGX takes 1.26 ms whereas the 2x16 network takes 1.97 ms. This makes the smallest neural configuration just 1.5x slower than an untextured, grey-shaded asset.

Note that we can choose any width and number of layers. The configurations used here are just examples.

(a) Reference (b) 3x64 (c) 2x32 (d) 2x16

Figure 2: The decoder size will influence performance and quality. While the quality goes down with smaller configurations (in particular the 2x16 version loses some of the color at grazing angles), performance improves. The render time for the reference image is 7.03 ms and goes down to 5.78 ms, 2.69 ms and 1.97 ms for the respective sizes. Model and textures created by Alex Liu and Zhelong Xu.

4 NEURAL MATERIALS IN UNREAL ENGINE

For a recent hardware launch, we tested the feasibility of neural materials and implemented them as part of Unreal Engine [Epi24].

4.1 Reference Data

The reference materials were created in Substrate [Epi25], Unreal's layered material system. Unlike traditional real-time material systems, Substrate allows artists to build layered materials with slabs, which in turn are tiny uber-shaders consisting of two GGXs, one diffuse and a sheen BRDF. Slabs can be combined with mixing and coating operations and can simulate sophisticated effects such as roughness accumulation and even scattering between layers. For our task, we disabled all performance-related automatic material simplifications and directed our artists to produce materials without considering the run-time performance of the material.

We ended up with 22 materials, which we split between hero and background materials. Our hero materials were designed to showcase a wide range of different materials, impossible to create with current-gen game materials. We had five hero materials: a star sapphire with a strong view-dependent highlight; a tarnished silver with thin-film effects; a shot silk where warp and weft were weaved with differently colored threads, resulting in a blue-green view-dependent anisotropy; a silk rope with anisotropic highlights; and wood, where the internal fibers produce shifting highlights. The materials had between three and five slabs. We used a combination of vertical and horizontal operators to create the materials.

Furthermore, we had 17 background materials. These were much simpler materials that could be created with current-gen material models. They were implemented as material instances that utilized the same shader code. Compared to the hero materials, they had a limited number of input textures, namely albedo, normal and ORM[1] maps. However, both hero and background assets had a large number of UDIMs and 4K textures.

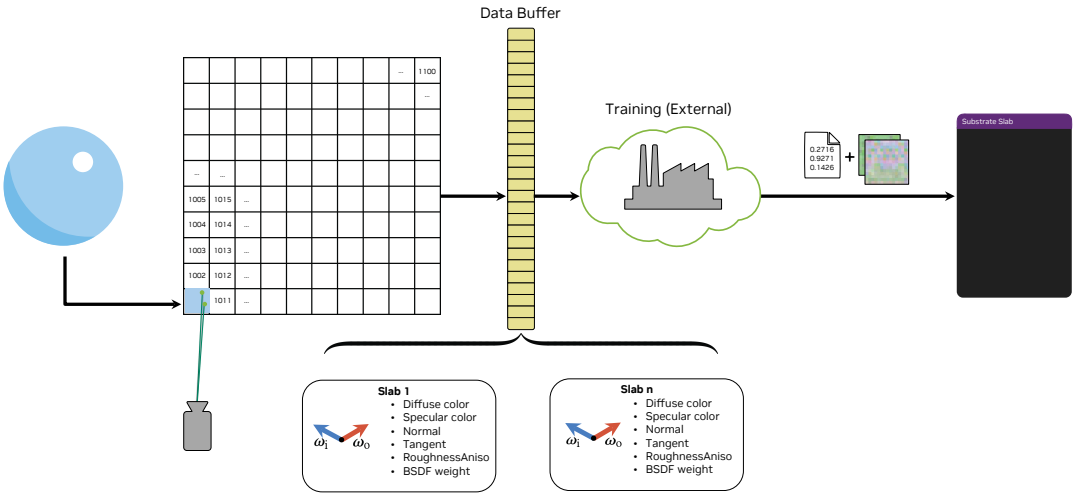


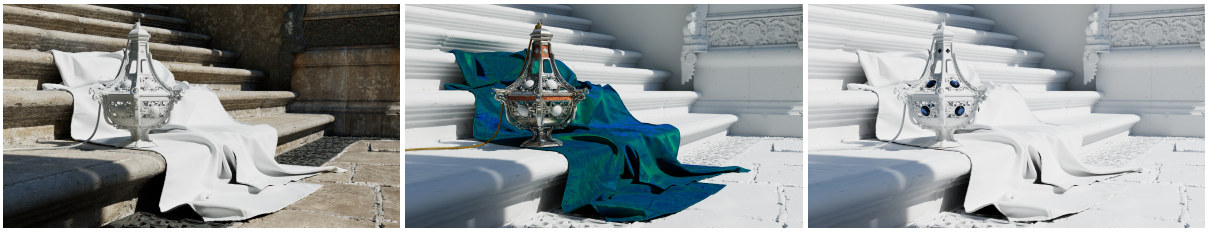
Figure 3: Overview of our training pipeline. We train a material on a plane, independent from the geometry. The data buffers are exported to an external training framework, and the training results are brought back into Unreal.

4.2 Training Pipeline

We trained each material separately and independent from the geometry. While ideally one can train on the geometry directly, we instead opted to train on a single plane for simplicity and decided to bake geometry-dependent parameters into textures. Our plane could handle training of up to 100 UDIM tiles. The camera was positioned above the plane and would intersect the plane at a specific point, but an arbitrary ray pair was evaluated on that point. This setup allowed us to easily query Unreal's material system.

Once generated, the material evaluation data and its guide buffers were handed over to the training framework, which is part of Falcor. To help the network identify features, we supported several buffers per slab: diffuse and specular albedo, normal map, tangents and roughness, which we could extract from the material system. Slabs

[1] A texture containing occlusion, roughness and metallic values in R, G and B, respectively.



(a) Background materials: 2x16 (b) Hero set 1: 2x32 (c) Hero set 2: 3x64

Figure 4: We split our materials into three groups. The background materials were simple enough to be captured by a 2x16 network. The hero materials were split into two groups: wood, fabric, rope and metal could be captured with a 2x32 decoder, but for the gemstones we needed a 3x64 decoder.

were chosen sequentially until we iterated through all of them. An illustration of our training pipeline can be seen in Figure 3.

We used a different number of training samples per material: background, wood and rope used 100K iterations; metal and fabric used 200K; gemstone, as the most complex material, used 500K. Other training parameters stayed mostly the same. Our training time for a single material was generally fast. A single material can be trained within minutes whereas data generation is normally the bottleneck. However, it should be noted that for materials where we also trained LODs (see Section 4.3), the numbers were significantly higher. For LOD training, we need to sample and train all appearance levels simultaneously so that the material accurately filters across distance. This is necessary to avoid noise that would appear if we would use high resolution latent textures on objects further away from the camera. Alternatively, if we would filter the latent texture similarily to what one can do with e.g. diffuse textures, the result would not be correct anymore. Instead, we increase the footprint of the samples and accumulate the results. The network trains from all distance levels at the same time and stores a latent texture for each mipmap level. These textures are querried during rendering like normal mipmap textures. Note that we only trained LODs where necessary. The most expensive material to train was the fabric due to its 26 UDIMs and need for LODs due to its fine structure. Training took several hours.

The network size was determined by the expected quality of the materials. We opted to use the smaller 2x16 network size for the background materials (Figure 4a). They were simple enough that we didn't lose quality. For the hero materials, we chose two different network sizes (Figures 4b and 4c), 2x32 and 3x64. Only the gemstones needed the largest network size to retain quality since the extremely narrow anisotropic highlights were hard to train with smaller networks.

Once training was finished, we could import the neural material back into UE by loading the weights and the two latent textures which were stored as int8. A new material type was created within Unreal in the Substrate pipeline to support neural materials.

4.3 Results and Lessons Learned

Our target was to render our scene at 4K and 60 FPS. Therefore, we implemented neural materials into the rasterizer pipeline and used Lumen for global illumination. All fabrics were trained with several LOD levels. This was necessary because we wanted to be able to capture the appearance of the fabric across different distances without introducing artifacts from mipmapping. As it can be seen in Figure 5, overall, we were able to match

(a) Reference Substrate materials

(b) Neural materials

Figure 5: Reference materials and their neural variants. While differences can be seen between reference and neural materials, overall we managed to match appearance while greatly reducing render time and memory requirements.

appearance to the ground truth while greatly improving memory footprint and performance.

Since latent textures are considered to be regular textures, we could compress them to reduce texture memory. We experimented with different texture compression algorithms but decided in the end against BC7 and in favor of NTC [Vai+23] to avoid block artifacts. In the final demo, the neural materials used 5x less memory compared to the reference materials.

By integrating neural materials into the rasterizer pipeline of Unreal Engine, we showed that the technology, which was originally developed for path tracing, is compatible with a rasterizer infrastructure and can be used in a commercial product. However, during integration we learned several valuable lessons important for productization.

For example, while path tracing does not need a solution for pre-convolution of IBLs and area lights, rasterizers have different requirements. Due to limited time in the development of the demo, we opted to revert to a simple diffuse + GGX model for the environment and ray-tracing passes and took the quantities (roughness and diffuse-specular weight) from the sampling network. Diffuse and specular albedos were not readily available and needed to be trained from the input materials. Moreover, the denoiser and other parts of the renderer rely on having "traditional" render data such as normal and roughness maps available. For multi-layered materials, it is not immediately obvious how this data can be represented and more research needs to be done to explore this area.

One more pain point was that we used the path-tracing pipeline to generate our reference data since it was far easier to implement the training there. During training, we noticed that the path-tracing material system and the rasterizer material system showed slight differences when it came to the handling of shading normals. These differences needed to be eliminated in order to match the reference materials.

(a) Unreal Engine Lumen rasterizer

(b) Falcor path tracer

Figure 6: The same asset rendered in a rasterizer and a path tracer, with two different lighting directions. Note that the remaining differences lie in the light transport – particularly indirect bounces, which are handled quite differently.

5 NEURAL MATERIALS AS MATERIAL EXPORTER

Converting expensive Substrate materials into a more efficient format makes it possible to render film-quality in real time, but it also opens up another new and exciting avenue: neural materials allow us to easily port appearances between applications.

Porting materials between renderers is always a painful and tedious process since it is time-consuming to understand implementations, and features can get lost. Neural materials, on the other hand, do not store input data like shader parameters but the entire evaluation of a shading graph. As such, they encompass an entire material system of a renderer without revealing the underlying implementation. This makes them an ideal tool for sharing assets between different renderers, or even between vendors, without sharing proprietary code.

To test this capability, we imported the trained Substrate materials into Falcor. Despite the fact that Falcor has no knowledge of Substrate, we could port the materials by simply loading the trained materials. We used the same data as we had within Unreal without modification, a set of weights for evaluation, and a sampling decoder and two latent textures per material. The result can be seen in Figure 6.

It should be noted that we used Lumen in Unreal while Falcor utilizes a path tracer. While we can use the same neural materials in both renderers, we don't have control over the lighting approximations. In particular, indirect bounces behave differently.

6 CONCLUSION

Neural materials present exciting possibilities for bringing film-quality material appearance to real-time rendering. Currently, their primary use case is for high-detail hero assets, but with ongoing research, neural materials have the potential to even replace traditional analytic materials during rendering. Moreover, neural materials offer significant advantages as a universal material representation, allowing asset exchange between different renderers without sharing source code or input data. This dual capability not only bridges the gap between offline and real-time rendering but also paves the way for more integrated and efficient workflows across the industry.

One limitation of neural materials is that they are currently a baking solution. Latents cannot be edited directly, and a latent space constrains what edits can be done in principle. However, we still want to highlight that neural materials can be used not only alongside analytical materials, but even with analytical materials together in e.g. a layered material approach. In the future, we hope to see more editing possibilities while maintaining the performance and the quality of our materials.

ACKNOWLEDGMENTS

The neural materials project and the CES demo would not have been possible without countless people. In particular, we would like to mention (in alphabetical order) Fabrice Rousselle, Jan Novak, Petrik Clarberg, Tizian Zeltner, Wessam Bahnassi and Yaobin Ouyang for integrating neural materials into Unreal Engine. Alexey Panteleev, Gautam Ramakrishnan, Jeff Bolz and Justin Holewinski for integrating NTC, performance optimizations and driver support, and Bill Xu and Peter Thacker for build support. We thank Alex Liu, Andrei Krapyvchenko, Boon Cotter, Matteo Mingozzi, Pierre Fleau for creating the artwork and Aaron Lefohn, Alex Dunn, Chris Perrella, Gabriele Leone, Jaako Haapasalo and Marc-Andre Carbonneau for coordinating the effort.

REFERENCES

[Bur12] B. Burley. "Physically Based Shading at Disney". In: *SIGGRAPH Courses: Practical Physically Based Shading in Film and Game Production*. 2012.

[Epi24] Epic Games. *Unreal Engine 5.5*. 2024. URL: https://www.unrealengine.com/en-US (visited on 07/28/2025).

[Epi25] Epic Games. *Substrate Materials*. 2025. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/substrate-materials-in-unreal-engine (visited on 07/28/2025).

[Fas+18] L. Fascione, J. Hanika, M. Leone, M. Droske, J. Schwarzhaupt, T. Davidovič, A. Weidlich, and J. Meng. "Manuka: A Batch-Shading Architecture for Spectral Path Tracing in Movie Production". In: *ACM Trans. Graph.* 37.3 (Aug. 2018). ISSN: 0730-0301. DOI: 10.1145/3182161. URL: https://doi.org/10.1145/3182161.

[FXG14] FXGuide. *UDIM UV Mapping*. 2014. URL: https://www.fxguide.com/fxfeatured/udim-uv-mapping/.

[Jak+19] W. Jakob, A. Weidlich, A. Beddini, R. Pieké, H. Tang, L. Fascione, and J. Hanika. "Path tracing in production: part 2: making movies". In: *ACM SIGGRAPH 2019 Courses*. SIGGRAPH '19. Los Angeles, California: Association for Computing Machinery, 2019. ISBN: 9781450363075. DOI: 10.1145/3305366.3328085. URL: https://doi.org/10.1145/3305366.3328085.

[Kuz+21] A. Kuznetsov, K. Mullia, Z. Xu, M. Hašan, and R. Ramamoorthi. "NeuMIP: multi-resolution neural materials". In: *ACM Trans. Graph.* 40.4 (July 2021). ISSN: 0730-0301. DOI: 10.1145/3450626.3459795. URL: https://doi.org/10.1145/3450626.3459795.

[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: Association for Computing Machinery, 1994, pp. 239–246. ISBN: 0897916670. DOI: 10.1145/192161.192213. URL: https://doi.org/10.1145/192161.192213.

[SS16] D. Smythe and J. Stone. *MaterialX: An Open Standard for Network-Based CG Object Looks*. 2016. URL: https://materialx.org.

[Szt+21] A. Sztrajman, G. Rainer, T. Ritschel, and T. Weyrich. "Neural BRDF Representation and Importance Sampling". In: *Computer Graphics Forum* 40.6 (2021), pp. 332–346. DOI: https://doi.org/10.1111/cgf.14335. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14335. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14335.

[Vai+23] K. Vaidyanathan, M. Salvi, B. Wronski, T. Akenine-Moller, P. Ebelin, and A. Lefohn. "Random-Access Neural Compression of Material Textures". In: *ACM Trans. Graph.* 42.4 (July 2023). ISSN: 0730-0301. DOI: 10.1145/3592407. URL: https://doi.org/10.1145/3592407.

[Wal+07] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. "Microfacet Models for Refraction through Rough Surfaces." In: *Rendering techniques* 2007 (2007), 18th.

[Wei19] A. Weidlich. "Production Quality Materials". In: *SIGGRAPH Courses: Path Tracing in Production Part 2: Making Movies*. 2019. URL: https://doi.org/10.1145/3305366.3328085.

[ZBC22] T. Zeltner, B. Burley, and M. J.-Y. Chiang. "Practical Multiple-Scattering Sheen Using Linearly Transformed Cosines". In: *ACM SIGGRAPH 2022 Talks*. SIGGRAPH '22. Vancouver, BC, Canada: Association for Computing Machinery, 2022. ISBN: 9781450393713. DOI: 10.1145/3532836.3536240. URL: https://doi.org/10.1145/3532836.3536240.

[Zel+24] T. Zeltner*, F. Rousselle*, A. Weidlich*, P. Clarberg*, J. Novák*, B. Bitterli*, A. Evans, T. Davidovič, S. Kallweit, and A. Lefohn. "Real-time Neural Appearance Models". In: *ACM Trans. Graph.* 43.3 (June 2024). ISSN: 0730-0301. DOI: 10.1145/3659577. URL: https://doi.org/10.1145/3659577.

REVISION HISTORY

- v1.0 (**Aug 10, 2025**) – Initial version.
- v1.1 (**Dec 23, 2025**)
  - Added reference for “flexible layering systems” to Section 2.1.
  - Clarified performance timings in Section 3.2.
  - Added more detail regarding LODs to Section 4.2.
  - Minor rewording for clarity elsewhere.